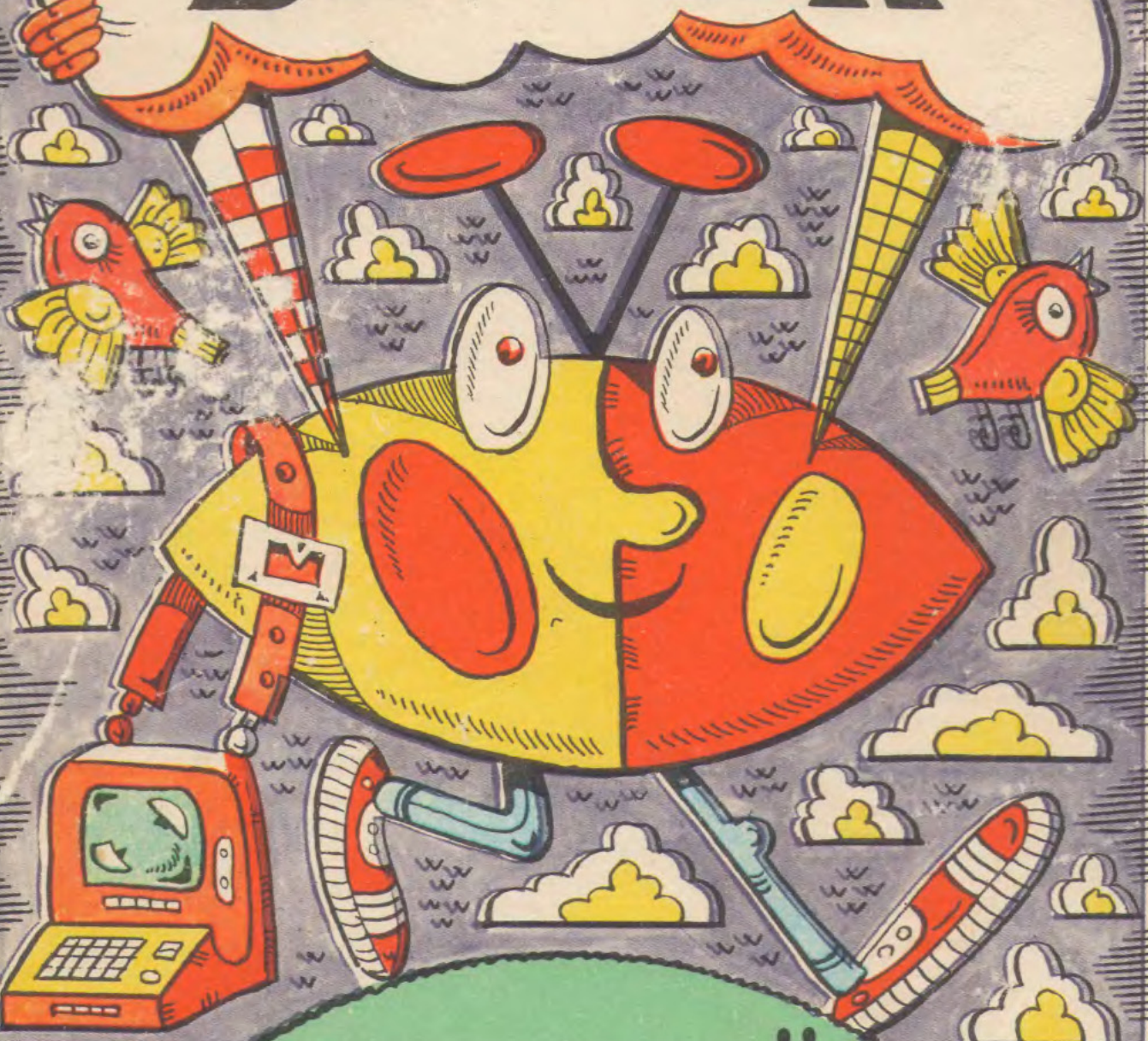
С. ВАТТ
М. МАНГАДА

БЕЙСИК



ДЛЯ ДЕТЕЙ

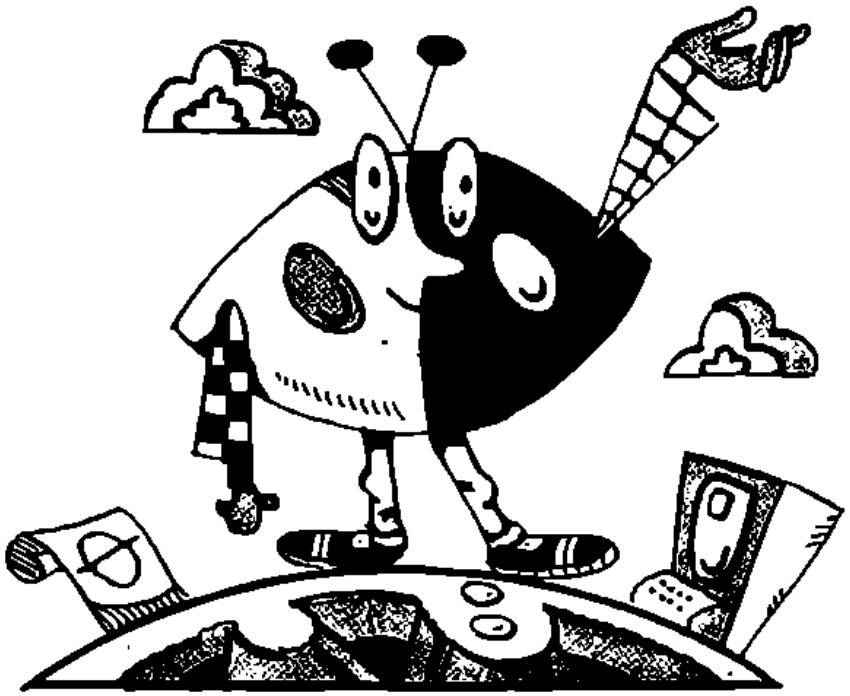
Кристине

потому, что... всё хорошо!

С. ВАТТ
М. МАНГАДА

БЕЙСИК ДЛЯ ДЕТЕЙ

Перевод с испанского



КИЕВ
«РАДЯНСЬКА ШКОЛА»
1990

ББК 32.97
В21

Перевод с издания:
Watt S., Mangada M. Basic para niños.— Madrid: Paraninfo, S. a Watt S., Mangada M. Basic avanzado para niños.— Madrid: Paraninfo, S. a.

Переводчики *А. Г. Олейник, В. П. Симоненко*
Художник *В. Н. Игнатов*

Ватт С., Мангада М. Бейсик для детей: Пер. с исп.— К.: Рад. шк., 1990.— 222 с.

В книге в игровой и наглядной форме, доступной младшим школьникам, излагаются основы программирования и объясняется азбука одного из наиболее распространенных и удобных в использовании языков программирования — языка бейсик. Текст снабжен необходимыми комментариями для взрослых — родителей и учителей, которые даже не имея специальных знаний в области программирования, смогут проконсультировать юных читателей.

Для учащихся младшего и среднего школьного возраста.

Редактор *Г. В. Криволапова*

В $\frac{4802020000-246}{M210(04)-90}$ 381—90

ISBN 5-330-00135-8

ПРЕДИСЛОВИЕ

«Бейсик для детей» дает первоначальные знания об использовании компьютера и сообщает элементарные принципы программирования на языке бейсик.

Предлагаемый материал облегчает раскрытие секретов компьютера и в соответствии с творческими возможностями и опытом детей позволяет им продвигаться вперед в изучении бейсика.

Структура книги такова, что и ребенок, и взрослый вместе открывают мир компьютера, при этом родители или учитель выступают гидом, помощником, но не преподавателем. Основной функцией взрослого является не предоставление ребенку информации, а стимулирование обучения. И именно эта книга дает ребенку соответствующий материал, который позволит ему найти ответы на интересующие его вопросы.

Родители и учитель найдут в этой книге толкования и комментарии, методические указания, дидактический материал по обучению ребенка языку бейсик.

Книга «Бейсик для детей» выполнена структурно в такой форме, которая облегчает ее использование родителями и учителями, не имеющими никаких предварительных знаний по этому языку.

НЕКОТОРЫЕ ПРЕДВАРИТЕЛЬНЫЕ ЗАМЕЧАНИЯ

Б каждом пункте книги «Бейсик для детей» освещается новое понятие. Пункты расположены по степени нарастанння трудности понятий. Кроме того, операторы (команды) языка и материал, уже рассмотренный ранее, все время повторяются для того, чтобы дети прочно усваивали информацию и понятия.

Постепенное усложнение материала способствует индивидуализации изучения, гарантирует такое распределение усилий и энергии между процессами усвоения материала и практической деятельностью, что дети 8—14 лет вполне в состояиии освоить язык бейсик, останавливаясь на любом уровне изучения и оставаясь на нем до тех пор, пока не будут готовы перейти на новый, более высокий уровень.

Материал в книге изложен так, чтобы облегчить повторение и закрепление основных понятий языка бейсик. Даже рискуя повториться, авторы все время совершают «круг назад» — к рассмотренной ранее информации, с тем, чтобы по прочтении книги был охвачен весь процесс изучения: получение, усвоение, закрепление и запоминание информации.

Специально в книге не рассматриваются некоторые более сложные конструкции бейсика, которые не соответствуют целям этой книги.

Книга состоит из двух частей. В конце каждой части книги предлагается несколько программ простых игр с использованием уже знакомых команд бейсика.

Чтобы успешио достичь наших целей следует иметь в виду ряд важнейших требований.

- Взрослый должен сначала сам прочитать текст (или вместе с ребенком). Целесообразно также до начала работы с ребенком прочитать *замечания для взрослых*, имеющиеся в каждом пункте.

- Ребенок практикуется только тогда, когда он этого пожелает. В любом случае практические занятия не должны быть слишком продолжительными, чтобы не вызывать физической и умственной усталости ребенка.

- Нужно проводить такое количество занятий, какое необходимо для закрепления каждого понятия. Не следует переходить к изучению нового понятия, если вы не уверены в том, что ребенок глубоко понял предыдущий материал.

- Позволяйте ребенку идти дальше, придумывать новые практические упражнення, обнаруживать новые возможности использования компьютера. Пусть ребенок сам все откроет.

НЕКОТОРЫЕ ПРЕДВАРИТЕЛЬНЫЕ ЗАМЕЧАНИЯ

- Пробуждайте любознательность ребенка, заинтересуйте его, добивайтесь, чтобы он экспериментировал, если необходимо, с помощью кратких инструкций, таких как: «Что произойдет, если...», «Попытайся...». Откройте дорогу инициативе ребенка. Достигнутые им результаты будут служить ему стимулом в обучении.

- Не изучайте в один день более одной команды бейснка. Перед тем как начать работу, повторите основные понятия, изученные на предыдущих уроках.

- Целесообразно с помощью компьютера выполнить все программы, которые даны в книге, даже если они кажутся очень легкими.

НАШ БЕЙСИК

Для того чтобы работать с нашими программами, необходимо использовать инструкции, имеющиеся при вашем персональном компьютере. В книге рассматривается обобщенный, стандартный язык бейсик.

Но может быть, что в некоторых персональных компьютерах используются другие команды, другие комбинации клавиш для выполнения одной и той же операции. В случае возникновения сомнений посмотрите руководство по программированию вашего персонального компьютера. Отличия будут минимальными.

Например:

- В некоторых компьютерах требуется в конце каждой программы наличие команды END, а в других в этом нет необходимости.

- В некоторых моделях машин не используется команда LET. Следовательно, нет необходимости использовать этот оператор, когда присваиваете значение переменной. Так:

  пишите 10 A=A+1
  вместо 10 LET A=A+1

- Функция RND может представляться в различных формах: часто встречается форма RND(1), RND(0) или RND(X). Кроме того, используется просто RND.

- Некоторые компьютеры требуют включения в начало программы

НЕКОТОРЫЕ ПРЕДВАРИТЕЛЬНЫЕ ЗАМЕЧАНИЯ

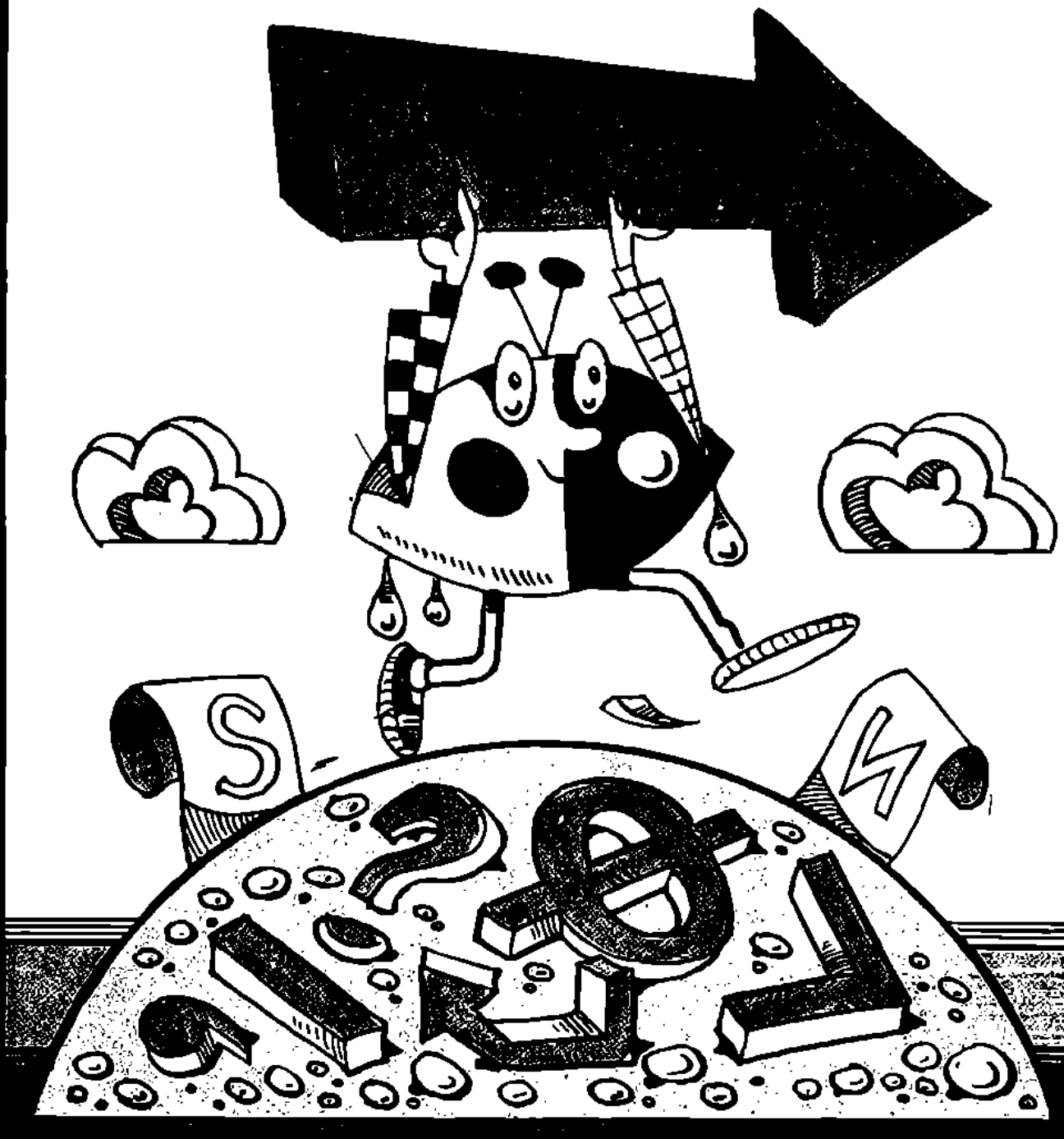
функции RANDOMIZE для получения нескольких серий случайных чисел всякий раз, когда исполняется программа.

- Для того чтобы прервать программу во время ее исполнения, можно использовать различные команды, в зависимости от типа компьютера. Например, BREAK или RUN STOP.

- Повторяем, что эти различия минимальны и ознакомление с руководством по программированию вашего компьютера развеет любое сомнение, которое может у вас появиться.

Часть I

БЕЙСИК ДЛЯ ДЕТЕЙ



Здравствуй! Меня зовут Артуро. Любой компьютер может сделать много интересных вещей. Он может поиграть с тобой, загадать загадки, помочь в изучении математики, умеет много другого. Единственно, что ты должен сделать – это научиться разговаривать с ним. Эта книга научит тебя этому. Ты увидишь как это легко. Если тебе будет что-нибудь непонятно, тебе помогут твои родители, преподаватели и я. НАЧНЕМ?

Как уже сказал Артуро, персональный компьютер может тебе помочь во многих делах, и ты можешь провести с ним время очень интересно и полезно. Компьютер быстро понимает и так же очень быстро выполняет задания (намного быстрее тебя), однако, чтобы сделать их, ему нужно, чтобы ты сначала объяснил ему, что он должен делать. И это, ты должен объяснить компьютеру на языке, который ему понятен.

В самом деле, если ты заранее не объяснишь компьютеру, что нужно сделать, то он ничего и не исполнит. Эта книга научит тебя средству общения, *языку*, который твой компьютер понимает и с помощью которого ты сможешь общаться с ним. Этот язык называется *бейсик*.

Представь себе, что Артуро не умеет играть в прятки. Ты объясняешь ему правила игры примерно так: «Сначала я считаю до десяти, ты прячешься. Затем я тебя ищу. Если я найду тебя, я выиграл. Если не найду, то выиграл ты, а я проиграл».

С компьютером происходит то же, что и с Артуро. Ты сначала должен объяснить компьютеру, что ты хочешь от него и что он должен сделать. При этом ты должен учесть, что компьютер очень пунктуален, аккуратен, точен и что, когда он уже работает, ему нельзя объяснять. Мы должны сразу говорить ему всё очень точно.

Кроме этого, ты должен всегда учитывать, что компьютер выполняет действия в том порядке, какой ты ему указал. Например, если ты хочешь объяснить ему, как играть в прятки, ты должен сказать:

1. Я считаю до десяти
2. Ты прячешься
3. Я тебя ищу
4. Если я тебя найду, то я выиграл
5. Если я тебя не найду, то выиграл ты

Эти объяснения компьютер будет *хранить в своей памяти,* то есть будет помнить сначала первую команду, затем вторую, третью и так до последней.

Запоминает компьютер очень быстро, и, когда ты закончишь объяснять, как играть, он уже будет уметь играть. Совокупность приказов, *команд,* которую ты ему дал, называется *программой.* Те пять команд, которые мы написали, и есть программа для игры в прятки.

Как ты уже мог увидеть, эта программа имеет нумерованные предложения-строки, которые компьютер читает, чтобы узнать, как играть. Эти строки должны быть пронумерованы для того, чтобы компьютер знал, что ему делать сначала, что потом и чем закончить. Поэтому всегда нужно нумеровать предложения-строки от меньшего номера к большему.

Теперь напиши на бумаге программу, которую ты уже изучил, и внеси в нее изменения, которые мы сейчас будем делать.

Представь себе, что ты хочешь изменить игру, введя новое правило: «Нельзя прятаться внутри дома». Тогда ты должен написать новую команду в программе:

6. Если ты спрячешься в доме, то игра прекращается. Тогда программа будет выглядеть так:

1. Я считаю до десяти
2. Ты прячешься
3. Я тебя ищу
4. Если я тебя найду, то я выиграл
5. Если я тебя не найду, то выиграл ты
6. Если ты спрячешься в доме, то игра прекращается

Как видишь, компьютеру нужно сообщить все.

Так как мы должны быть очень точными, то лучше было бы команду 6 поставить между командами 2 и 3, что означало бы: игра прекратится как только Артуро начнет играть нечестно и спрячется в доме.

В этом случае мы сэкономим много времени, которое затратили бы на поиск спрятавшегося в доме Артуро. Обрати внимание на важность этого и на разницу между тем, стоит эта команда в конце или между командами 2 и 3.

Но дело в том, что у нас нет места, чтобы поставить новую команду между строками 2 и 3. Что будем делать?

Этот прием тебе будет полезен всегда. Это очень просто. Нужно нумеровать строки не одну за другой, а десятками — от 10 с шагом 10, и тогда ты всегда будешь иметь место, чтобы поставить или вставить, где нас устраивает, другое предложение-команду, которое нам необходимо.

Для компьютера это не имеет значения, так как единственное, что ему нужно, чтобы строки были пронумерованы от меньшего к большему, с тем, чтобы он знал, в каком порядке нужно читать и исполнять наши команды.

В таком случае, предыдущая программа будет выглядеть так:

```
10  Я считаю до десяти
20  Ты прячешься
30  Я тебя ищу
40  Если я тебя найду, то я выиграл
50  Если я тебя не найду, то выиграл ты
```

А новая команда программы может иметь любой номер между 20 и 30 (например, номер 25):

```
25  Если ты спрятался в доме, то игра прекращается
```

Ты не должен беспокоиться, что написал эту команду и сообщил компьютеру после всех других. Компьютер поставит эту команду автоматически на то место, которое соответствует ее номеру.

Когда ты посчитаешь, что ты не имеешь больше команд и что программа уже готова, ты дашь компьютеру специальный приказ (команду) и он сделает то, чему ты его предварительно научил. Это называется *выполнить* программу или *прогнать* программу.

Немного позже мы рассмотрим, как это все сделать. Скоро мы сыграем в прятки с компьютером.

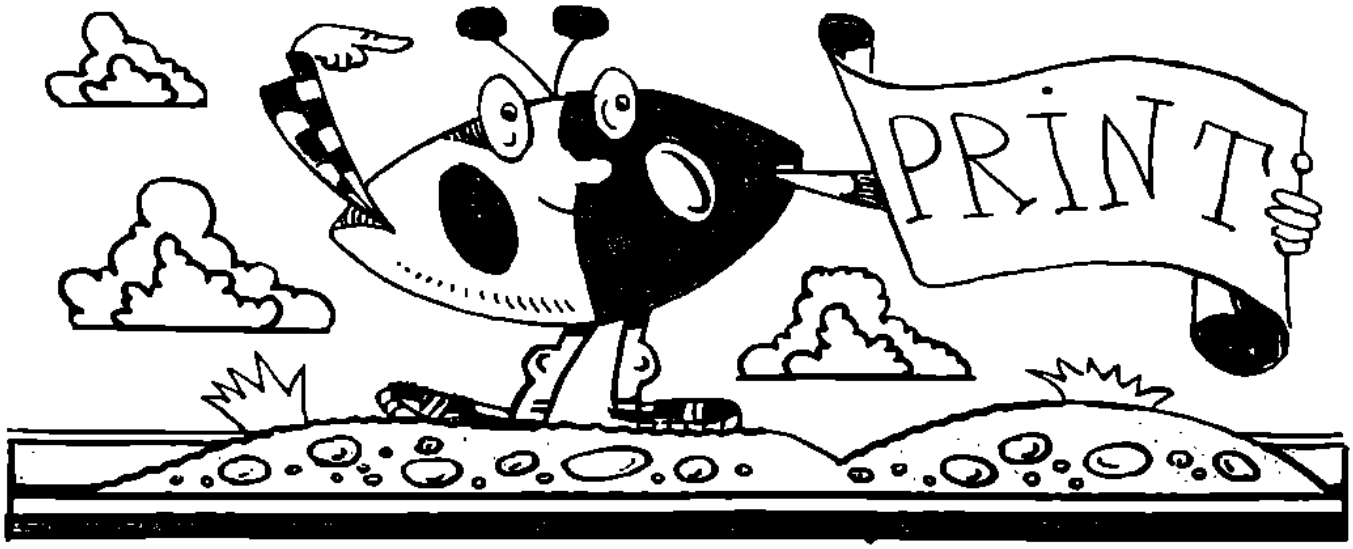
ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Используя диалоговый язык изложения и легко усваиваемые обычные слова и понятия, ребенок постепенно привыкает к специфичной для микроинформатики термииологии.

- Повторяйте с ребеиком осиовиые понятия на примерах, используя сиачала обычный язык, а затем термииы, выделеиные в тексте.

PRINT

Как мы уже раньше говорили, компьютер понимает и использует только определенные точные приказы и команды. Давай познакомимся с первой командой языка бейсик. Это команда PRINT.



В основном PRINT приказывает компьютеру отпечатать все, что ты ему укажешь и что находится между знаками " " (кавычки). Компьютер выведет это на экран.

■ Попробуем написать имя нашего друга Артуро. Используя клавиши своего компьютера, напиши:

```
10 PRINT "АРТУРО"
```

Для того чтобы ввести эту строку в память компьютера, ты должен нажать клавишу, специально для этого предназначенную. Это может быть клавиша с обозначением ENTER, RETURN, ВК или другим, например ↵. Узнай, какая именно.

Обрати на это внимание. Это очень важно. Каждый раз, когда ты нажимаешь на эту клавишу,— это все равно, что сказать компьютеру: «Запоминай!» Не забудь, что всегда, когда ты напишешь с помощью клавиатуры любую команду и захочешь, чтобы компьютер ее запомнил, нужно нажать эту клавишу. Это очень важно, иначе он забудет эту строку.

А сейчас что будем делать?

Когда ты решишь, что программа полностью закончена и не надо ничего добавлять, введи команду RUN.

Это будет вторая команда, с которой мы ознакомимся. Она используется для того, чтобы компьютер начал выполнять программу, которую он имеет в своей памяти.

Ты уже знаешь, что запуск программы мы называем также исполнением, прогоном программы. Поэтому команду RUN ты должен использовать всегда, когда закончишь программу и захочешь посмотреть, как она работает.

В нашем случае, после того как мы закончили ввод программы (даже если она состоит из одной строки), исполним ее. Нажми RUN. И ты сразу увидишь, что на экране появится имя нашего друга.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Спросите у ребенка, что произойдет, если он забудет ввести (написать) одну или две кавычки в комаиде PRINT. Пусть посмотрит, что выведет компьютер в этом случае. (Могут быть разные варианты в зависимости от модели компьютера.)

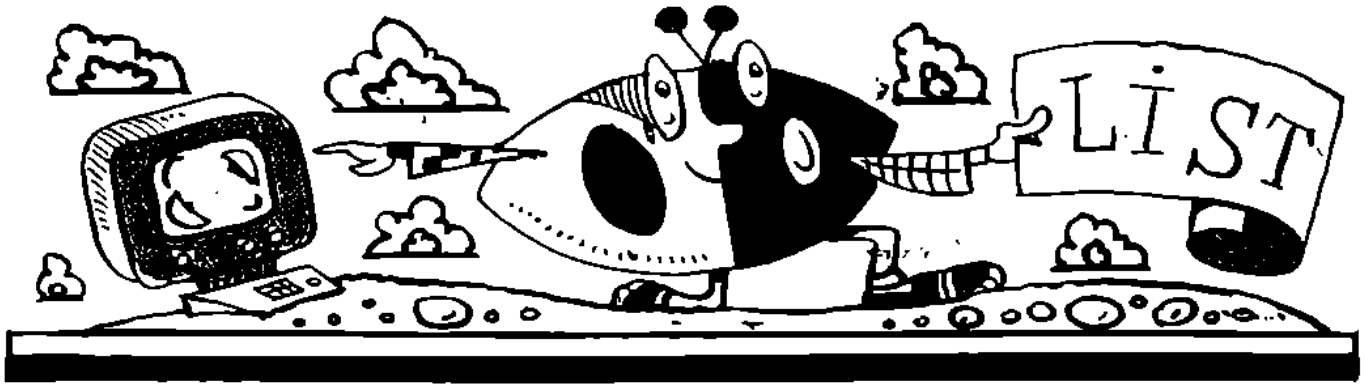
- Не забудьте напомнить ребенку, что он должен нажимать клавишу ENTER или RETURN (ВК) после ввода каждого оператора, команды.

- Ребенок должеи выполнить иа компьютере все программы, представлеиные в книге, даже если оии кажутся очень простыми.

ПОПРОБУЕМ ДРУГИЕ ВЕЩИ

■ Давайте расширим программу так, чтобы написать (вывести на печать) имена друзей Артуро. Но для этого нам нужно увидеть нашу первоначальную программу.

Для того чтобы сделать это, набери слово LIST.



Команда LIST нужна для того, чтобы иметь возможность посмотреть на экране дисплея все строки программы, которую хранит компьютер в своей памяти. Задавая команду LIST, мы будто спрашиваем у компьютера: Что ты уже знаешь? И тогда компьютер *выведет*, или напишет, на экране строки программы, которые мы уже имеем в его памяти.

После выполнения этой команды на экране появится:

```
10  PRINT "АРТУРО"
```

Вспомни, что это наша начальная программа. Увеличь ее, добавляя другие строки с именами друзей Артуро.

Введи:

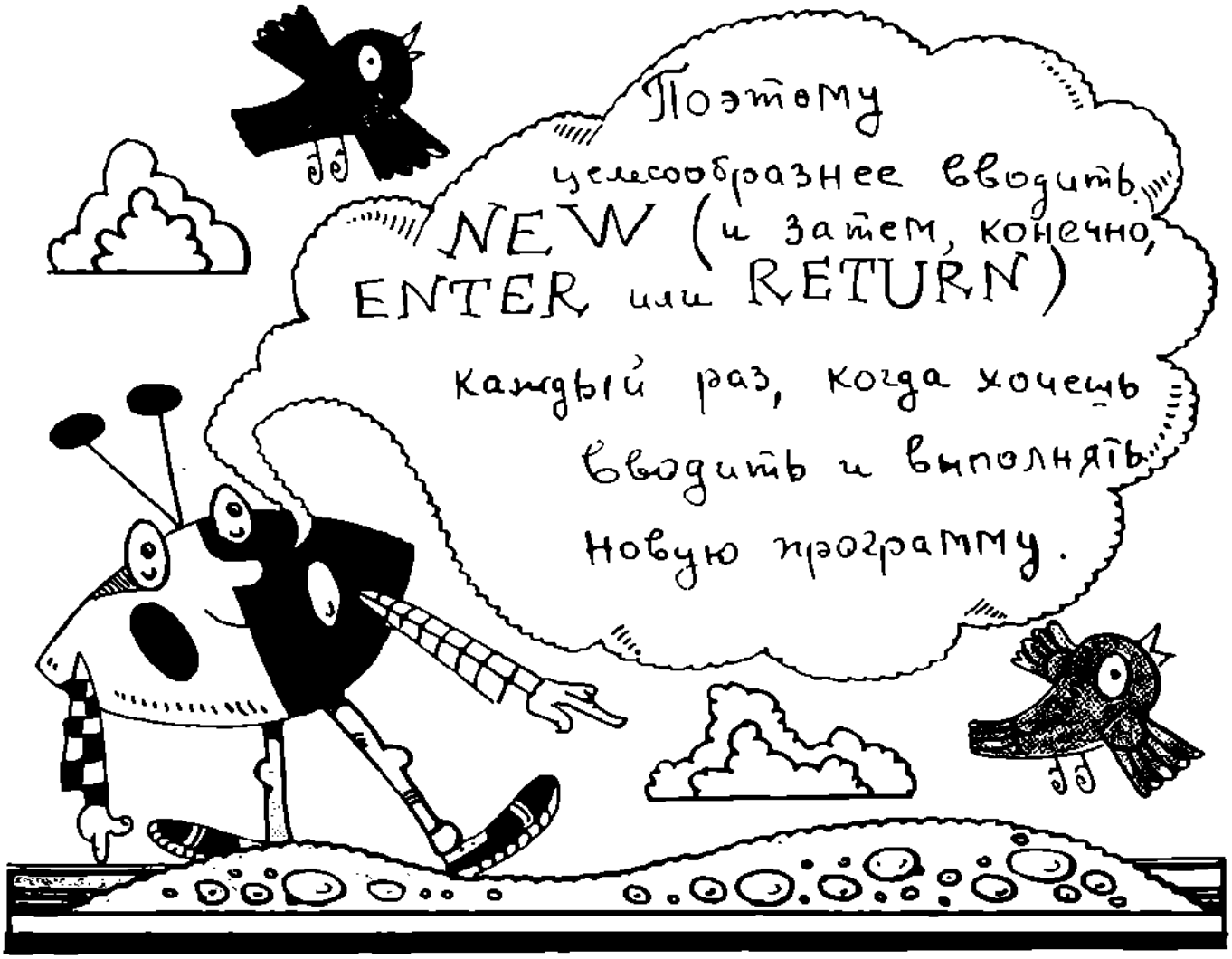
```
20  PRINT "КАРЛОС"
30  PRINT "ХУАНИТО"
40  PRINT "ХОРХЕ"
```

Не забудь, что после того как ты с помощью клавиатуры наберешь каждую строку программы, ты должен занести ее в память компьютера (нажимая клавишу ENTER или RETURN). Когда закончишь ввод, нажми RUN, и ты увидишь на экране имена всех друзей Артуро.

NEW

Вот ты уже и составил свою первую программу! Теперь пойдем дальше и попробуем создать программу более сложную и бóльшую по объему. Так как мы будем создавать совершенно новую программу, то сначала сотрем из памяти компьютера все, что мы занесли туда раньше. Для этого можно использовать несколько способов. Дальше мы изучим эти способы. Один из них (очень простой) — это выключение и включение компьютера. При этом компьютер вернется в свое первоначальное состояние, и ты можешь опять разъяснять ему, что делать.

Другой способ (более сложный) — это набрать команду NEW. При этом ты тоже стираешь из памяти компьютера все, что было туда помещено ранее и сохранено. И тогда он не будет помнить ничего из того, что он знал и делал раньше.



Составим новую программу с именами друзей Артуро и их любимыми игрушками. Введи:

```
10  PRINT "АРТУРО     ВЕЛОСИПЕД"
20  PRINT "КАРЛОС     МЯЧ"
30  PRINT "ХУАНИТО    КНИГА"
40  PRINT "ХОРХЕ      ПОЕЗД"
```

Выполни эту программу (нажми RUN) и ты увидишь на экране дисплея:

```
АРТУРО     ВЕЛОСИПЕД
КАРЛОС     МЯЧ
ХУАНИТО    КНИГА
ХОРХЕ      ПОЕЗД
```

■ А сейчас обрати внимание на одну очень важную вещь. Ты будешь иметь то же самое на экране, если изменишь программу, и она будет иметь следующий вид:

```
10 PRINT "АРТУРО", "ВЕЛОСИПЕД"
20 PRINT "КАРЛОС", "МЯЧ"
30 PRINT "ХУАНИТО", "КНИГА"
40 PRINT "ХОРХЕ", "ПОЕЗД"
```

При исполнении этой программы (с RUN) увидишь, что получил на экране то же самое, что и раньше.

Это произошло потому, что запятую (,) компьютер воспринимает не так, как мы. То, что ты напишешь в команде PRINT после запятой, компьютер напишет, отделяя от предыдущего пробелами.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Работайте, используя команды LIST и RUN, пока не достигиете иеобходимого усвоения и не приобретёте практических навыков.

- В одной команде PRINT все тексты в кавычках, отделёниые друг от друга с помощью запятых, будут печататься компьютером в разных зонах экраиа. Экраи может быть разделен иа две и больше зои, в эависнмости от типа компьютера. Будьте внимательны, так как если текст, разделенный запятыми, довольно длинный, то он может заинмать следующие зоны и строки экрана.

- Покажите ребенку наиболее простые способы для стирания программы. Используйте NEW.

- Необходимо выбирать продолжительность изучеиия каждой новой команды языка бейснк в соответствии с желанием и способиостью ребенка к восприятию во время обучения.

ПОПРОБУЕМ ЕЩЕ

■ Карлосу подарили ракетку для игры в теннис, и сейчас она ему больше нравится, чем мяч. Как нам теперь исправить нашу программу?

Обрати внимание на то, что сейчас ты должен не создавать новую программу, а только изменить ту, что имеешь. Мы должны изменить в программе только строку, которая соответствует Карлосу. Если ты точно не помнишь номер строки, выведи программу на экран и затем с тем же номером напиши еще раз строку:

```
20 PRINT "КАРЛОС", "РАКЕТКА"
```

Введи эту строку в программу (нажми ENTER или RETURN), выполни ее (нажми RUN) и посмотри, что получилось. Ты на экране будешь иметь:

■ Теперь Хуанито уехал на каникулы, и его друзья не могут играть с ним. Как исключить Хуанито из твоего списка?

Напиши только номер строки, которая соответствует Хуанито (30) и введи ее непосредственно в память. Это все равно, что ты сообщишь компьютеру, что в этой строке сейчас ничего нет. При выполнении всей программы ты увидишь, что Хуанито уже нет в списке.

■ А ты хочешь быть членом команды Артуро? Как включить твое имя в список?

Ты должен включить новую строку в программу таким же образом, как и раньше.

■ Попробуем сделать что-нибудь другое.

Набери

```
15 PRINT
```

Введи в память и выполни. Что произошло? Теперь попробуй, что произойдет, если добавить к программе следующую строку:

```
25 PRINT
```

Выполни программу и ты на экране будешь иметь:

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Помните, что ребенок должен просматривать программу каждый раз при любом ее изменеиии, чтобы убедиться, что все сделано правильно.

- Для того чтобы добавить новые строки программы, используйте номера большего значения, чем номер последией строки программы.

- Для того чтобы при выводе на экран появилась пустая строка, достаточно добавить новую строку с иомером и командой PRINT.

- Необходимо обращать внимание ребенка на то, что он не должен снова вводить уже использованные номера строк программы, иначе они сотрутся.

- Помогите привыкнуть ребенку к особенностям и характеристикам клавиатуры его компьютера, стирая какие-либо строки, буквы, написанные неправильно, используя клавиши пробела, удаления и т. д.

ЕЩЕ НЕМНОГО О КОМАНДЕ PRINT

С помощью компьютера ты можешь делать рисунки. Один из способов — использование команды PRINT.

Попробуй нарисовать прямоугольник с помощью следующей программы (используй клавишу пробела, когда необходимо):

```
10   PRINT"XXXXXXXX"
20   PRINT"X      X"
30   PRINT"X      X"
40   PRINT"X      X"
50   PRINT"X      X"
60   PRINT"XXXXXXXX"
```

Выполни эту программу и посмотри, что получится.

ПРОДОЛЖАЙ ПРОБОВАТЬ

Как сделать прямоугольник меньше?

Сотри, например, строки 40 и 50. Если ты забыл, как это сделать, вспомни, что ты делал, когда исключал Хуанито из компании Артура. Выполни программу.

Если ты все сделал правильно, то на экране появится такой прямоугольник:

```
XXXXXXXX
X      X
X      X
XXXXXXXX
```

Ты можешь сделать более сложный рисунок, используя другие символы. Попробуй сделать что-то похожее на это:

```
10  PRINT "000$000"
20  PRINT "00$$$00"
30  PRINT "0$$$$$0"
40  PRINT "$$$$$$$"
50  PRINT "0$$$$$0"
60  PRINT "00$$$00"
70  PRINT "000$000"
```

Попытайся сделать более сложные рисунки. Попробуй нарисовать лодку с парусом. Чтобы парус получился красивым, вместо того чтобы окружать рисунок какими-либо символами (как мы это сделали с ромбом, окружив его нулями), подсчитай количество необходимых пробелов слева от фигуры и оставь пустые места. Для этого используй клавишу пробела. Введи следующую программу:

```
 10  PRINT"          X"
 20  PRINT"         XX"
 30  PRINT"        XXX"
 40  PRINT"       XXXX"
 50  PRINT"      XXXXX"
 60  PRINT"     XXXXXX"
 70  PRINT"    XXXXXXX"
 80  PRINT"   XXXXXXXX"
 90  PRINT"  XXXXXXXXX"
100  PRINT" XXXXXXXXXX"
110  PRINT"XXXXXXXXXXX"
120  PRINT"XXXXXXXXXXXX"
130  PRINT"          X"
140  PRINT"          X"
150  PRINT" XXXXXXXXXXXXXXXXXXXXXXX"
160  PRINT"  XXXXXXXXXXXXXXXXXXXXX"
170  PRINT"   XXXXXXXXXXXXXXXXXXX"
180  PRINT"     XXXXXXXXXXXXXXXXX"
190  END
```

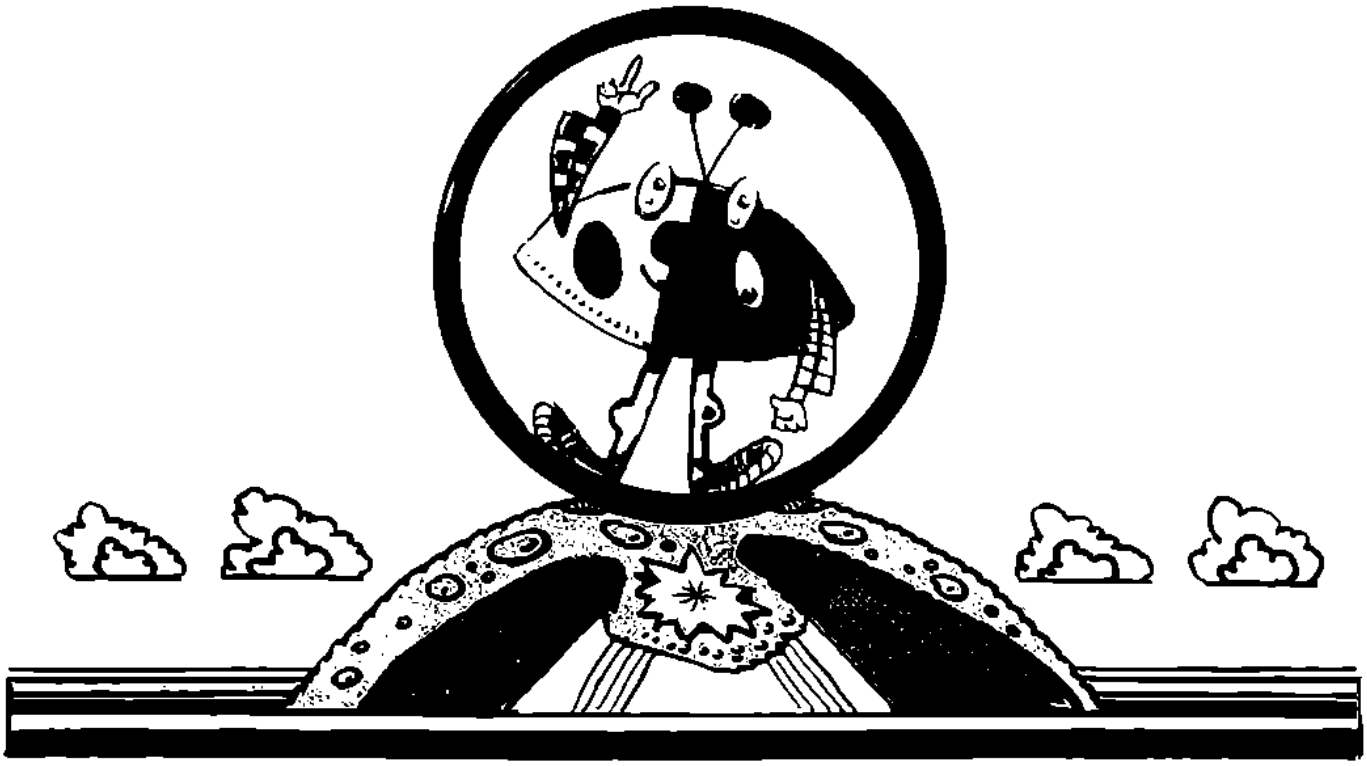
ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Для того чтобы стереть строку, достаточно ввести ее иомер и нажать клавиши ENTER или RETURN.

- Предложите ребеику новые рисунки: треугольники, окружности и др., при этом побуждайте его создавать собствеиные произведения.

- Не забывайте об очистке памяти компьютера, о стирании программы (используя NEW или другне соответствующие клавиши) перед иачалом создаиня иового рисунка. Эта комаида стирает всю информацию, хранящуюся в памяти компьютера. Команду NEW следует прнменять только тогда, когда мы будем начинать новую программу.

- Желательно, чтобы дети, используя миллиметровую бумагу (или бумагу в клеточку), предварительиио выполняли свой рисунок иа бумаге.



LET

Теперь ознакомимся с широко используемой командой LET. Для того чтобы понять этот оператор лучше, приведем пример. Посмотри хорошо на эту маленькую программу:

```
10  LET A=14
20  PRINT A
```

■ Выполняя программу, ты увидишь, что на экране появится 14. Что случилось? Вторая строка программы нам уже хорошо известна. Ты приказал компьютеру отпечатать А.

Но вместо того чтобы написать А, компьютер вывел число 14. Это случилось потому, что в первой строке ты указал компьютеру, что А равняется 14. Это сделано с помощью команды LET. LET A=14 — это то же самое, что сказать: пусть A=14.

Посмотри: здесь ты уже не пишешь PRINT и потом букву А внутри кавычек (" "). Это потому, что ты не хочешь, чтобы компьютер отпечатал букву А, а тебя интересует только значение, которое ты присвоил (дал) этой букве (в нашем случае 14).

■ Попробуем еще раз. Запиши программу:

```
10  LET A=14+1
20  PRINT A
```

Выполни программу, и на экране появится число 15. Как видишь, команда LET позволяет тебе дать букве A любое значение, какое ты захочешь. Компьютер сохранит в своей памяти то значение A, которое ты ей присвоил. Вместо буквы A ты можешь использовать любую другую букву, или сочетание нескольких букв (как AB, HZ и т. д.), или комбинацию букв и цифр, всегда начинающуюся с буквы (A2, P3 и т. д.).

Всем им ты можешь присвоить любое значение, какое захочешь. Они называются *переменными,* потому что могут изменять свое значение.

■ Ты можешь изменять их значения, когда захочешь.

Например:

```
10  LET X=6
20  LET X=8
30  PRINT X
```

При выполнении этой программы компьютер выведет на экране число 8, так как во второй строке программы с номером 20 мы присвоили переменной X значение 8, которое аннулировало предыдущее значение переменной X, равное 6.

■ Посмотри, как можно использовать оператор LET для выполнения вычислений.

Введи следующую программу:

```
10  LET M=4
20  LET X=6
30  LET S=M+X
40  PRINT S
```

Когда выполним эту программу, компьютер выведет число 10. В двух первых строках мы присвоили M значение 4, а X значение 6.

В строке 30 присвоили переменной S значение суммы M + X (которое равно 10). А в строке 40 приказали вывести это значение.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Команда LET предназначена для присвоения переменной, стоящей слева от знака =, значения или выражения справа от этого знака. Все эти переменные есть числовые переменные.

- В некоторых компьютерах можно не использовать слово LET для присвоения переменным их значений. Достаточно записать, например, 10 X=6 (или 10 X:=6) вместо 10 LET X=6

● Покажите ребеику, что комаида PRINT A (или любой другой переменной) предназиачена для вывода на экран значеиия, присвоенного этой переменной. Объясните ребенку различие в написании PRINT A и PRINT "A".

ПРОДОЛЖАЕМ ПРОБОВАТЬ

■ Следующая, очень простая программа нам скажет, сколько всего книг у Артуро. Он имеет три книги «Капитан Труэно», шесть «Пятеро» и две «Питуфос». Для того чтобы выразить это проще, обозначим количество книг «Капитан Труэно» переменной T, «Пятеро» — переменной C и «Питуфос» — переменной P. Тогда программа будет такой:

```
10  LET T=3
20  LET C=6
30  LET P=2
40  LET S=T+C+P
50  PRINT S
```

Переменная S будет равна сумме различных типов книг, которые имеет Артуро (посмотри строку 40 программы).

Выполни программу и проанализируй результат.

■ Если Артуро подарят новую книгу «Питуфос», то что нужно изменить в программе? Очевидно, только строку 30:

```
30  LET P=3
```

Как видишь, с помощью команды LET ты можешь присвоить различные значения каждой переменной.

■ Мама Артуро сказала ему, чтобы он пошел в магазин и купил один пакетик карамелек стоимостью 10 песет (название монеты) и два пирожных по 25 песет. Сколько денег нужно Артуро?

Программа для вычисления будет следующей:

```
10  LET C=10
20  LET P=25
30  LET X=P*2
40  LET S=C+X
50  PRINT "АРТУРО НУЖНО";
60  PRINT S;
70  PRINT "ПЕСЕТ"
```

Посмотрим, с какими переменными в программе мы работаем и какие значения имеет каждая переменная.

C: эта переменная обозначает стоимость конфет, и, так как мы знаем, что пакетик конфет стоит 10 песет, мы имеем C=10. Поэтому запишем LET C=10.

P: эта переменная обозначает стоимость пирожного. Как известно, одно пирожное стоит 25 песет, то есть P=25. Поэтому запишем LET P=25.

X: эта переменная обозначает стоимость двух пирожных. Как известно, одно пирожное стоит P песет, а два пирожных стоят два раза по P песет. Поэтому запишем LET X=P*2.

S: переменная, обозначающая количество денег, необходимых Артуро для покупки конфет и пирожных, то есть сумму стоимости пакетика конфет и двух пирожных. Поэтому запишем LET S=C+X.

Выполни программу и посмотри результат на экране.

■ Очень важное замечание! Посмотри, что произошло, когда мы использовали знак ; (точка с запятой) в команде PRINT.

Что произошло с выполнением следующих команд PRINT? И что произойдет, если не написать знак ; (точка с запятой)?

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Разъясните ребенку назначение строки 30 в программе.
- Точка с запятой (;) в команде PRINT дает возможность записать следующую команду PRINT как продолжение первой. Рассмотрите это с ребенком.
- Там, где это необходимо, оставляйте пробелы рядом с кавычками, чтобы текст в последующих PRINT, идущих сразу через точку с запятой, не был записан слитно с предыдущим. Пояснение этого будет дано для детей позже.
- Предложите ребенку, чтобы он выполнил другие вычисления, связанные с жизнью. Используйте задачи на сложение, вычитание, умножение, деление.
- В бейсике умножение представляется символом *, деление — символом /.

ПРОДОЛЖАЕМ ПРОБОВАТЬ

До сих пор, как мы видели, оператор LET использовался для присвоения переменной любого цифрового значения. А теперь посмотрим, как можно присвоить значению переменной слово или текст.

Для этого достаточно написать после переменной символ $ и не забыть записать слово или текст внутри кавычек (" ")

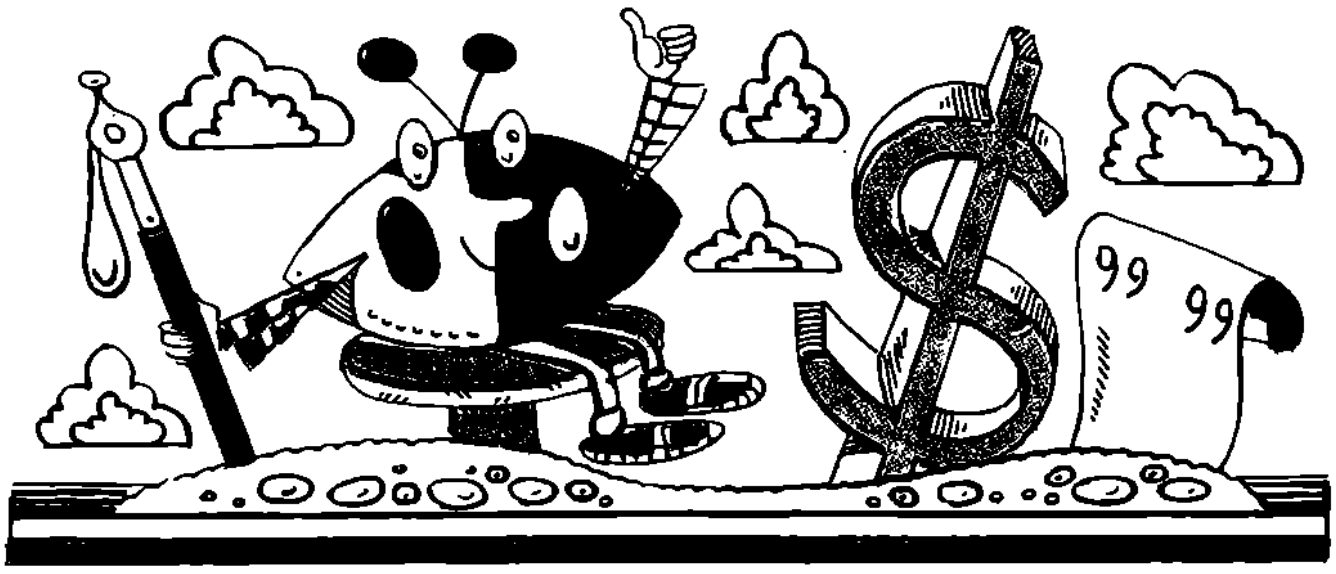
Введи программу:

```
10  LET  A$="АРТУРО"
20  LET  B$="ХОРХЕ"
30  LET  C$="КАРЛОС"
```

Теперь компьютер имеет в памяти имена, которые мы дали каждой переменной. Попробуй выполнить эту программу, добавив к ней следующие строки:

```
40  PRINT  C$;
50  PRINT  " ДРУГ ";
60  PRINT  A$
```

Запусти программу. Как видишь, компьютер имеет очень хорошую память. Ты уже знаешь, как выполняет компьютер вывод, когда в командах PRINT используется точка с запятой (;).



Ты можешь получить тот же результат, если сотрешь строки 40 и 60, а строку 50 напишешь так:


```
50  PRINT A$;" ДРУГ "; C$
```

Выполни программу. Для того чтобы получить красивый вывод, обрати внимание на строку 50 (в любой из двух программ). Ты должен оставить пробел между первой кавычкой и словом ДРУГ и словом ДРУГ и второй кавычкой. Если ты не сделаешь этого, компьютер все выведет слитно.

■ Как видишь, это очень интересно сделать так, чтобы группа слов или текст были значением какой-то переменной.

Кроме того, ты можешь работать с этими переменными точно так же, как и с другими. Вспомни предыдущие пункты, где ты мог оперировать с переменными: ты мог суммировать, делить, умножать...

Только что рассмотренные переменные мы можем суммировать. Нет смысла в выражении: "Артуро" умножен на "Кар-

лоса". Однако можем говорить: "Артуро" и "Карлос". Как иллюстрацию, рассмотрим программу:

```
10  LET A$="АРТУРО"
20  LET B$="ЕСТЬ"
30  LET C$="НЕ ЕСТЬ"
40  LET D$="ВЫСОКИЙ"
50  LET E$="УМНЫЙ"
60  LET F$="ПЛОХОЙ"
70  LET G$="ХОРОШИЙ"
80  PRINT A$+B$+D$
```

Выполни программу. Потом измени строку 80, используя другие переменные. Например, PRINT A$+C$+F$ или также PRINT A$+B$+G$.

Как видишь, можно суммировать эти переменные и получать группы соответствующих текстов.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Эти переменные иазываются символьными перемеииыми, и им могут быть присвоены строки цифровых, алфавитных или алфавитно-цифровых символов. Единственное, что можно с ними делать,— это суммировать. Нельзя суммировать цифровую переменную с символьной.

- В тексте использовались выражеиия «текст» или «группа слов», когда речь шла об операциях с символьными строками. Используйте сначала эти понятия, постепеино меняя их на более точные термины, с предпочтнтельным использоваиием термиина «символьная строка».

- Обратите внимание ребеика на значение запятых, пробелов в текстах, используемых в команде PRINT.

INPUT

Это одна из самых красивых команд. С ее помощью ты можешь «говорить» с компьютером, который тебя спрашивает и отвечает.

INPUT тебе позволит вводить данные с клавиатуры компьютера во время исполнения программы. Посмотри на следующий пример:

```
10  LET A=2
15  PRINT "ОЖИДАЮ ЧИСЛО"
20  INPUT M
30  LET B=A*M
40  PRINT B
```

■ Запусти программу. Компьютер начнет ее выполнять. Прочтет первую строку, присвоит значение 2 переменной А. Наконец доходит до команды INPUT и останавливается, ожидая какое-нибудь число, которое ты должен ввести с помощью клавиатуры.

Это число есть значение, которое ты даешь переменной М, то есть компьютер сообщает: жду ввода числа и значение этого числа присвою переменной М. Это и есть то, что делает команда INPUT.

Дай М любое значение, например 5. Для этого нажми клавишу 5 (затем ENTER или RETURN). Компьютер продолжит выполнять программу и, так как все переменные имеют свои значения, выполнит операцию, указанную в строке 30, и выведет результат.

В нашем случае возьмет А со значением 2 и М, которому мы дали значение 5, умножит их и выведет результат — 10. ■ Попробуй дать различные значения М, а затем измени значение А.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

● Команда INPUT позволяет обеспечить связь между компьютером и пользователем, вводить даниые во время выполиения программы. Опе-

рируя с командами PRINT и INPUT и комбинируя их, можем получить диалоговое взаимодействие между ребенком и компьютером, что мы и увидим в дальнейшем.

ПРОДОЛЖАЕМ ПРОБОВАТЬ

■ Давай сделаем программу, с помощью которой ты сможешь разговаривать со своим компьютером.

```
 10  PRINT "НАПИШИ СВОЕ ИМЯ"
 20  INPUT N$
 30  PRINT "В КАКОМ ГОДУ ТЫ РОДИЛСЯ?"
 40  INPUT A
 50  PRINT "В КАКОМ ГОДУ ЖИВЕМ?"
 60  INPUT B
 70  LET X=B-A
 80  PRINT
 90  PRINT
100  PRINT
110  PRINT
120  PRINT
130  PRINT "ХОРОШО ";N$;" ТЫ ИМЕЕШЬ ";X;" ЛЕТ"
```

Запусти программу и отвечай на вопросы. Компьютер может говорить с тобой!

Теперь немного проанализируем эту программу. В строках программы 10, 30 и 50 ты объясняешь компьютеру, о чем ему нужно спрашивать тебя. В строке 20 ты видишь, что мы поставили символ $ после переменной. Это сделано потому, что значением этой переменной будет не число, а слово. Если ты этого хорошо не поймешь, то вернись и прочитай еще раз страницу 32.

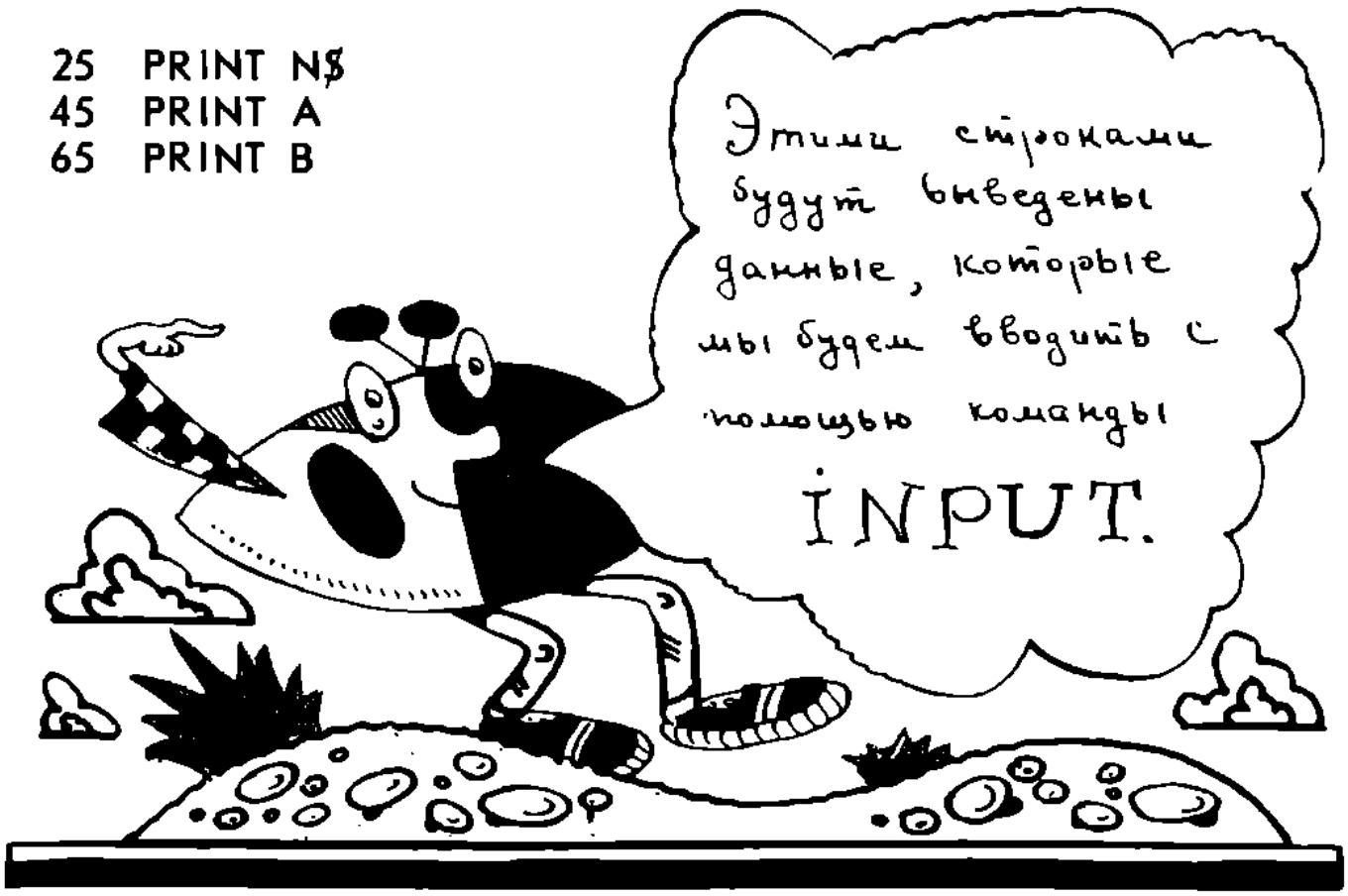
В строках 40 и 60 символ $ не используем, так как переменные A и B будут иметь числовые значения. В программе можешь увидеть, что A — это год рождения, а B — год, в котором живешь.

С помощью строк с 80 по 120 мы пропустили строки (сделали их пустыми), для того чтобы отделить вопросы от ответов.

INPUT

Выполним очень интересное изменение программы. Сделаем так, чтобы наши ответы тоже были выведены на экран. Добавим строки:

```
25  PRINT N$
45  PRINT A
65  PRINT B
```



Выполни новую программу и посмотри на различия.

■ Артуро готовится к каникулам и хочет знать, сколько дней он пробудет дома. Так как он знает, сколько дней он будет на пляже, он написал следующую программу:

```
10  PRINT "СКОЛЬКО ДНЕЙ КАНИКУЛ?"
20  INPUT T
30  PRINT "СКОЛЬКО ДНЕЙ БУДУ НА ПЛЯЖЕ?"
40  INPUT P
50  LET D=T-P
60  PRINT "МНЕ ОСТАНЕТСЯ ";D;" ДНЕЙ КАНИКУЛ ДОМА"
```

Запусти программу. Компьютер будет тебя спрашивать. Отвечай вместо Артуро. Компьютер даст тебе окончательный ответ.

Так же, как ты сделал в предыдущей программе, модифицируй эту таким образом, чтобы твои ответы выводились на экран.

■ Артуро очень нравится быть на пляже, и он хочет знать, сколько часов он может наслаждаться песком и морем. Для того чтобы сделать этот расчет, используем следующую программу. Ты, конечно, знаешь, что сутки имеют 24 часа.

```
10  PRINT "СКОЛЬКО ДНЕЙ БУДЕТ АРТУРО НА ПЛЯЖЕ?"
20  INPUT P
30  LET H=24
40  LET S=P*H
50  PRINT "АРТУРО БУДЕТ НАХОДИТЬСЯ ";S;" ЧАСОВ
    НА ПЛЯЖЕ"
```

Выполни программу и ты получишь ответ. Скорректируй программу так, чтобы твой ответ компьютеру был на экране.

Однако Артуро не учел, что он не может находиться на пляже целые сутки. На отдых отводится только несколько часов в день. С помощью следующего расширения программы помоги ему посчитать, сколько часов Артуро будет на пляже.

INPUT

Ответь на вопросы, которые тебе задаст компьютер, когда программа полностью будет дополнена.

Сначала добавим к предыдущей программе:


```
 55  PRINT
 56  PRINT
 57  PRINT
 58  PRINT
 60  PRINT "СКОЛЬКО ЧАСОВ ОТДЫХАЕТ ДОМА АРТУРО
     КАЖДЫЙ ДЕНЬ?"
 70  INPUT X
 75  PRINT X
 70  LET Z=24-X
 80  LET Q=Z*P
100  PRINT  АРТУРО БУДЕТ НА ПЛЯЖЕ ";Q;" ЧАСОВ В
     ТЕЧЕНИЕ СВОИХ КАНИКУЛ"
```

Если, кроме этого, ты хочешь помочь Артуро узнать, сколько часов он будет играть каждый день, измени строку 100 следующим образом:

```
100  PRINT "АРТУРО БУДЕТ НА ПЛЯЖЕ ";Q;" ЧАСОВ В
     ТЕЧЕНИЕ СВОИХ КАНИКУЛ, ТО ЕСТЬ ";Z;" ЧАСОВ
     КАЖДЫЙ ДЕНЬ"
```

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

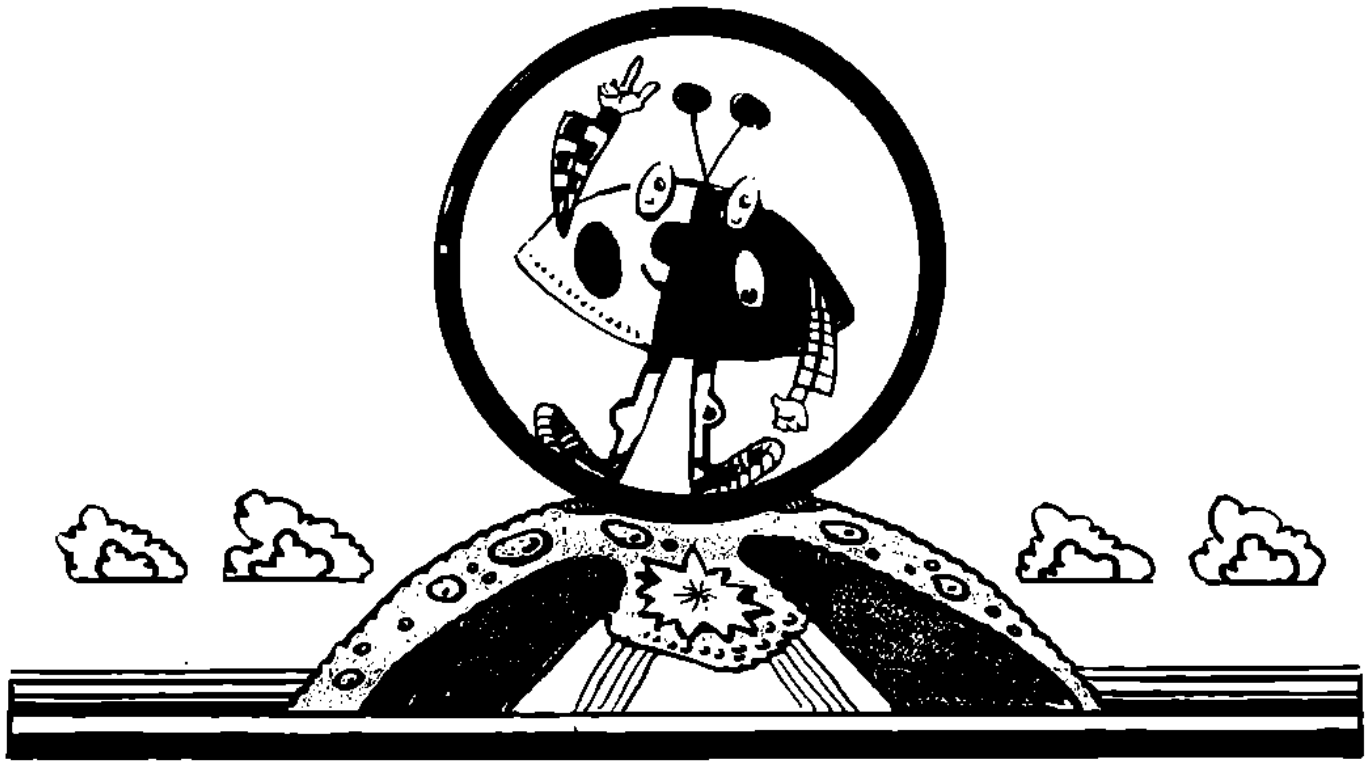
- Поработайте внимательно с этой программой. Советуйте ребенку использовать команду LIST каждый раз при любой необходимости. Будьте внимательны к использованию пробелов во фразах команды PRINT, для того чтобы эти фразы были правильно выведены.

- Помните о важности точки с запятой в строках с PRINT. Обратите внимание ребенка на различия при использовании запятой в качестве приказа вывода в следующей зоне экрана и просто виутри текста в команде PRINT, как это сделано в строке 100. Покажите на нескольких примерах использование запятой вне и виутри кавычек текста в команде PRINT.

- Повторите несколько раз программу, анализируя с ребенком значения, присвоенные различным переменным с помощью оператора присваивания LET.

- Сделайте так, чтобы ребенок каждый раз видел строки, где были сделаны исправления и добавления к программе.

- На этой основе помогите ребенку прндумать новые варианты задач и вместе сделайте соответствующие программы. Побуждайте ребенка к этой работе.



GOTO

Сделаем наши программы более полными, используя команду GOTO.

С помощью этой команды ты можешь дать точное указание компьютеру. Оно означает ПЕРЕЙТИ НА, и компьютер должен перейти к той строке программы, номер которой ты укажешь.

Эта команда изменяет нормальный ход выполнения программы.

■ Например, в программе:

Компьютер прочитает первую строку и выведет на экран «ПРИВЕТ». Потом прочитает вторую, которая укажет — идти на строку 10. Тогда он еще раз выведет ПРИВЕТ. Затем прочтет вторую строку, которая прикажет снова перейти на десятую строку. И еще раз выведет ПРИВЕТ и так далее. Эта программа никогда не закончится! Запусти ее и ты увидишь, что получится.

На твоем экране появился столбец слов ПРИВЕТ. И хотя на экране уже нет места, чтобы писать, компьютер будет продолжать выполнять программу. Для того чтобы остановить его, необходимо ввести BREAK (или STOP), эта команда прекратит выполнение программы (вполне вероятно, что твой компьютер использует другую команду вместо BREAK, выясни какую).

Измени строку 10 программы:

```
10  PRINT "ПРИВЕТ",
```

и ты еще раз увидишь важную роль запятой.

Производя изменения, не забудь перед этим нажать BREAK.

Измени строку 10 еще раз, поставив точку с запятой (;) на место запятой. Посмотри, как это повлияло на вид вывода.

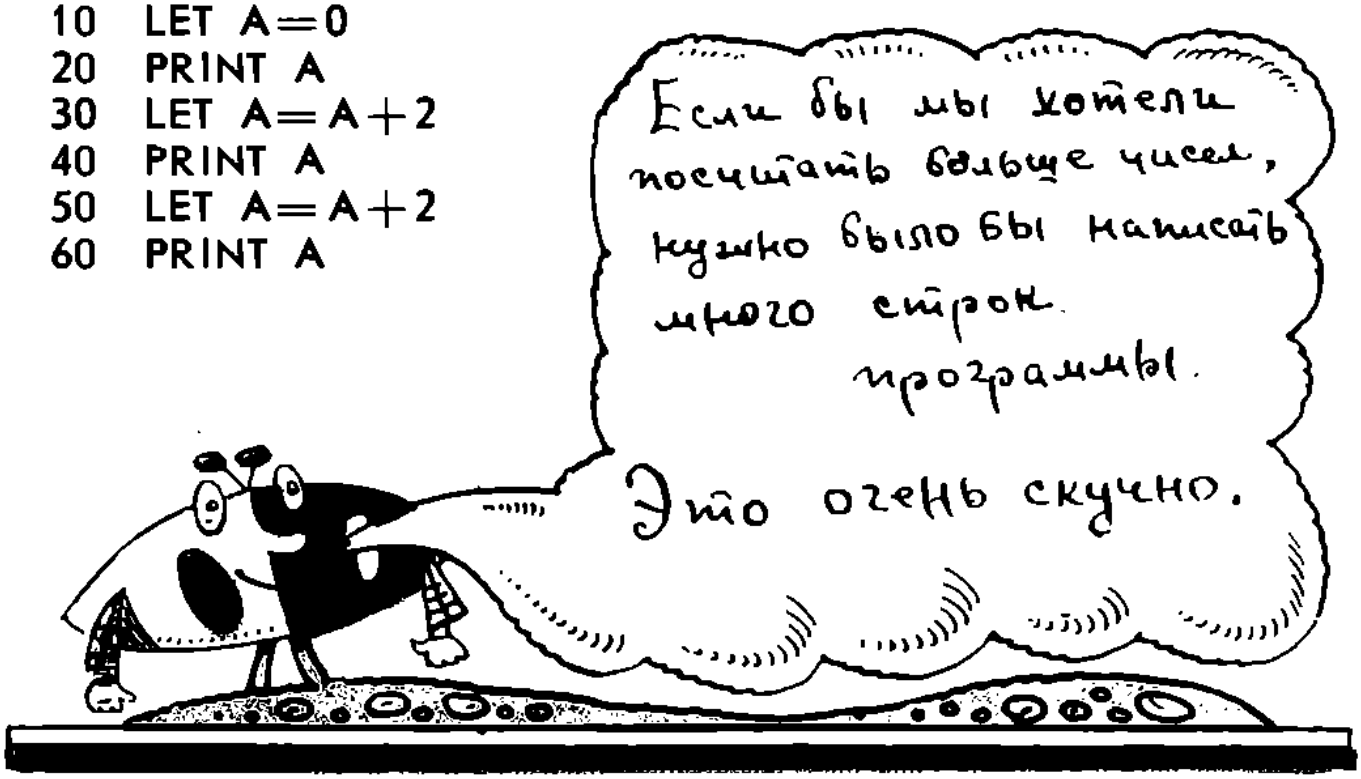
Не забывай, что не одно и то же поставить запятую внутри кавычек (" ") или вне их. Попробуй написать строку 10 так:

```
10  PRINT "ПРИВЕТ,"
```

Вернемся к команде GOTO и рассмотрим несколько программ, где она используется.

■ Например, Артуро хочет посчитать числа через 2. Мы могли бы написать следующую программу:

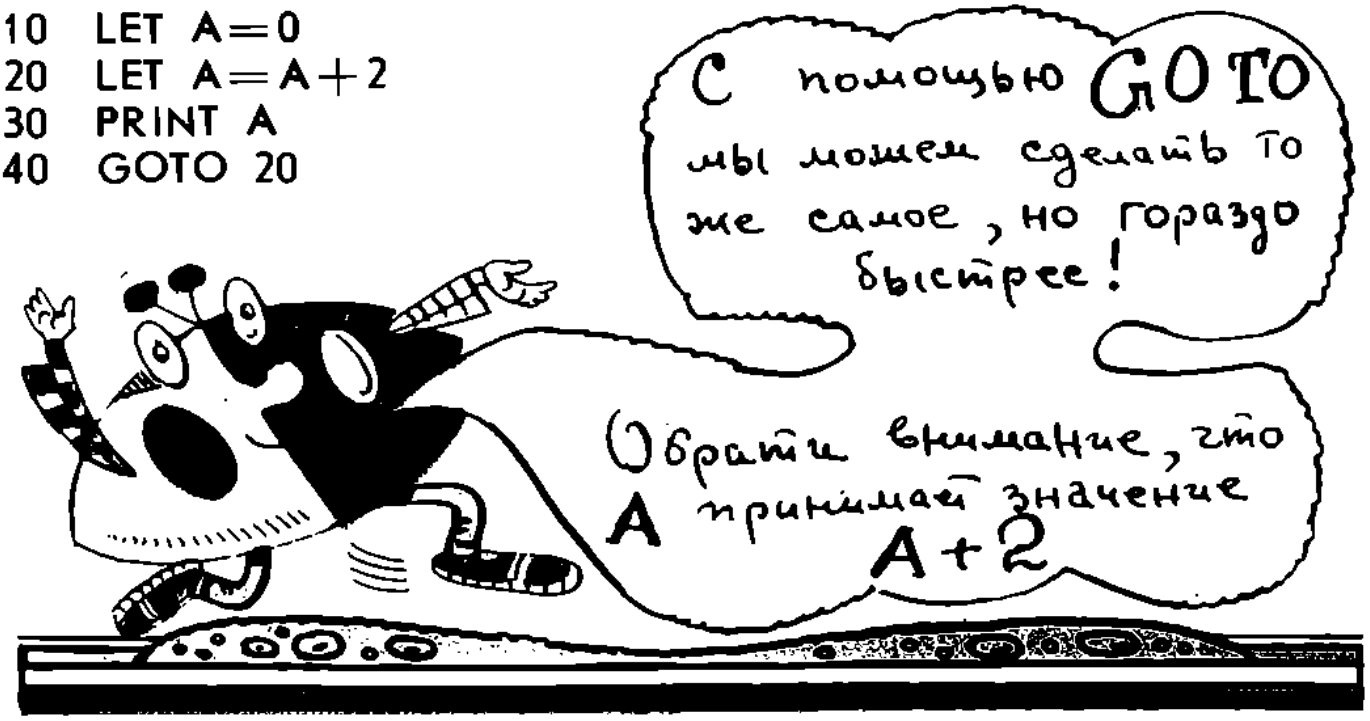
```
10  LET A=0
20  PRINT A
30  LET A=A+2
40  PRINT A
50  LET A=A+2
60  PRINT A
```



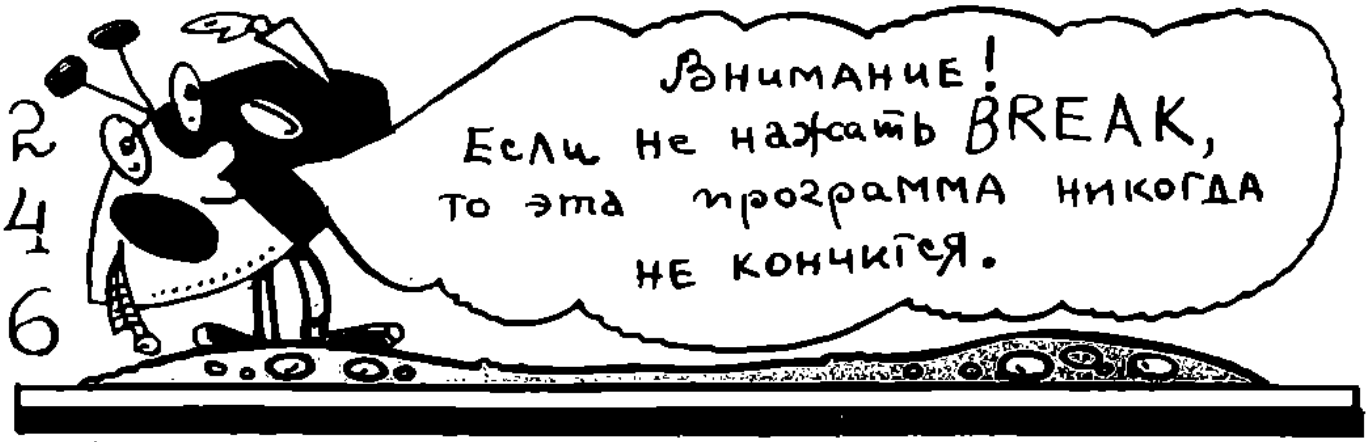
Эта программа считает только до 4. Если хотим, чтобы счет продолжался и дальше, программа была бы очень длинной...

Давай используем GOTO. Напишем следующую программу:

```
10  LET A=0
20  LET A=A+2
30  PRINT A
40  GOTO 20
```



Если запустить программу, то на экране появится то же самое, что мы видели раньше:



Посмотри внимательно программу. В строке 10 ты присваиваешь значение переменной A. В данном случае значение 0.

В строке 20 ты даешь A новое значение: первоначальное значение увеличивается на ту величину, какую ты хочешь (в нашем случае это 2), то есть A теперь имеет значение 2.

В строке 30 выводим значение, которое имеет A в этот момент (то есть 2).

В строке 40 приказываем возвратиться на строку 20, и компьютер берет последнее значение A, которое в настоящий момент равно 2, и добавляет две единицы. При этом A принимает значение 4.

В дальнейшем выводится на экран это значение, и опять команда GOTO в строке 40 посылает компьютер на строку 20, где он берет последнее значение A (которое сейчас равно 4) и суммирует его с 2. Печатает новое значение (равное 6) и т. д. Лучше проводить этот анализ с программой перед глазами.

■ Сделай свою собственную программу, чтобы она считала суммы по 5. Для этого измени строку 20. Если не знаешь, как это сделать, то посмотри, что говорит Артуро.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Команда GOTO обозначает скачок или безусловный переход. Он меняет последовательный порядок выполнения программы по строкам. Эти переходы или скачки могут быть на строки, имеющие меньшие или большие номера, чем иомер строки, в которой находится команда GOTO.

- Попытайтесь показать и объяснить ребенку с помощью программ в этой главе, что каждый раз, когда компьютер исполняет строку с командой GOTO и возвращается на строку, стоящую раньше, он в памяти имеет последние текущие значеиия переменных.

- Обратите внимание, что в математике выражение $A = A + 2$ не имеет смысла. В информатике же оно верно, так как сначала вычисляется значение, которое находится после знака =, а потом это значение присваивается переменной, стоящей слева от этого знака.

ПРОДОЛЖАЕМ ПРОБОВАТЬ

■ Артуро изучает суммирование и хочет сделать таблицу сложения. Поможем ему составить программу, которая сделает это, например такую, которая могла бы суммировать любое натуральное число и 3. Введем:

```
10  LET X=0
20  LET S=X+3
30  PRINT X;"+3=";S
40  LET X=X+1
50  GOTO 20
```

Запустим эту программу, и ты увидишь на экране таблицу:

```
0+3=3
1+3=4
2+3=5
3+3=6
4+3=7
```

■ Теперь давай сделаем таблицу, которая поможет Артуро умножать на 3. Изменим строки 20 и 30 программы. Попробуй это сделать сам. Сейчас необходимо умножать, а не суммировать. Если не получится, не огорчайся. Мы только начинаем программировать.

Смотри, как это сделать:

```
20  LET S=3*X
30  PRINT "3*";X;"=";S
```

Измени эти строки в программе, запусти ее и на экране получишь таблицу:

```
3*0=0
3*1=3
3*2=6
3*3=9
3*4=12
3*5=15
```

GOTO

■ Команда GOTO может быть применена не только для перехода на строки программы, которые уже выполнились, но может тоже использоваться для перехода к строкам программы, имеющим номера намного больше, чем имеет строка с GOTO, то есть со строки 40 можем перейти непосредственно, например, на строку 100.

Это мы более подробно рассмотрим позже. А сейчас ты можешь потренироваться немного со «странной» программой, приведенной ниже.

Обычно в одной программе ты не сделаешь более бессмысленной последовательности строк, как в этой программе, но она поможет тебе потренироваться и посмотреть, с какой скоростью работает компьютер.

Введи:

```
10  GOTO 40
20  PRINT "ТВОЙ ДРУГ ";
30  GOTO 80
40  PRINT "СКАЖИ КТО ";
50  GOTO 20
60  PRINT "КТО ТЫ "
70  END
80  PRINT "И ТОГДА СКАЖУ ";
90  GOTO 60
```

Обрати внимание на строку 70. Она содержит команду END, которая служит для того, чтобы компьютер остановился, когда дойдет до нее. В противном случае компьютер будет бесконечно повторять серию слов (попробуй стереть строку 70 и выполни программу).

Команда END имеет важное значение. Некоторые компьютеры используют STOP вместо END, но их функции одинаковы.

Да! В этом типе программ ты должен обратить особое внимание на пробелы, которые ты ставишь в тексте между кавычками, для того чтобы текст хорошо читался.

IF

Рассматриваемая нами команда GOTO будет более полезной, когда мы ознакомимся с командой IF. Эта команда позволит твоему компьютеру немного думать самому и принимать свои решения. Естественно, он всегда делает это в соответствии с командами и информацией, которые ты ему задал.

■ Посмотрим, как действует эта команда в программе.

```
10  PRINT "СКОЛЬКО БУДЕТ ДВА ПЛЮС ДВА?"
20  INPUT A
30  IF A=4 THEN 60
40  PRINT "ТЫ ОШИБСЯ"
50  GOTO 10
60  PRINT "ТЫ УГАДАЛ!"
70  END
```

Эта программа работает следующим образом. В строке 10 начинается разговор с тобой. В строке 20 компьютер ожидает число, которое ты должен ввести с помощью клавиатуры.

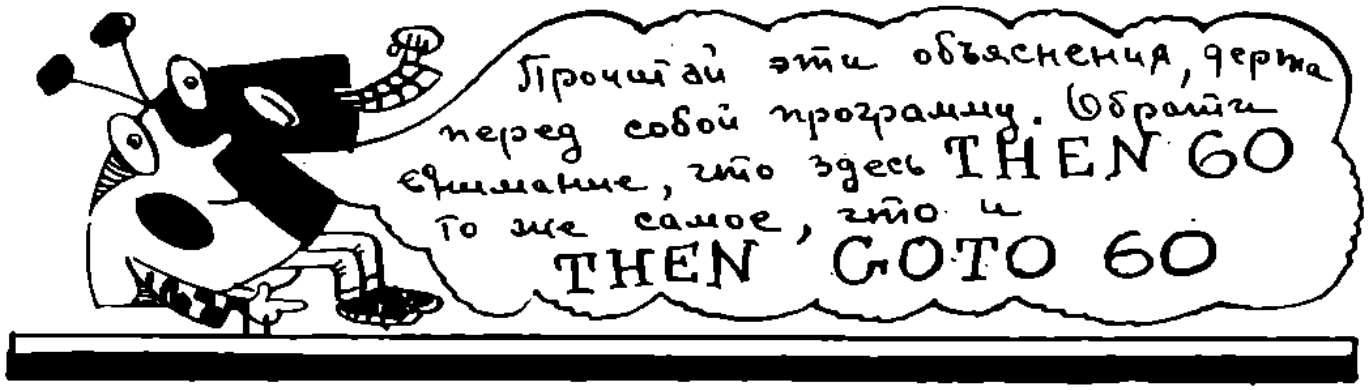
Теперь будет выполняться самая интересная строка. В строке 30 компьютер встретится с условием: если значение, которое ты дал A равно 4, тогда выполняется обязательный переход на строку 60, без исполнения строк 40 и 50. И ты получишь сообщение, что ответил правильно.

Но, когда A не равно 4, программа продолжает выполняться нормально. Если проанализировать строку с командой условного перехода, то станет ясно, что компьютер может принять только одно решение по условию.

Как видишь, если А не равно 4, тогда выполняется строка 40 и тебе сообщается, что ты ошибся, а следующая строка 50 передаст управление строке 10 для повторного вопроса.

Короче говоря, с помощью команды IF мы сообщаем компьютеру некоторое условие. Если это условие соблюдается, компьютер выполнит некоторое действие (это действие нужно заранее определить). Если условие не соблюдается, то выполняется другое действие (также определенное тобой).

В предыдущем примере мы указали такое условие: А равно (=) 4. Условия, которые будем давать компьютеру, могут быть и другими, такими, как *меньше чем* (которое обозначается символом <), *больше чем* (обозначается символом >), *отличное от* (представляется символом <>) и другими.



■ Предыдущую программу можно написать и следующим образом:

```
10  PRINT "СКОЛЬКО БУДЕТ ДВА ПЛЮС ДВА?"
20  INPUT A
30  IF A<> 4 THEN 60
40  PRINT "ТЫ УГАДАЛ!"
50  GOTO 80
60  PRINT "ТЫ ОШИБСЯ"
70  GOTO 10
80  END
```

Обрати внимание, как необходимо организовать все строки программы при изменении условия. Но результат будет тот же самый.

Условие, которое проверяется, было: если А *не равно* 4, то иди на строку 60.

Проанализируй хорошо, как работает каждая из этих программ.

■ Так же ты можешь использовать строки знаков (то есть тексты).

Давай напишем программу почти такую же, как первая, но с вопросом об имени Артуро:

```
10  PRINT "КАК ЗОВУТ НАШЕГО ДРУГА?"
20  INPUT A$
25  PRINT A$
30  IF A$="АРТУРО"THEN 60
40  PRINT  "ТЫ ОШИБСЯ"
50  GOTO 10
60  PRINT "ТЫ УГАДАЛ!"
70  END
```

Ты должен быть очень внимателен, отвечая компьютеру. Если ты не ответишь ему словом, которое он знает, то он не поймет твоего ответа. Попробуем сделать один опыт с компьютером. Запусти еще раз программу и ответь «АТУРО» вместо «АРТУРО». Компьютер не знает слово «АТУРО» и поймет его как другое имя. Он сообщит тебе, что ты ошибся и, так как A$ не будет точно равно слову «АРТУРО», компьютер не перейдет на строку 60 и будет выполнять оператор строки 40.

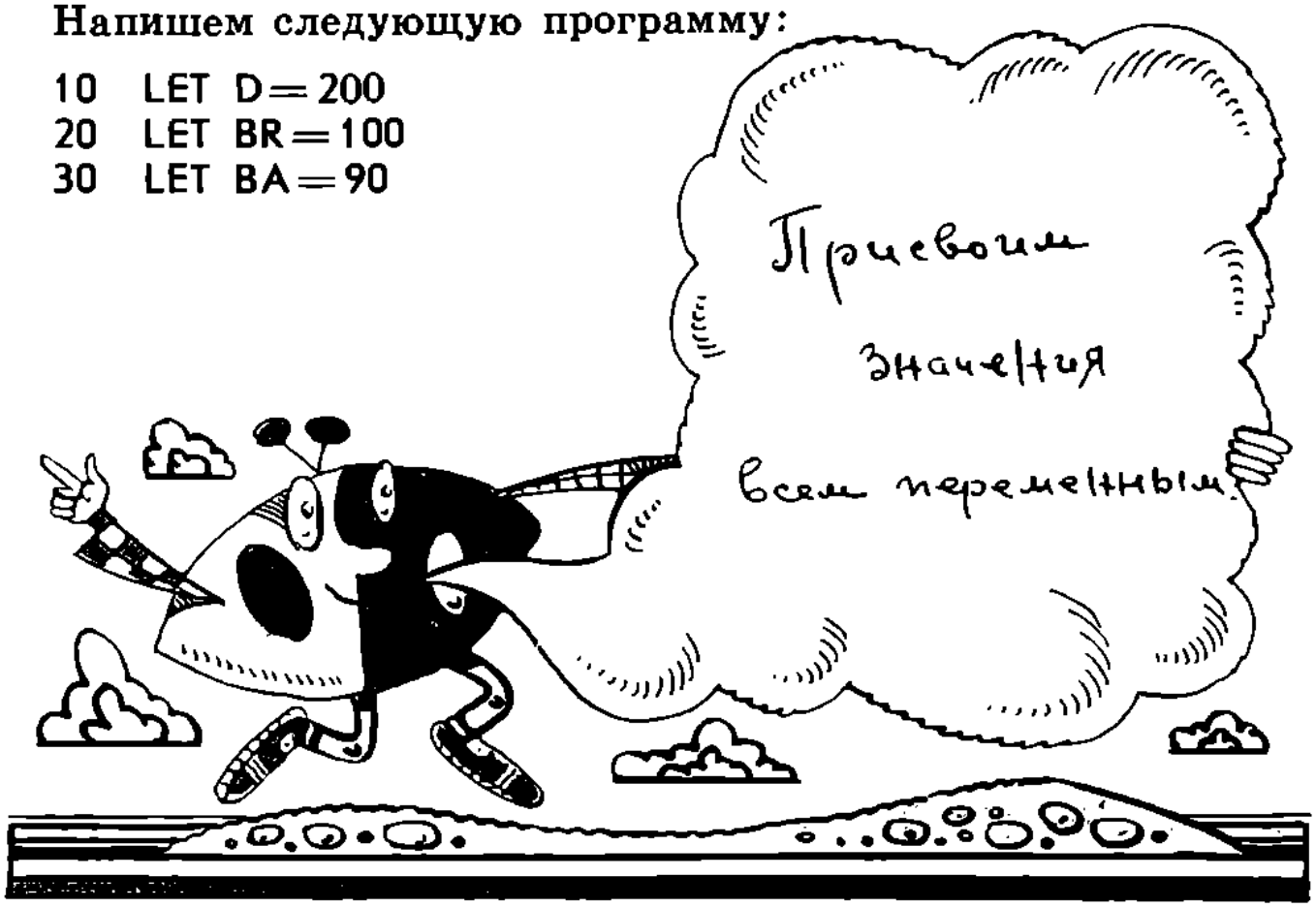
■ Рассмотрим другой пример, для того чтобы тебе было более понятно, как использовать эту команду.

Программа тебе может показаться немного длинной, но она очень простая, так как мы знаем все команды, которые в ней использованы. С помощью этой программы мы решим одну задачу нашего друга Артуро. Он хочет купить несколько мячей в магазине. У него есть только 200 песет.

В магазине имеется два типа мячей — красные и голубые. Красный мяч стоит 100 песет, а голубые — по 90 песет. Артуро должен решить: купить ему один мяч красный, или один голубой, или оба мяча. Он также хочет знать, сколько денег у него останется в каждом случае.

Напишем следующую программу:

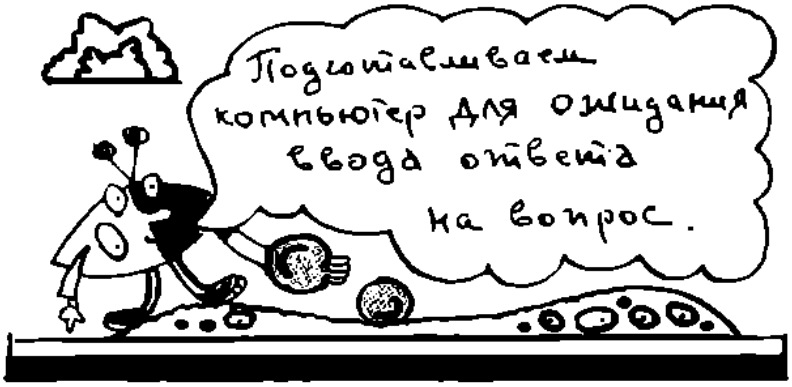
```
10  LET D=200
20  LET BR=100
30  LET BA=90
```



В этих строках программы обозначим через D, сколько денег имеет Артуро, BR и BA — цены красного и голубого мяча соответственно.

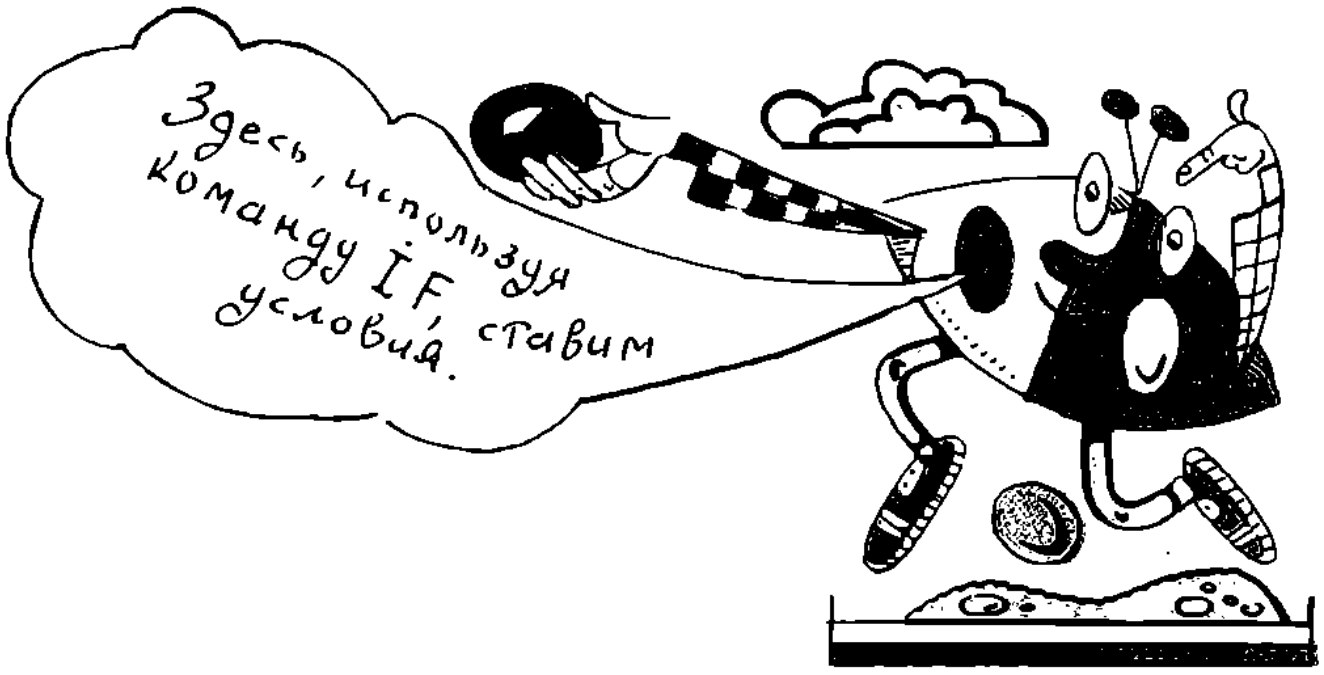
Продолжим:

```
40  PRINT "СКАЖИ, КАКОЙ МЯЧ КУПИШЬ: КРАСНЫЙ,
    ГОЛУБОЙ ИЛИ ОБА"
50  INPUT A$
55  PRINT A$
```



С помощью этих строк программы обучаем компьютер спрашивать нас и указываем, что он должен ждать нашего ответа и вывести его на экран.

Продолжим дальше программу:



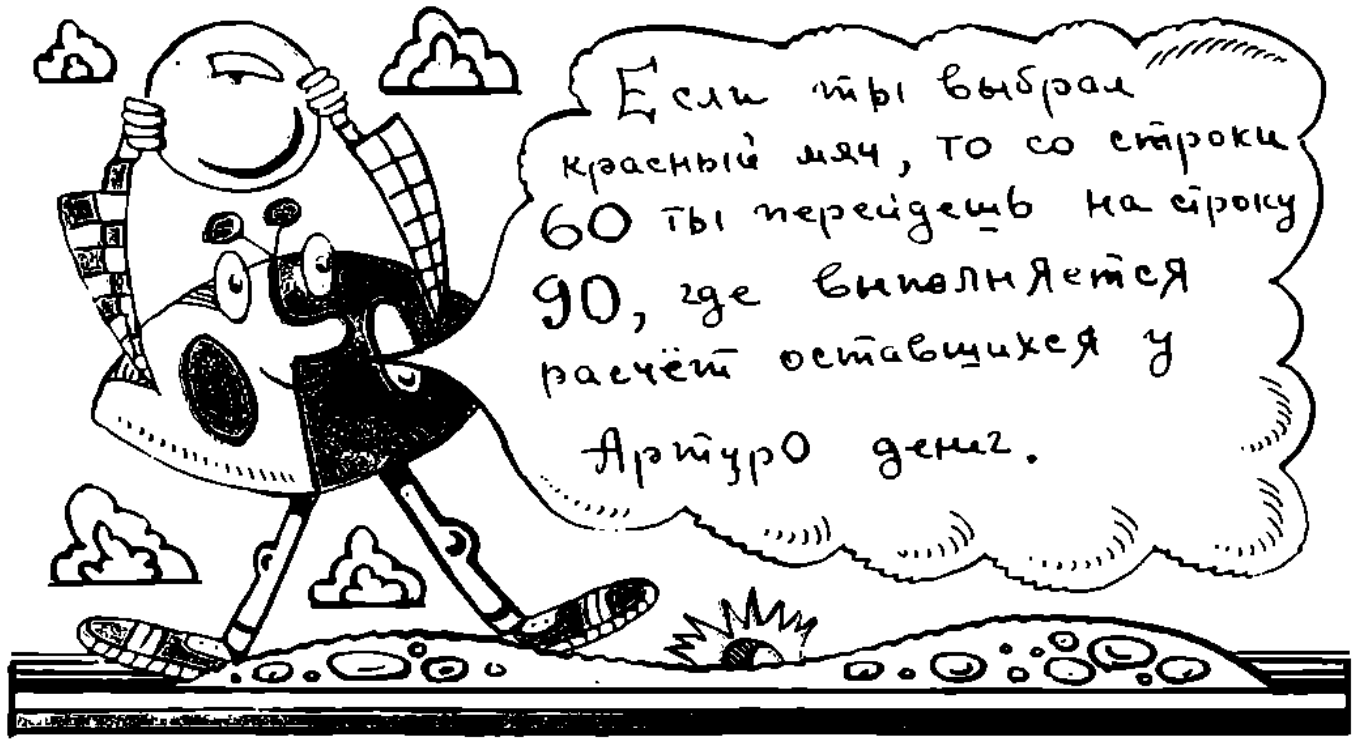

```
60  IF A$="КРАСНЫЙ" THEN 90
70  IF A$="ГОЛУБОЙ" THEN 110
80  IF A$="ОБА" THEN 130
85  PRINT "НЕ ПОНЯЛ. ОТВЕЧАЙ ПРАВИЛЬНО НА МОЙ
    ВОПРОС"
87  GOTO 40
```

Здесь мы указали компьютеру, что он должен делать в зависимости от нашего ответа. Как ты видишь, в зависимости от выбранного варианта, компьютер передаст управление на одну или другую строку программы и, таким образом, выполнит все расчеты в соответствии с выбранным вариантом покупки, не рассчитывая остальное.

Что случится, если твой ответ не совпадет с ожидаемыми ответами в строках 60, 70 и 80? Например, если ты ответишь «зеленый» или «синий», компьютер не перейдет ни на какую строку и будет продолжать выполнять программу последовательно. Он прочитает строки 85 и 87 и повторит вопрос.

Продолжим программу:

```
 90  LET X=D-BR
100  PRINT "ТЫ ДОЛЖЕН ЗАПЛАТИТЬ ";BR;" ПЕСЕТ И У
     ТЕБЯ ОСТАНЕТСЯ ";X;" ПЕСЕТ"
105  GOTO 150
```



Компьютер перейдет к выполнению строки 90 только в том случае, если ты выбрал красный мяч. (Вспомни строку 60.) И когда он перейдет на строку 90, то вычислит X (сколько денег останется у Артуро).

Для этого компьютер вычтет стоимость мяча (BR) из имеющихся у Артуро денег (D). Вспомни, что эти значения заносятся в память компьютера при выполнении строк 10 и 20.

Продолжим:

```
110  LET X=D-BA
120  PRINT "ТЫ ДОЛЖЕН ЗАПЛАТИТЬ ";BA;" ПЕСЕТ
     И У ТЕБЯ ОСТАНЕТСЯ ";X;" ПЕСЕТ"
125  GOTO 150
```

Как видишь, компьютер станет выполнять строку 110 только в том случае, если ты выберешь голубой мяч. (Вспомни строку 70.) Компьютер выполнит все расчеты так же, как и для красного мяча в предыдущем случае, но с ценой для голубого мяча (BA).

Нет смысла объяснять, что будет выполнять компьютер в случае, если ты хочешь купить два мяча; он будет действовать точно так же, но со строки 130.

Проверь себя: сможешь ли ты написать строку 130 и строки после нее? Эти строки можно написать так:

```
130  LET X=D-BR-BA
140  PRINT "ТЫ ДОЛЖЕН ЗАПЛАТИТЬ ";BR+BA;" ПЕСЕТ И
     У ТЕБЯ ОСТАНЕТСЯ ";X;" ПЕСЕТ"
150  END
```

Выполни программу.

■ Обрати внимание на очень интересную деталь в строке 140. Общую сумму, которую нужно заплатить за два мяча, мы обозначили как BR+BA. Компьютер выполнит вычисления автоматически.

Но правильнее было бы, если бы значение этой суммы присвоили какой-то переменной. Например, мы могли бы записать:

```
135  LET Z=BR+BA
```

Тогда строка 140 выглядела бы так:

```
140  PRINT "ТЫ ДОЛЖЕН ЗАПЛАТИТЬ ";Z;" ПЕСЕТ И
     У ТЕБЯ ОСТАНЕТСЯ ";X;" ПЕСЕТ"
```

Выполни программу с двумя вариантами и ты увидишь, что результат одинаков.

Обрати внимание также на то, что в этой программе мы использовали переменные с двумя буквами, для того чтобы было легче их различать.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Команда IF позволяет определить некоторые условия внутри программы. В основном она используется для того, чтобы выполнить условиый переход внутри этой же программы. При этом используется конструкция IF ...THEN или IF ...THEN GOTO (в соответствии с типом компьютера). Другими словами, переход выполняется только в том случае, если условие, заданное после слова IF, выполняется. В противном случае выполняется строка, следующая после строки с командой IF.

- Эта команда очень важна в программе, однако требует определенных знаний и способности к анализу у детей. Если ребенок испытывает трудности, ограничьтесь на время только толкованием, что делает эта комаида в программе, без углублеиия в тему.

- Примеры, представленные в тексте, очень просты. Проанализируйте их вместе с ребенком, чтобы ои убедился, что все использованные в них команды ему уже известны.

- Эта комаида может использоваться в комбинации с другими командами бейсика, такими, как PRINT, LET, STOP, END и др. Некоторые примеры будут объяснеиы позже (например, программа на с. 81).

Другими примерами могут быть:

```
IF X=1 THEN PRINT"..."
IF A=B THEN STOP
IF M=5 THEN LET C=C+1
```

ПРОДОЛЖАЕМ ПРОБОВАТЬ

А сейчас дадим более подробные пояснеиия к команде IF. Ты должен быть очень внимательным, но, если что-либо не поймешь, не огорчайся. Ты поймешь все позже.

Как ты уже знаешь, команда IF позволяет включить некоторые условия внутрь программы. Если условие выполняется, программа переходит на выполнение другой строки, которая нам нужна. Если условие не выполняется, то программа выполняется дальше по порядку.

Помнишь предыдущую программу? В строках 60, 70 и 80 были заданы условия компьютеру. Проанализировав эти условия, компьютер выполнит переход на строки 90, 110 и 130 соответственно.

Мы можем убрать строку программы с номером 60. Кроме того, ты можешь стереть строки 85 и 87. Сделай это и убедись, что результат будет тот же.

Это происходит потому, что если ты выбираешь красный мяч, то не выполняется проверка ни одного из условий в строках 70 и 80, и компьютер продолжает выполнять программу автоматически, считывая и выполняя строку с номером 90.

Прочитай наши пояснения, глядя на программу, и ты все поймешь.

Внимание! Если ты не выберешь «голубой» или «оба», компьютер все равно придет к строке 90, независимо ни от чего. Например, введи «зеленый» (которого не существует) и ты увидишь, что это так.

■ А сейчас поставим Артуро отметки за первую контрольную работу. Если его оценка 1 или 2, то, значит, он учится неудовлетворительно, если 3, 4, 5, то удовлетворительно. Сейчас ты станешь преподавателем Артуро по математике. Оцени его знания, как считаешь нужным.

```
10  PRINT "КАКУЮ ОЦЕНКУ ПОЛУЧИТ АРТУРО?"
20  INPUT A
30  IF A<=2 THEN 60
40  PRINT "АРТУРО — ХОРОШИЙ УЧЕНИК"
50  GOTO 70
60  PRINT "АРТУРО УЧИТСЯ ПЛОХО"
70  END
```

Специальный знак, который используется в строке 30, обозначает «меньше или равно». Поэтому действие строки 30 аналогично тому, что мы сказали бы: если значение, которое мы дали переменной А, меньше или равно 2, то следует перейти на строку 60.

В противном случае программа будет выполняться нормально, то есть со строки 40.

Запусти программу и поставь разные оценки Артуро.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Существует несколько специальных символов для обозначения отношений между членами какого-либо выражения. Эти символы очень похожи на символы, используемые в математике. Разумеется, есть небольшая разница и особенности, которые мы должны учитывать в языках программирования.
  Составим список наиболее употребительных символов. Мы их называем логическими операторами.
  = равно
  < меньше
  > больше
  < = меньше или равно
  > = больше или равно
  < > не равно

- Придумайте какую-нибудь простую программу, в которой ребенок смог бы применить любой из этих логических операторов.

- Внимательно проанализируйте с ребенком комментарии, данные на странице 55 о том, что мы можем убрать строку 60 и какие это будет иметь последствия.

- Команды STOP и END выполняют аналогичные функции, но они не идентичны. STOP представляет собой логический конец программы, а END — физический конец программы. END должно всегда стоять в последней строке программы.

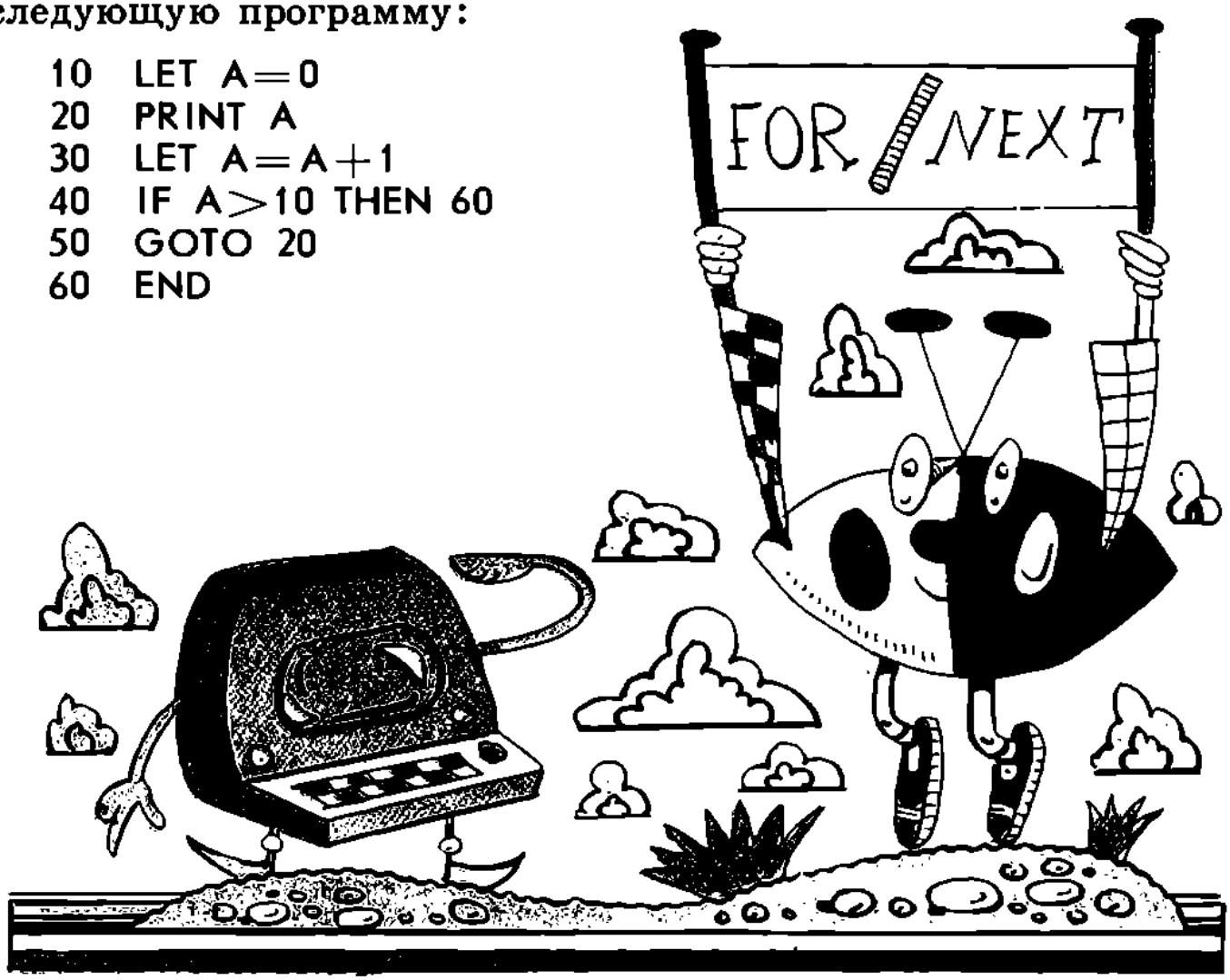
FOR / NEXT

С теми знаниями, что ты уже имеешь, можно составить достаточно сложную программу на языке бейсик. Немного фантазии и терпения, и ты уже способен сделать некоторые, очень интересные вещи.

В этом пункте и в последующих рассмотрим некоторые команды, без которых в принципе можно программировать, но которые очень полезны при составлении программ. Они помогут тебе во многих программах сэкономить время и труд.

Одна из этих команд FOR ... NEXT. Ты увидишь, как она пригодится! Допустим, ты хочешь написать программу, которая напечатает тебе список всех номеров от 0 до 10. Если использовать уже изученные нами команды, ты можешь написать следующую программу:

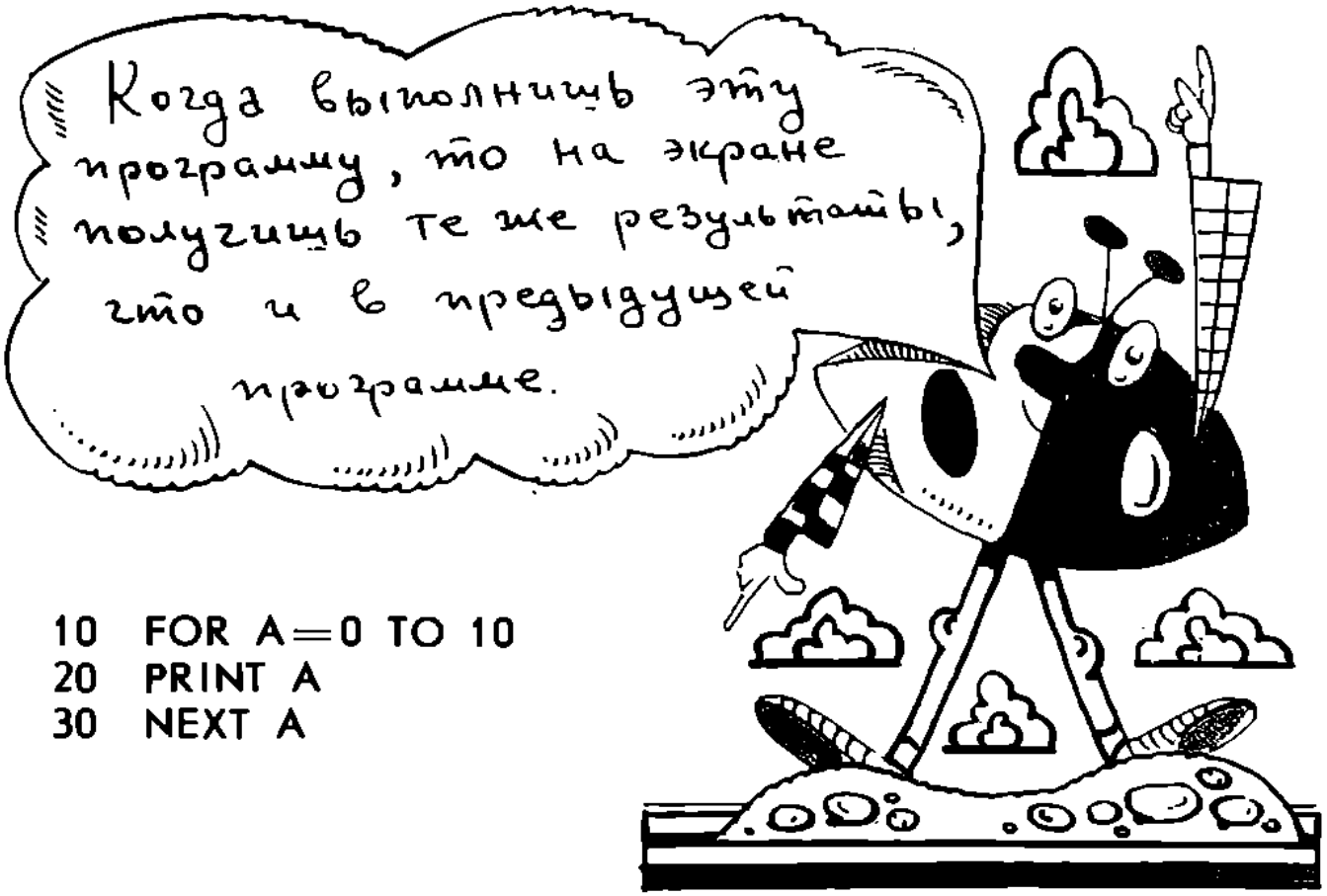
```
10  LET A=0
20  PRINT A
30  LET A=A+1
40  IF A>10 THEN 60
50  GOTO 20
60  END
```



В строке 40 компьютеру поставили такое условие: когда A принимает значение *больше* 10, компьютер должен перейти на строку 60 программы и там остановиться.

Пока это условие не выполняется, программа будет выполняться до строки 50. Здесь она получит приказ вернуться на строку 20, где выведет на экран значение, которое имеет А в этот момент. Потом, в строке 30, присвоит новое значение переменной А.

А вот с командой FOR ... NEXT ты можешь написать ту же программу в следующем виде:




```
10  FOR A=0 TO 10
20  PRINT A
30  NEXT A
```

Выполни ее и убедись в этом. Ты сэкономил много строк программы. Посмотрим, как работает эта команда.

Строки 10 и 30 работают вместе: когда мы имеем строку с FOR ... TO, всегда должны ввести и строку с NEXT.

Обрати внимание на наш пример. В строке 10 мы указали компьютеру, что переменной А мы будем давать значения от 0 до 10, то есть А будет иметь сиачала значение 0, затем 1, потом 2, 3, 4 ... и до 10. Хорошо. Компьютер это запомнит.

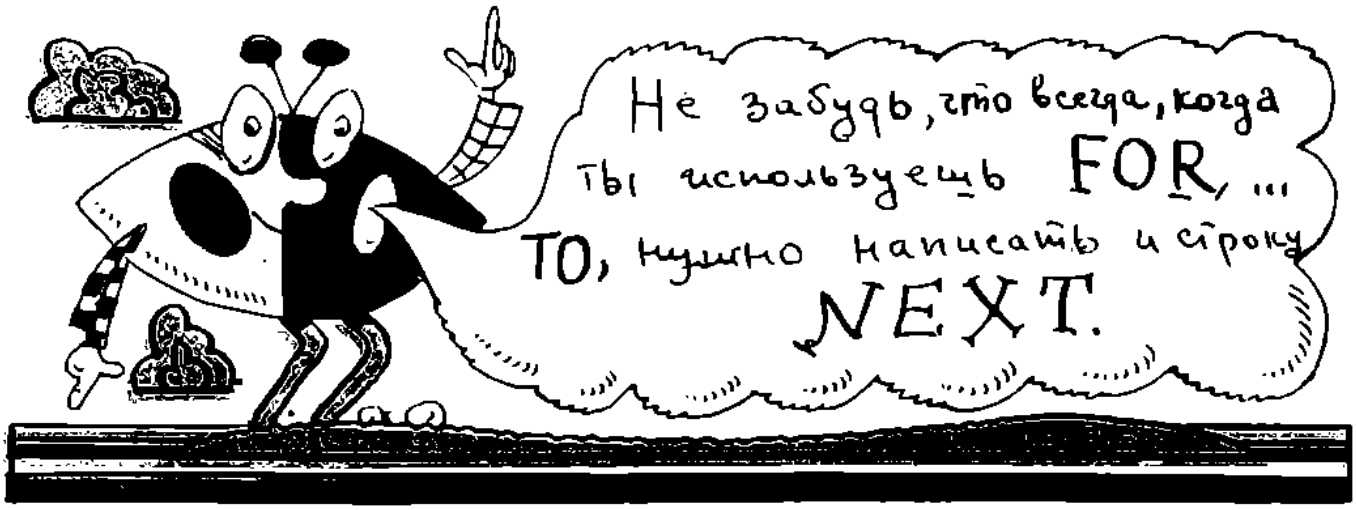
В строке 20 сообщим компьютеру, что мы будем делать с теми значениями, которые присвоены А. В нашем случае мы хотим, чтобы они выводились. В других случаях можем

выполнить с ними любые вычисления, какие захотим. Мы это позже увидим.

В строке 30, используя NEXT, сообщаем компьютеру, чтобы он взял значения A из своей памяти. Это как-будто мы сказали: «Возьми одно за другим все значения A, которые мы тебе дали с помощью FOR ... TO».

Поэтому эта команда имеет две части, которые появляются всегда вместе в программе, хотя они и могут быть разделены другими строками программы.

Между первой частью (FOR ...TO) и второй (NEXT) мы можем указать компьютеру выполнить любые действия, какие захотим. В нашем примере мы ему указали, чтобы он просто вывел на экран значение A.



Значит, программа работает так. В строке 10 компьютер запоминает, что A принимает последовательно значения 0, 1, 2, 3, 4 ... 10. В строке 20 получает приказ вывести заданные значения A. В строке 30 указываем компьютеру взять следующее значение A из тех, что он запомнил. Следовательно, строки программы выполняются так:

В строке 10 запоминаются значения, которые будет иметь A.

В строке 20 выводится 0 (первое значение).

В строке 30 берется следующее значение A (равное 1) и возвращается на строку 20.

В строке 20 выводится 1.

В строке 30 берется следующее значение A (равное 2) и возвращается на строку 20.

В строке 20 выводится 2.

И так далее, пока A не станет равным 10, выведет это значение и в этот момент компьютер закончит выполнение команд программы.

Этот процесс называется *циклом.* Внутри цикла можно приказать компьютеру сделать любые вычисления, какие захотим. Циклы очень полезны, когда хотим повторить какую-либо операцию с различными значениями входящих в нее величин. Как мы видели, компьютер берет только различные значения величин и выполняет с ними одинаковые действия.

■ Посмотрим другую форму использования этой команды. Выполним следующую программу:

```
10  FOR A=1 TO 50
20  PRINT "ПРИВЕТ"
30  NEXT A
```

С помощью этой программы мы приказали компьютеру 50 раз отпечатать слово ПРИВЕТ.

Переменная A не использовалась для выполнения вычислений со своими значениями, а мы ее использовали просто как «счетчик». Интересно, правда?

■ Обрати внимание на программу:

```
10  FOR A=1 TO 10
20  LET X=A*5
30  PRINT X
40  NEXT A
```

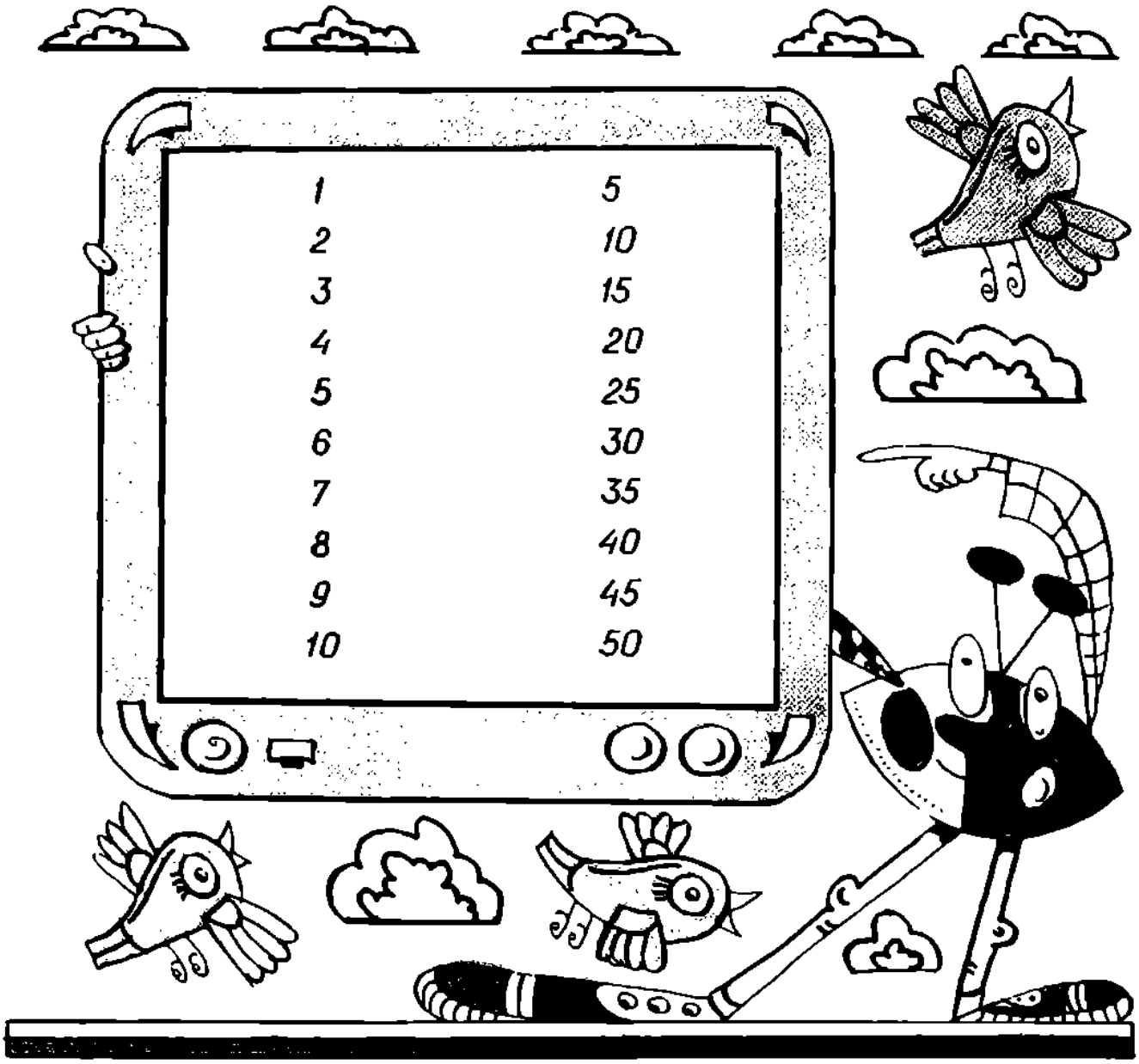
Эта программа означает, что компьютер будет присваивать переменной A последовательно значения от 1 до 10, умножать их на 5 и выводить результат.

Выполни ее.

■ Выполни эту программу, изменив строку 30 так:

```
30  PRINT A,X
```

В этом случае на экране будешь иметь такую таблицу:



В первой колонке будут выведены значения, которые присваиваются A. Во второй колонке будут соответствующие значения X (то есть результат умножения последовательных значений A на 5).

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Команда FOR/NEXT организует цикл, или повторяющуюся последовательность, которая начинается, когда переменной присваивается начальное значение, указанное в команде, и закаичивается, когда ей присваивается последнее значение.

- Эта команда широко используется в программах на бейсике благодаря скорости и удобству, но ее надо учиться использовать постепенно. Не заставляйте детей, не форсируйте ее изучение. Для данного этапа более важно, чтобы ребенок умел использовать ее на практике н понимал весь процесс работы этой команды в общем.

- Во всех играх, которые даны в конце первой части книги, в программы введены некоторые циклы этого типа. Их использование в программах легко объяснить. Вместе с ребенком рассмотрите важность и необходимость применения этой команды.

- В цикле FOR/NEXT можно предусмотреть другое изменение значения переменной. Для этого достаточно добавить в команду слово STEP. Так, еслн мы хотим, чтобы A изменяло свое значение с шагом 2, то напишем:

```
10  FOR A=0 TO 10 STEP 2
```

- Не забудьте, что после окончання цикла компьютер продолжит выполнять программу со следующей строки после слова NEXT.

- Внутри цикла FOR/NEXT можно включить команду (с GOTO), которая может прервать цикл и выйти из него, но никогда нельзя войти в цикл извне.

GOSUB

Ты уже почти настоящий программист. Сейчас мы покажем тебе одну команду, которая не является обязательной, но которая тоже очень полезна.

Она называется GOSUB и всегда используется с другой — RETURN. Ты помнишь GOTO? С помощью GOTO мы заставляли компьютер переходить на другую строку программы. С GOSUB происходит то же самое, но затем с помощью RETURN мы автоматически возвращаемся в программу. А точнее, на строку, которая следует за строкой с GOSUB.

А для чего это нужно?

Представь себе, что у тебя есть программа и совокупность команд, которые несколько раз повторяются. Для того чтобы не писать в программе их несколько раз, ты напиши их всего один раз (обычно в конце программы) и посредством GOSUB будешь выполнять их, когда необходимо. Компьютер будет автоматически переходить на эту часть программы.

Давай рассмотрим пример.

Допустим, что ты заполняешь анкету, где требуется указать имя и фамилию. Программа будет выглядеть так:

```
 10  PRINT "ИМЯ"
 20  INPUT A$
 30  PRINT "ОТЧЕСТВО"
 40  INPUT B$
 50  PRINT "ФАМИЛИЯ"
 60  INPUT C$
 70  PRINT "ДАТА РОЖДЕНИЯ"
 80  INPUT D$
 90  PRINT
100  PRINT
110  PRINT
120  PRINT A$+B$+C$
130  PRINT "РОДИЛСЯ "; D$
```

Запусти программу. Когда будешь отвечать, ставь пробелы после ввода своего имени, отчества, фамилии и т. д. Благодаря этому текст на экране дисплея будет красивее и слова не будут написаны слитно.

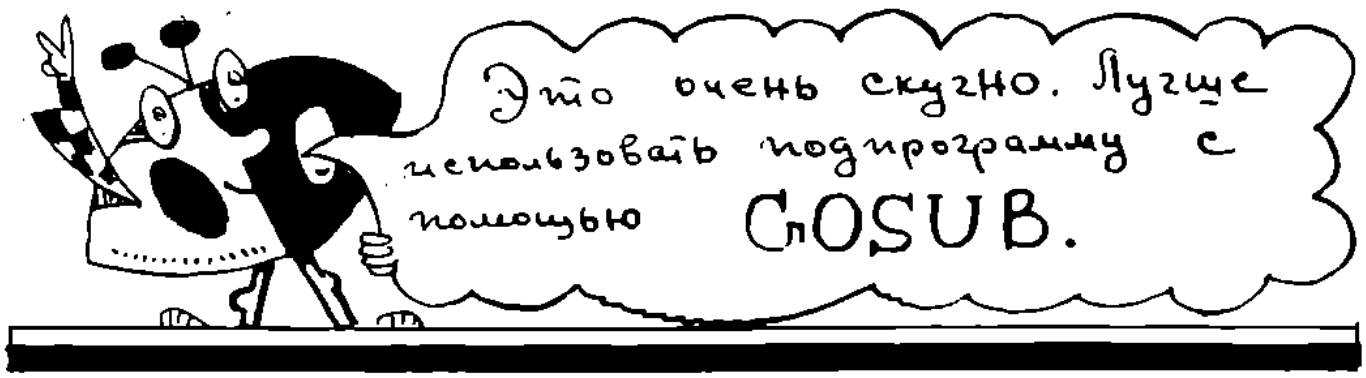
Допустим, что мы хотим, чтобы после каждой строки вопросов была линия с точками для отделения одного вопроса от

другого. Изменим нашу программу, добавив следующие строки:

```
15  PRINT ".—.—.—.—.—.—.—.—.—.—."
35  PRINT ".—.—.—.—.—.—.—.—.—.—."
55  PRINT ".—.—.—.—.—.—.—.—.—.—."
75  PRINT ".—.—.—.—.—.—.—.—.—.—."
```

Сделай изменения и запусти программу. Процесс добавления этих строк был очень скучным, тем более, что ты должен быть внимательным, не пропустить черточки и точки и считать их количество.

А если бы у тебя была программа, где было бы 20 или 30 вопросов, то ты затратил бы целый день, чтобы написать строки, подобные строке 15.



Для того чтобы этого не было, используем *подпрограмму*. Подпрограмма — это все равно, что программа внутри основной программы. Почти всегда она ставится в конце основной программы и может иметь много строк.

■ В нашем примере выполним это повторение строк с помощью подпрограммы:

```
200  PRINT ".—.—.—.—.—.—.—.—.—.—."
210  RETURN
```

А сейчас изменим строки 15, 35, 55 и 75 на строки с таким содержанием:

```
GOSUB 200
```

Каждый раз, когда в программе встречается команда GOSUB, происходит переход на строку 200, где выполняется

печать линий с точками. А в строке 210 выполняется возврат в основную программу. Возврат всегда происходит на строку, следующую за той строкой, где встретилась команда GOSUB.

В нашем примере возврат будет на строки 20, 40, 60 и 80.

Компьютер будет выполнять программу до следующей команды GOSUB и так далее.

■ В программу необходимо добавить еще одну строку, чтобы указать, где будет заканчиваться программа.

Добавь:

135 STOP

Может быть, тебе покажется, что нет большой разницы в том, писать в программе каждый раз строку с точками или обращаться к подпрограмме. Действительно, в нашем примере нет большой разницы, потому что наша подпрограмма очень простая.

Представь себе, что в подпрограмме будет какой-то сложный расчет со многими различными операциями и, чтобы его выполнить, тебе пришлось бы писать их каждый раз внутри программы. Без сомнения ты предпочтешь использовать GOSUB!

Кроме того, не забывай, что подпрограмма может иметь много строк.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Постарайтесь показать ребенку преимущества подпрограмм. Но все-таки не нужно настаивать на использовании этого понятия до тех пор, пока дети не научатся хорошо использовать другие команды.

- Если сочтете нужным, вы можете попытаться придумать пример с более сложными подпрограммами со сложными расчетами или с большими текстами.

- Обратите внимание ребенка на то, что строки алфавитно-цифровых символов могут состоять из групп слов или цифр. Так, дата рождения в приведенном выше примере представлена как символьная переменная.

READ и DATA

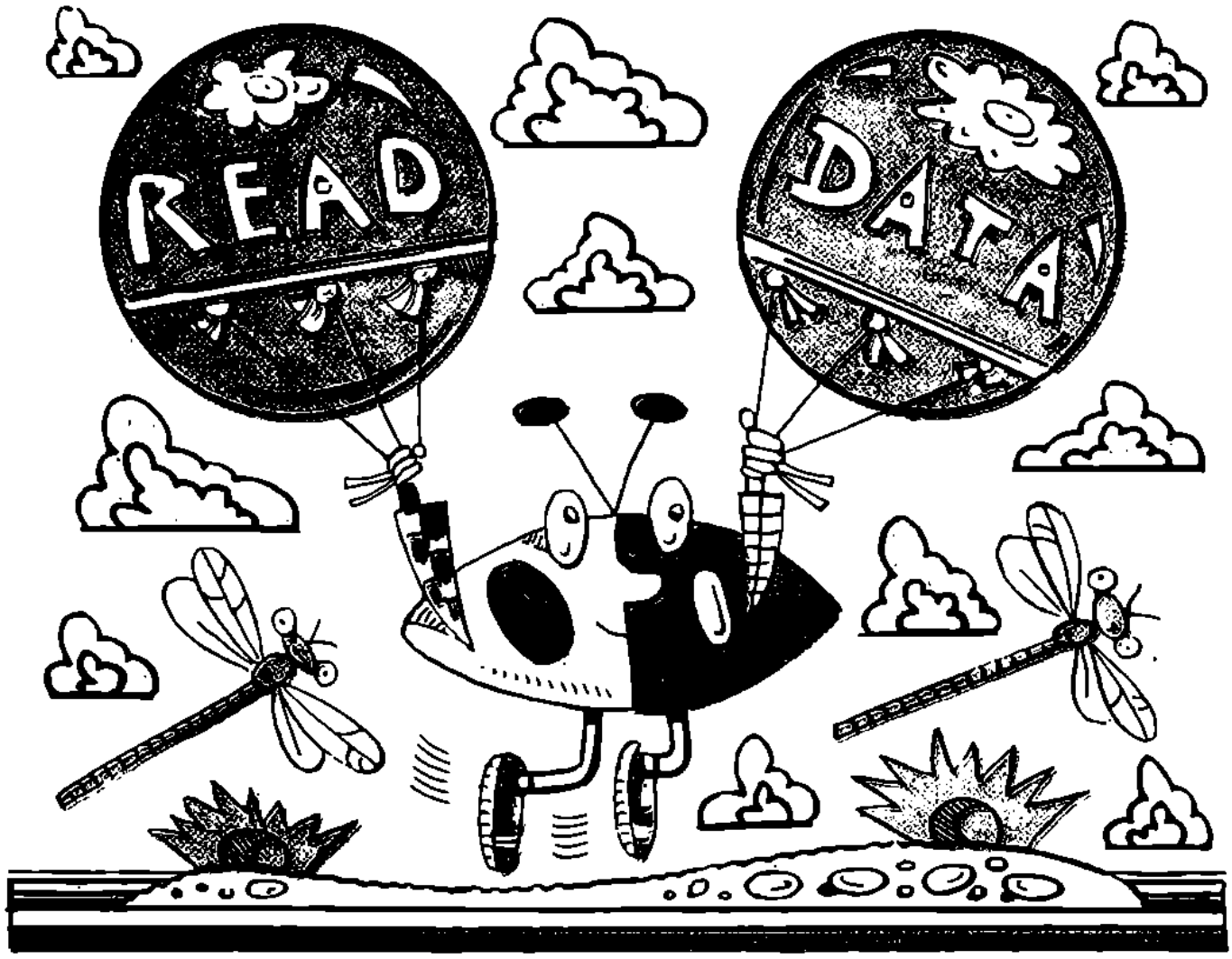
Для экономии времени очень полезны следующие команды — READ и DATA.

Когда в программе необходимо ввести много данных, то команды LET и INPUT нас загружают лишней работой.

Обрати внимание на следующую программу:

```
10  LET A = 5
20  LET B = 7
30  LET C = 4
40  LET D = 2
50  LET E = A + B
60  LET S = A + B + C + D + E
70  PRINT S
```

Выполни эту программу, и на экране появится сумма, которая равна 30.



READ и DATA

■ Тот же результат мы получим, используя команды READ и DATA. С ними программа будет выглядеть так:

```
10  READ  A, B, C, D, E
15  LET E = A + B
20  DATA 5, 7, 4, 2, E
30  LET S = A + B + C + D + E
40  PRINT S
```

Результат такой же — 30. Как видишь, DATA выполняет роль хранилища данных, которые отделяются друг от друга запятыми. Компьютер не берет во внимание эти данные до тех пор, пока не встретит команду READ, принуждающую читать данные, которые задаются в команде DATA.

Компьютер берет значения, находящиеся в DATA, и присваивает их в том же порядке переменным, находящимся в READ.

В нашем примере A присваивается значение 5, B — значение 7 и т. д.

Заметь, что E равно значению суммы A + B.

В одной и той же программе одна команда DATA может иметь больше данных или значений, чем в READ содержится переменных. Данные, которые остаются, ожидают, когда новая команда READ заставит компьютер их считать.

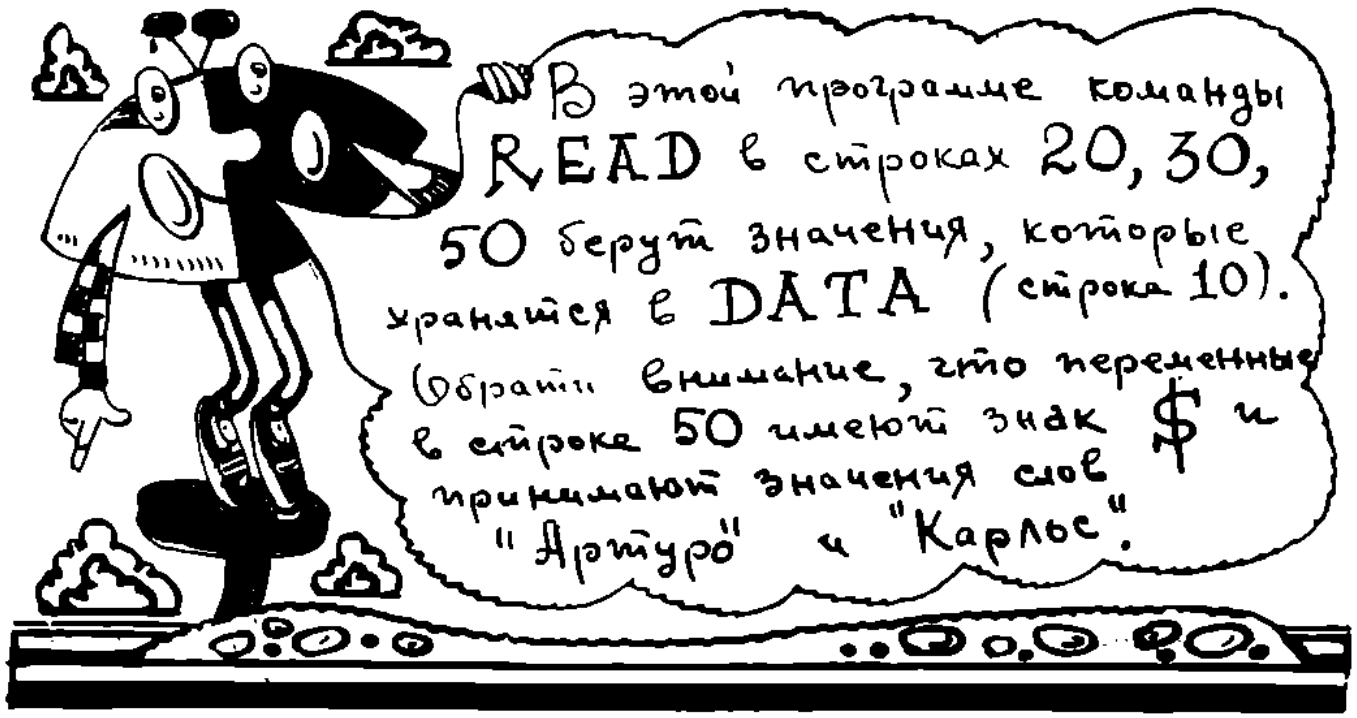
READ и DATA

На основе нашей предыдущей программы создадим программу, в которой ты сможешь увидеть, как разные команды READ присваивают значения, находящиеся в одной команде DATA.

■ Обрати внимание, что этими значениями могут быть слова, числа и выражения.

Рассмотрим программу:

```
10  DATA 1, 2, 3, 4, 5, "АРТУРО", "КАРЛОС"
20  READ A, B, C,
30  READ D, E
40  LET S = A + B + C + D + E
50  READ F$, G$
60  PRINT F$; " И "; G$; "  ИМЕЮТ "; S; " КОНФЕТ "
```



Ты увидел, что если имеем данных больше, чем переменных, то они «ждут», когда компьютер прочитает их с помощью новых операторов READ.

■ Но если компьютер должен считать данные, которых нет (то есть когда имеем больше переменных, чем данных), то на экране появится сообщение о том, что имеется ошибка. Программа не будет выполняться дальше. Удостоверься в этом, добавив к нашей последней программе строку

```
55  READ H, I
```

Посмотри: программа прервется и появится сообщение типа OUT OF DATA IN... (нет данных), указывающее строку, где компьютер встретил переменную, для которой не хватило значений.

■ Это сообщение может также появиться, когда в некотором цикле компьютеру при присвоении данных переменным, стоящим в READ, не хватит значений, указанных в DATA.

Например:

```
10  DATA 5, 7, 4, 2
20  READ A
30  PRINT A, 2*A
40  GOTO 20
```

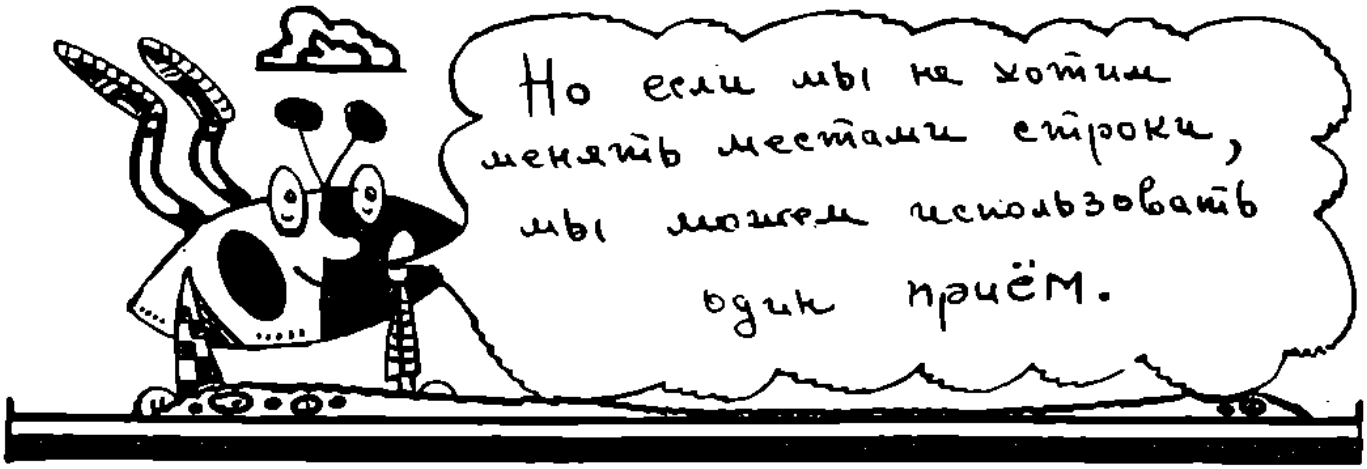
В этой программе компьютер возьмет первое значение (5) и присвоит его переменной A. В строке 30 выведет это значение вместе с результатом умножения этого значения на 2. В строке 40 возвратит управление на присвоение переменной A нового значения, хранящегося в DATA. Когда все значения будут присвоены и когда строка 40 вновь направит компьютер на чтение из DATA, появится сообщение типа OUT OF DATA.

■ Если не быть осторожным, то любая команда этого типа может привести к большим неприятностям в программе. Допустим, что мы хотим просуммировать данные из команды DATA.

Программа может быть следующей:

```
10  DATA 5, 7, 4, 2
20  LET S = 0
30  READ A
40  LET S = S + A
50  GOTO 30
60  PRINT S
70  END
```

Эта программа никогда не будет работать, так как она плохо структурирована. Было бы лучше поменять местами строки 50 и 60. Теперь в этой программе, до прихода к строке 60, выводится результат суммы, а затем компьютер возвращается к чтению данных из DATA, и наступит момент, когда он не найдет очередного значения в DATA.

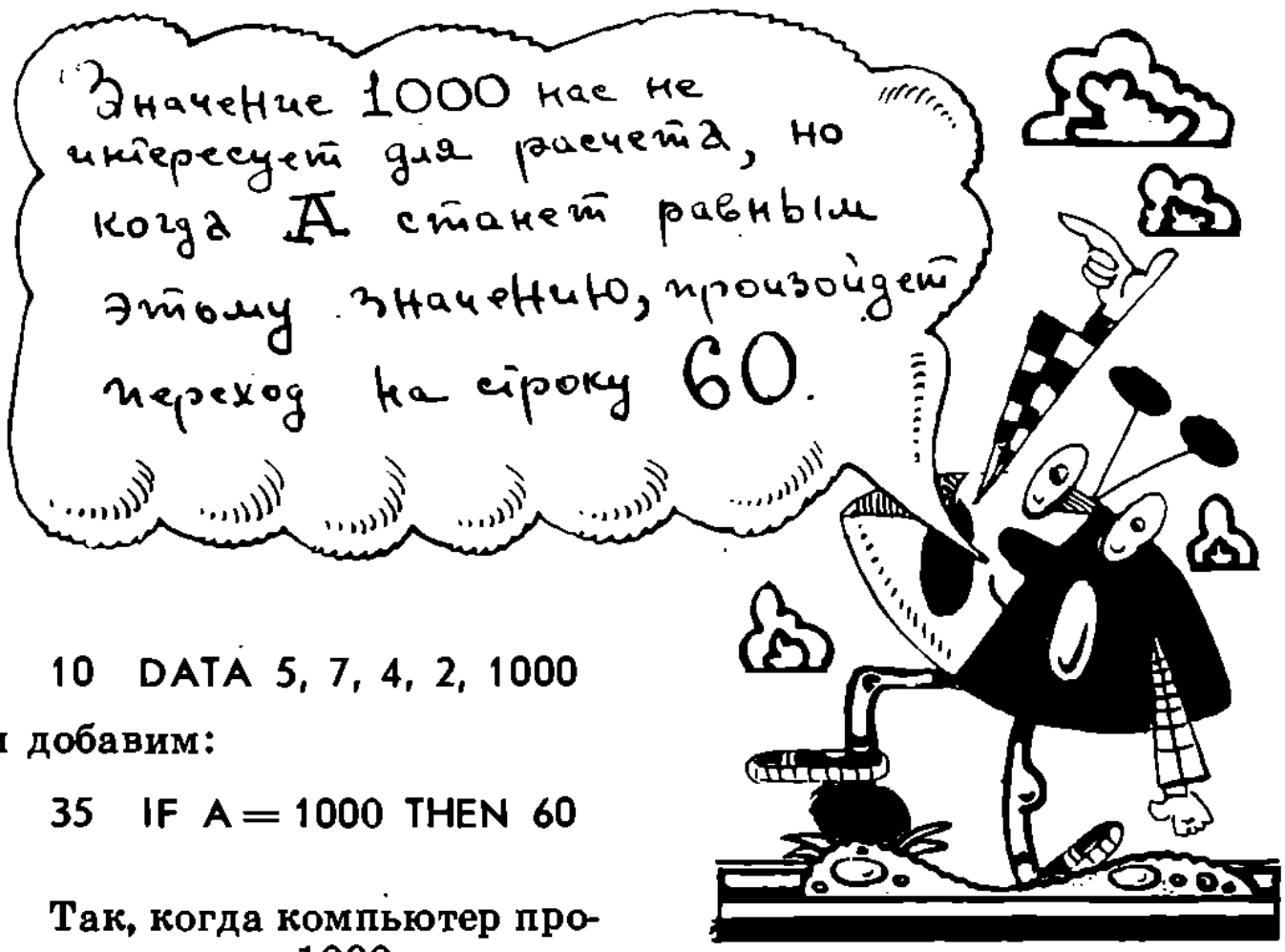


Если программа очень сложная и очень трудно поменять порядок строк программы, вы можете использовать такой прием.

Можно добавить одно фиктивное значение в команду DATA, то есть значение, которое нас не интересует для наших расчетов.

Кроме того, вводим еще одну команду IF, для того чтобы, когда компьютер прочтет это фиктивное значение, программа стала выполняться с интересующей нас строки.

В нашем примере преобразуем строку 10 так:




```
10  DATA 5, 7, 4, 2, 1000
```

и добавим:

```
35  IF A = 1000 THEN 60
```

Так, когда компьютер прочтет значение 1000, то произойдет переход на строку 60, где будет выведено значение суммы S, накопленной до этого момента, и программа закончится.

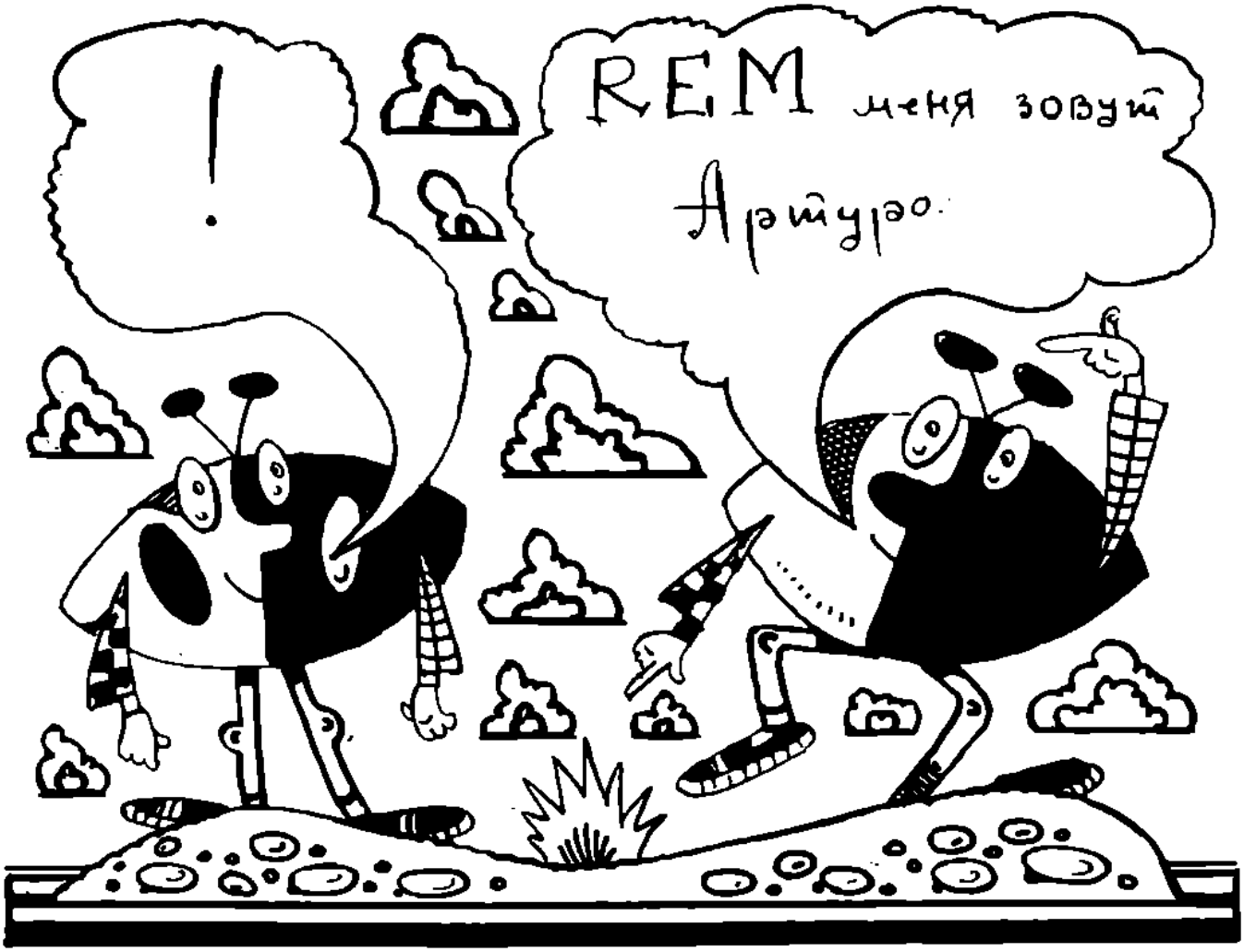
ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Эта команда присваивает переменным, стоящим в READ, значения данных, находящихся в команде DATA. Объясните ребенку, что одна команда DATA может «обеспечить» несколько команд READ.
- Лучше всего ставить DATA в начале илн конце программы.
- Эту команду можно широко использовать только в том случае, если все команды предыдущих глав уже хорошо усвоены.
- Если ребенок не в достаточной степени понял эту команду, не настаивайте на ее использовании. Всегда можно вернуться к ней, если вы сочтете это необходимым, так как она является только важным дополнительным средством в программировании.

REM

Последние команды, которые ты здесь узнал, немного сложнее. А эта очень простая! К тому же она помогает нам вспоминать.

Команда REM никаким образом не влияет на выполнение программы. Компьютер не принимает ее в расчет. Она служит нам для того, чтобы не забыть, что мы делаем.



Иногда бывают случаи, когда ты смотришь на программу, которую сам же составил, и не можешь вспомнить, для чего ты написал некоторые строки этой программы. Для этого ты можешь вставить между строками команду REM с пояснением там, где считаешь необходимым, чтобы облегчить чтение программы, когда вернешься к ней через какое-то время.

Например, в программе на странице 70 можно добавить строку:

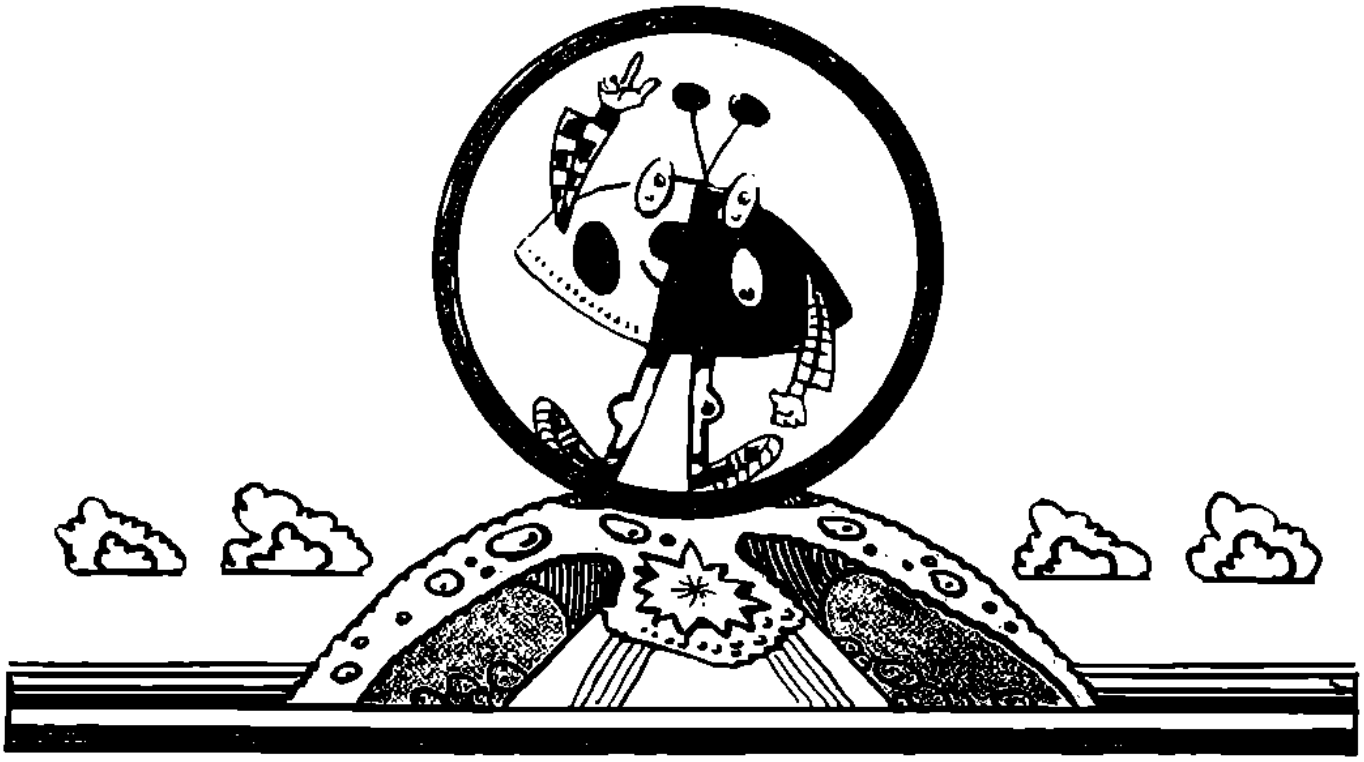
```
 5 REM ЗНАЧЕНИЯ, КОТОРЫЕ ПРИНИМАЕТ А
25 REM ВЫВЕДЕМ ЗНАЧЕНИЕ А И ЕГО
   УДВОЕННОЕ ЗНАЧЕНИЕ
```

Можно добавить любые, какие хочешь, пояснения с помощью REM. Ты уже знаешь, что они не влияют на программу, но ты их можешь всегда посмотреть, если выведешь на экран текст программы. Ты можешь написать любой текст в командах REM. Важно, что они тебе напоминают в тех местах, где ты их поставил, о шагах, которые выполняются в программе.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Попрактикуйтесь с ребенком на предыдущих программах, вставляя, где необходимо, пояснения.

- Не жалейте в программе комментариев, включенных с помощью команды REM! Команда REM не исполняется компьютером, а служит только для пояснения, замечаний.



INT

Эта команда, или лучше сказать, эта функция тоже очень полезна в программах вычислений и в игровых программах, в чем ты убедишься в дальнейшем.

Все, что делает INT,— это отбрасывает в числе дробную и оставляет только целую часть. Например, от числа 2,489 останется 2, а от числа 5,24689—5.

■ Рассмотрим пример. Напиши:

```
10  LET A = 2.3489
20  PRINT A
```

Запустим программу и на экране получим 2.3489. Обрати внимание (это очень важно), что вместо запятой, отделяющей целую часть числа от дробной, используется точка. Это всегда делается при программировании.

■ Добавим теперь строку:

```
15  LET A = INT(A)
```

Выполни теперь программу.

На экране ты получишь число 2. Осталась только целая часть числа. Необходимо знать, что функция INT не округляет. Точно отделит целую часть и отбросит дробную, то есть с INT число 2.9 преобразуется в 2, а не в 3.

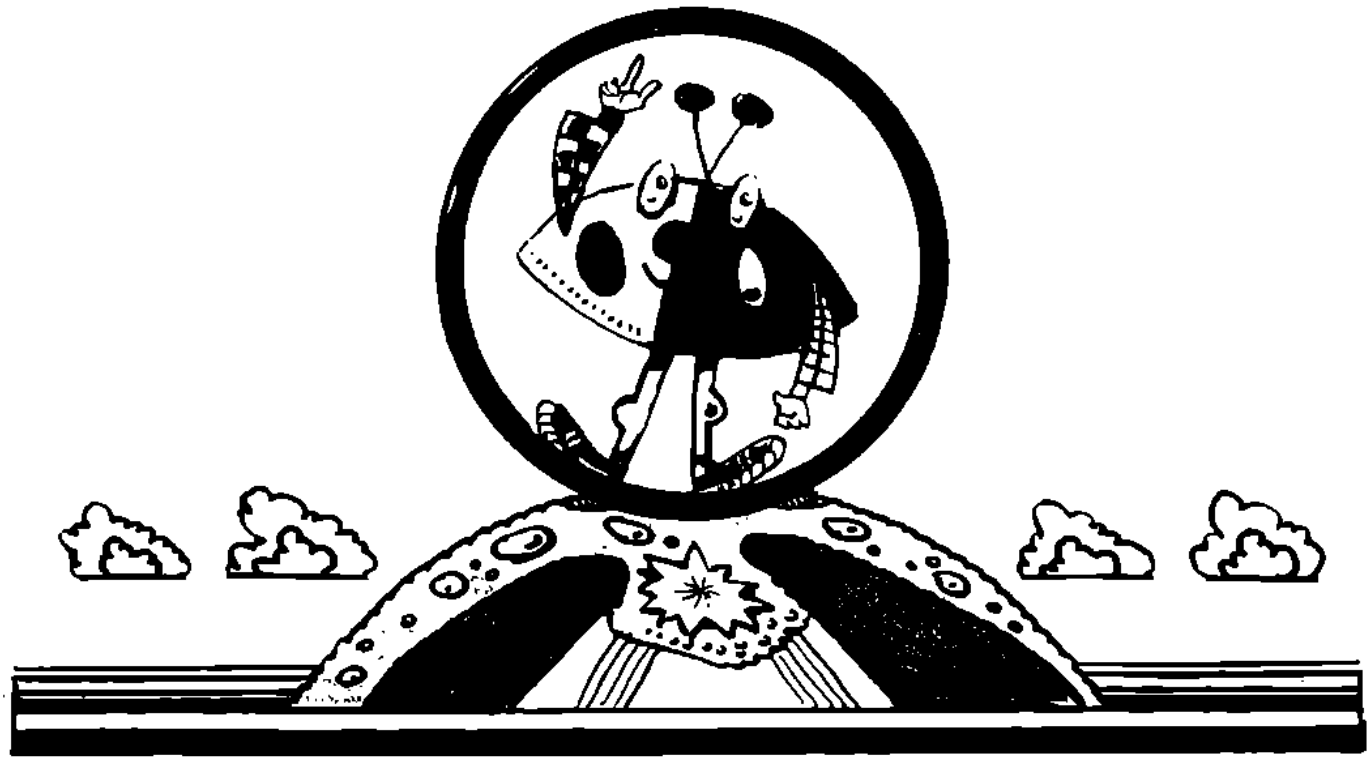
Как мы тебе уже говорили раньше, функция INT очень полезна в языке бейсик. Со временем ты это поймешь. В следующем пункте ты увидишь, как интересно ее можно использовать.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- В предыдущей программе сделайте так, чтобы ребенок несколько раз изменил строку 10, используя числа на свой выбор.

- Возможно, что ребенок не проявит интереса к этой функции. Предупредите его, что она очень важна при создании игр, расчетах и в более сложных программах.

- Подчеркните важность использования точки для отделения целой и дробной части. Кроме того, нужно знать, что в информатике не используется точка для выделения тысяч. Так, две тысячи сто нужио писать 2100.



RND

Знаешь ли ты, что такое случайность? Много раз ты говоришь: «Это произошло случайно», то есть ты хочешь сказать, что это *могло бы и не произойти* или *могло бы произойти что-то другое.*

Именно с помощью функции RND ты можешь ввести элемент случайности в программу. Для чего это нужно? Например, чтобы играть с компьютером, как ты увидишь немного позже, чтобы моделировать с помощью компьютера различные ситуации, которые могли бы возникнуть в реальной жизни и на которые влияет случайность.

■ Давай разбираться постепенно. С функцией RND компьютер создает или генерирует случайные числа между 0 и 1 (исключая 1).

Например:

```
10  LET A = RND(X)
20  PRINT A
```

Каждый раз, когда ты будешь запускать эту программу, ты получишь на экране число, больше или равное 0 и меньше 1.

Например:

```
0.53118986
0.83547393
0.97816467
0.24576398
```

■ Чтобы получить случайные числа, например от 0 до 10, необходимо умножить на 10 значение RND (X). Для этого изменим

строку 10 на следующую:

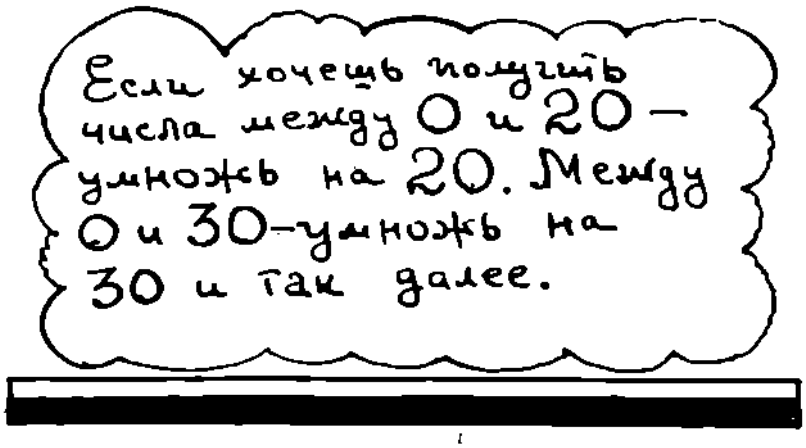
```
10  LET A = RND(X) * 10
```

Добавим еще строку:

```
30  GOTO 10
```

Получим список случайных чисел от 0 до 10, не включающий 10. Например:

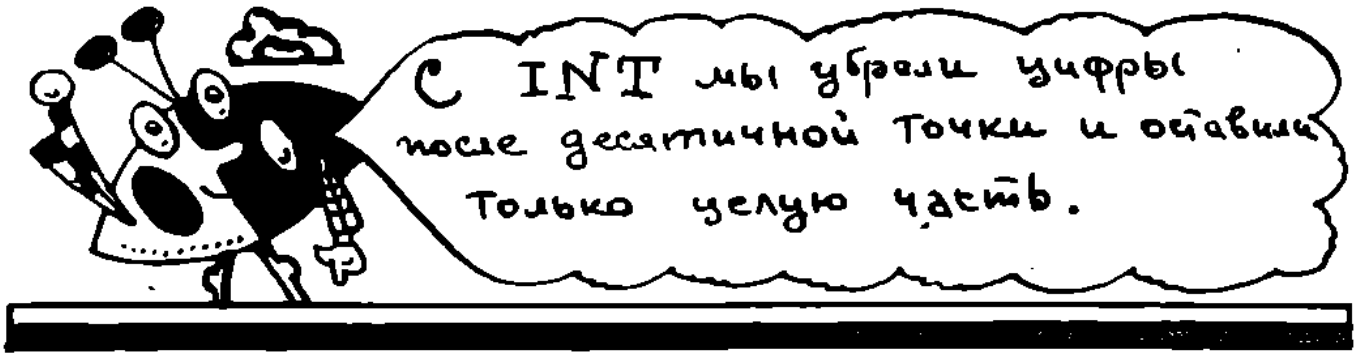
```
0.18976534
6.00789654
0.36928912
3.00000000
8.34698762
```



Ты помнишь функцию INT? Она тебе сейчас пригодится. С ее помощью ты можешь оставить только целые части этих чисел.

■ Измени строку 10 так:

```
10  LET A = INT(RND(X)*10)
```



Получим список целых чисел от 0 до 10 (без 10).

■ Допустим, ты хочешь проанализировать числа, которые получаются при выбрасывании игральной кости (кубика). По формулам из предыдущего примера ты напишешь:

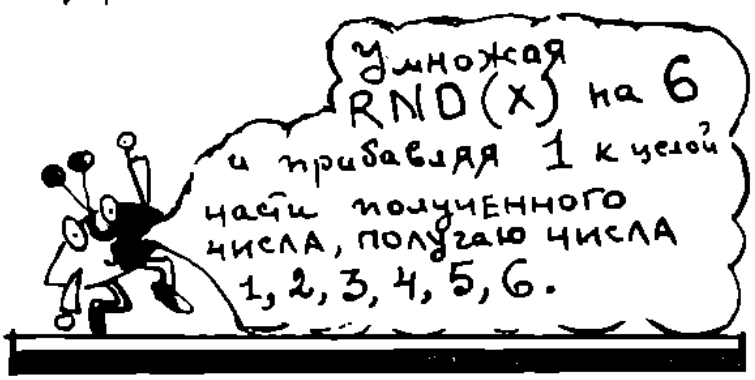
```
10  LET A = INT (RND  (X)*6)
```

Внимание! Эта функция даст тебе значения между 0 и 6 (без 6). И, кроме того, кубик (кость) не имеет 0. Поэтому ты должен добавить 1, для того чтобы получить числа от 1 до 6, включая и 1 и 6.

Тогда ты получишь:

```
10  LET A = INT (RND (X)*6) + 1
20  PRINT A; " ";
30  GOTO 10
```

Запустив программу, ты получишь на экране список чисел, которые можно получить при бросании кубика.



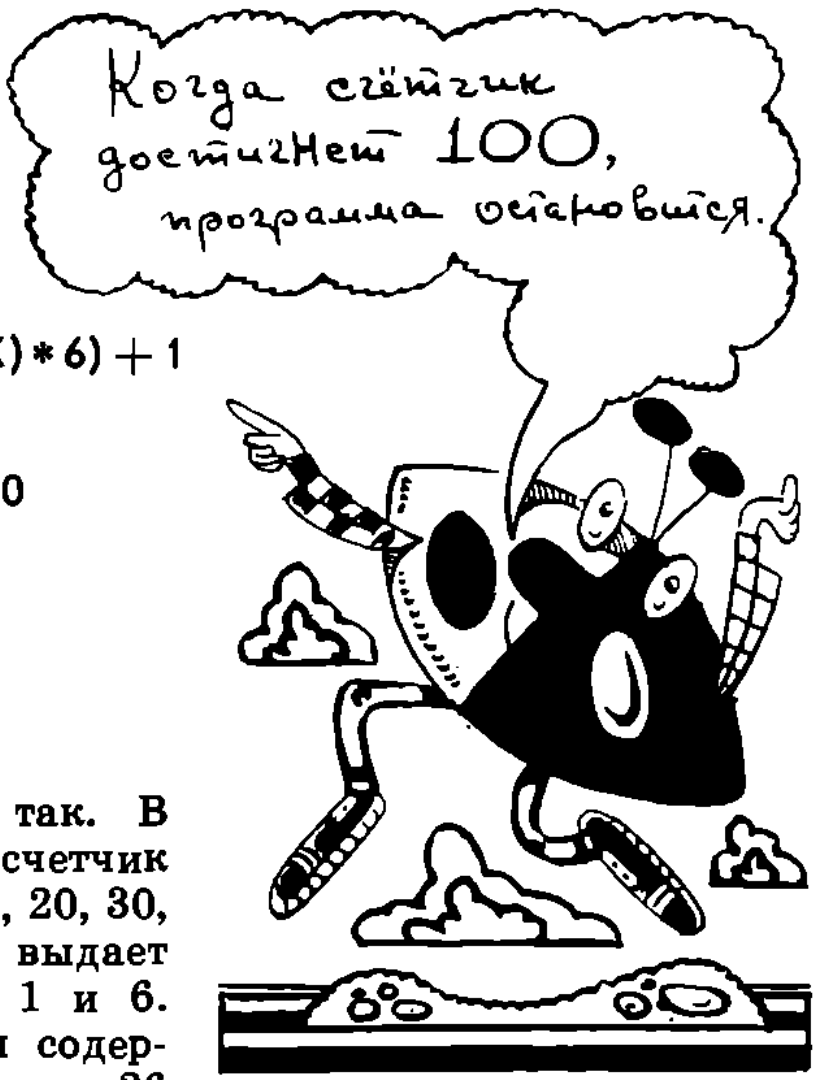
Ты смоделировал с компьютером реальную ситуацию, когда действительно играешь в кости. Убери строку 30, и тогда каждый запуск программы будет равносилен бросанию кости.

Допустим, мы хотим бросить кость 100 раз и посмотреть, какие числа можем получить при этом. Для того чтобы сделать это, изменим предыдущую программу, добавив «счетчик». Когда счетчик досчитает до 100, программа остановится. Добавим строки:

```
 5  LET C=0
25  LET C=C+1
26  IF C=100 THEN 40
30  GOTO 10
40  END
```

Теперь ты получил программу следующего вида:

```
 5  LET C=0
10  LET A=INT(RND(X)*6)+1
20  PRINT A;" ";
25  LET C=C+1
26  IF C=100 THEN 40
30  GOTO 10
40  END
```



Программа работает так. В строке 5 устанавливаем счетчик C в 0. Исполняя строки 10, 20, 30, компьютер формирует и выдает случайные числа между 1 и 6. В строке 25 увеличиваем содержимое счетчика на 1. В строке 26 указываем компьютеру, что, когда счетчик достигнет значения 100, нужно перейти на строку 40, в которой программа заканчивается.

Хорошо проанализируй эти пояснения, держа программу перед собой. Потом выполни ее.

А сейчас начинается самое интересное! Ты знаешь, что при бросании кости с равной вероятностью может появиться любое число (между 1 и 6).

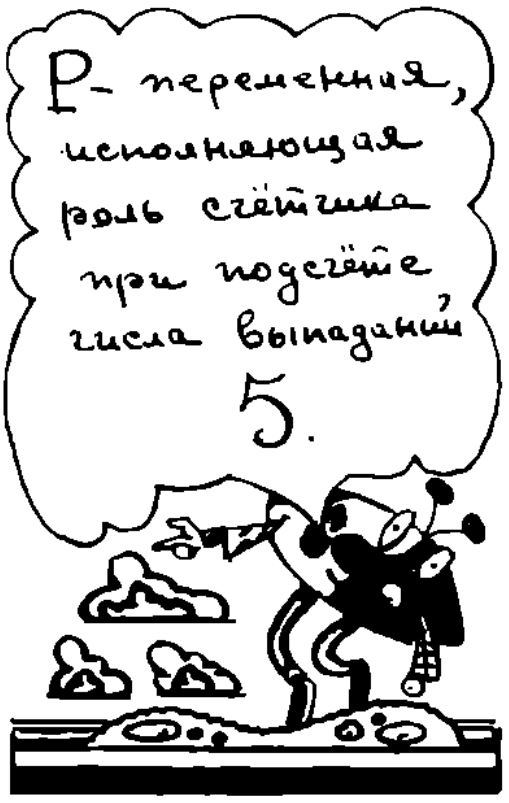
Теоретически мы можем предположить, что за сто бросков любой номер (например, 5) повторится 16 раз. (Ясно, что из 100 бросков, деленных на 6 номеров, получится 16 возможностей повторения одного номера.) Но на практике случайно может получиться большее или меньшее число. Мы можем это проверить, выполнив 100 бросков кости, но это очень скучно и долго.

С помощью компьютера мы можем смоделировать бросание кости. Гораздо быстрее можем узнать, что произойдет при бросках, и подсчитать, сколько раз выпадет один и тот же номер (например, 5).

Для того чтобы сделать это, нам нужно только добавить в предыдущую программу несколько строк, с помощью которых осуществляется подсчет появлений числа 5.

Добавим строки:

```
 7  LET P=0
22  IF A=5 THEN LET P=P+1
35  PRINT C, P
```



и поменяем строку 26:

```
26  IF C=100 THEN 31
```

Добавим также

```
31  PRINT
32  PRINT
33  PRINT
34  PRINT
```

С этими последними строками окончательный результат на экране будет отделен от других чисел.

В строке 7 очищаем (обнуляем) счетчик, в котором будем подсчитывать, сколько раз выпадет число 5. В строке 22 сообщаем компьютеру, что каждый раз, когда выпадает число 5, нужно счетчик увеличить на единицу.

В строке 35 компьютер выдаст общее количество бросков и укажет, сколько раз выпало число 5.

Как видишь, тебе нужно было немного изменить строку 26, чтобы программа была хорошо структурирована.

В законченном виде программа будет выглядеть так:

```
 5  LET C=0
 7  LET P=0
10  LET A=INT(RND(X)*6)+1
20  PRINT A;" ";
22  IF A=5 THEN LET P=P+1
25  LET C=C+1
26  IF C=100 THEN 31
30  GOTO 10
31  PRINT
32  PRINT
33  PRINT
34  PRINT
35  PRINT C, P
40  END
```

Выполни и проанализируй эту программу. В конце компьютер тебе выдаст, что проб было 100, и укажет, сколько раз при этом выпало число 5.

Обрати внимание на те преимущества, которые тебе дает компьютер при работе с ним. Допустим, ты хочешь знать, сколько раз выпадет число 5 при 1000 бросков.

Артуро попытался бросить 1000 раз кость, но он устал, бросив ее только 168 раз. С компьютером все это значительно быстрее.

Сотри строку 20 (для того, чтобы не выводились все числа, которые получаются) и измени строку 26:

```
26  IF C=1000 THEN 35
```

Измени также строку 35:

```
35 PRINT "В "; С;" БРОСКАХ ВЫПАЛО "; Р;" РАЗ ЧИСЛО
   ПЯТЬ"
```

Запусти программу. Тебе нужно немного подождать, но, конечно, значительно меньше, чем если бы ты ждал, пока Артуро тебе даст результат.

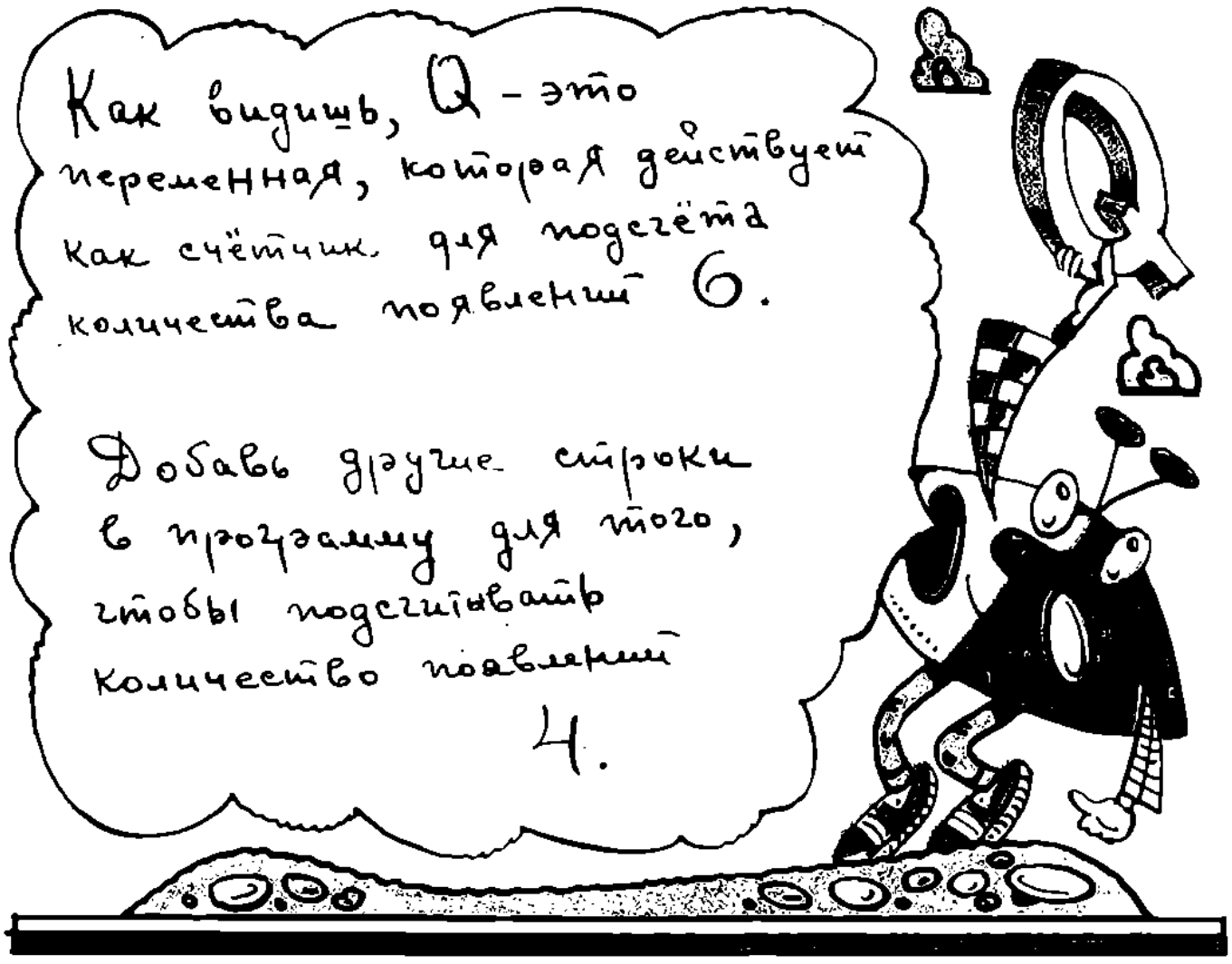
■ Хотел бы ты преобразовать программу так, чтобы компьютер тебе еще сообщил, сколько четверок и сколько шестерок выпало в тысяче бросков? Это очень легко. Ты должеи только иметь еще по одному счетчику для этих чисел. Например:

```
 6 LET Q = 0
23 IF A = 6 THEN LET Q = Q + 1
```

И изменить строку 35 на:

```
35 PRINT "В "; С;" БРОСКАХ ВЫПАЛО ";Р;" ПЯТЕРОК И ";
   Q; " ШЕСТЕРОК"
```

Сделай изменения и для других чисел.



ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Точно так же, как и в предыдущих пунктах, эту функцию иадо изучнть последовательно и постепенно. Едиистенно, что здесь действительно важио — это то, что ребеиок должеи усвоить поиятие случайиости, с которым связано использоваиие RND.

- Необходимо вызвать интерес ребеика после выполиеиия каждого шага изучения и добиться, чтобы он хорошо понимал преимущества следующего шага. Выполияйте все в виде эксперимеита.

- Фуикция RND в разных компьютерах может выполняться по-разному. Обычно в бейсике требуется, чтобы после RND присутствовало какое-то число в скобках. Это значеиие может быть просто X. Не беспокойтесь, если эта переменная уже присутствует в программе, где используется функция RND.

- Предварительно посмотрите ииструкцию своего персонального компьютера. Виимательно изучите, как используется функция RND, и сделайте соответствующне измеиения в наших программах (если это иеобходимо).

- В некоторых компьютерах требуется в иачале программы ставить команду RANDOMIZE для получения различных серий случайных чнсел каждый раз, когда программа выполияется. В противном случае каждый раз будет выдаваться та же серия. И тогда, иапример, пишут:

```
10 RANDOMIZE
20 LET A = RND(X)
30 PRINT A
```

СХЕМЫ

Нужно, чтобы ты знал, что во время программирования удобно использовать некоторые схемы или диаграммы для того, чтобы представить свои идеи более ясно и хорошо организовать предстоящую работу.

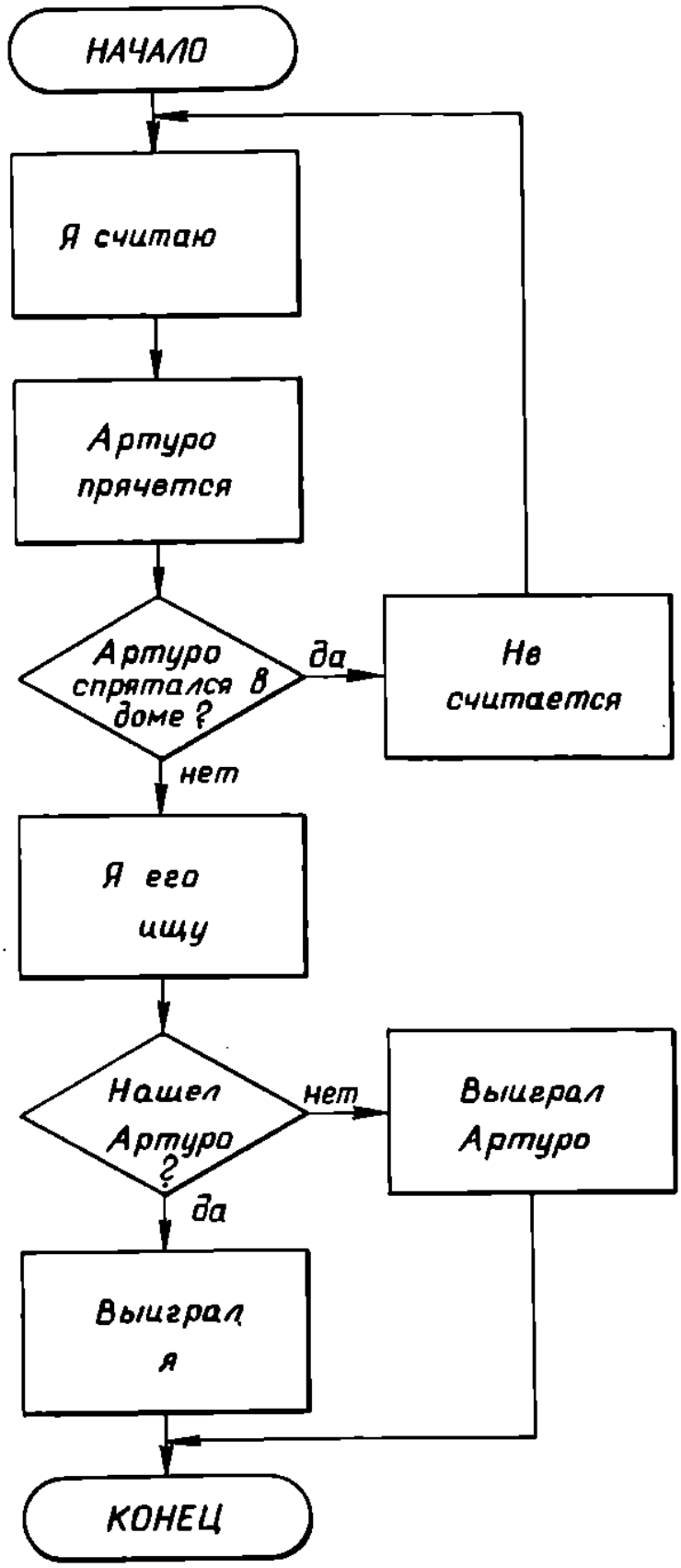
Кроме того, эти схемы помогают понять, какова структура программы и как она сконструирована. Ты можешь сразу увидеть, как она будет выполняться.

Лучше всего сделать схему до того, как начнешь программировать, чтобы было понятно, что объяснять компьютеру. Особенно это важно, когда программа достаточно сложная.

Мы не будем глубоко изучать это. Только рассмотрим некоторые очень простые примеры для того, чтобы ты понял, как это делать.

Ты помнишь игру в прятки и правила этой игры, описанные в начале книги?

Если бы нам нужно было сделать схему алгоритма этой игры, то мы бы сделали что-то подобное тому, что мы привели на этой странице. Посмотри внимательно и ты увидишь, что схема очень логично построена.



■ Возьмем очень простую программу, которая выводит иатуральные числа от 0 до 10.

Схема ее приведена слева на следующей странице.

Программа соответственно имеет вид:

```
10  LET A = 0
20  PRINT A
30  LET A = A + 1
40  IF A>10 THEN 60
50  GOTO 20
60  END
```

Запусти программу и ты увидишь, как появится список чисел от 0 до 10. Тебе покажется это очень простым, так как ты знаешь хорошо все команды.

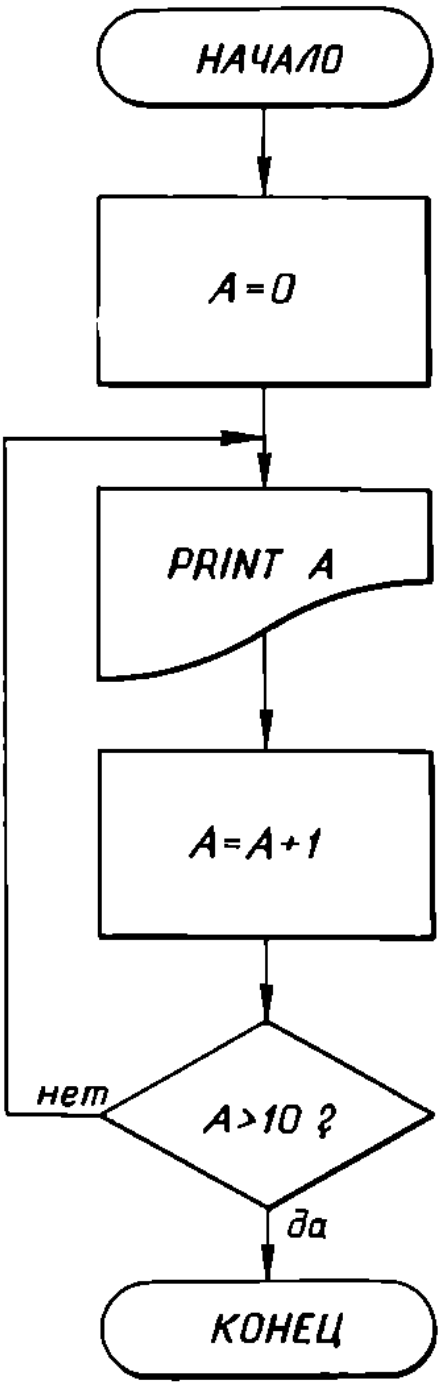
■ Как будет выглядеть схема программы, которая вычисляет удвоенные числа между 0 и 50?

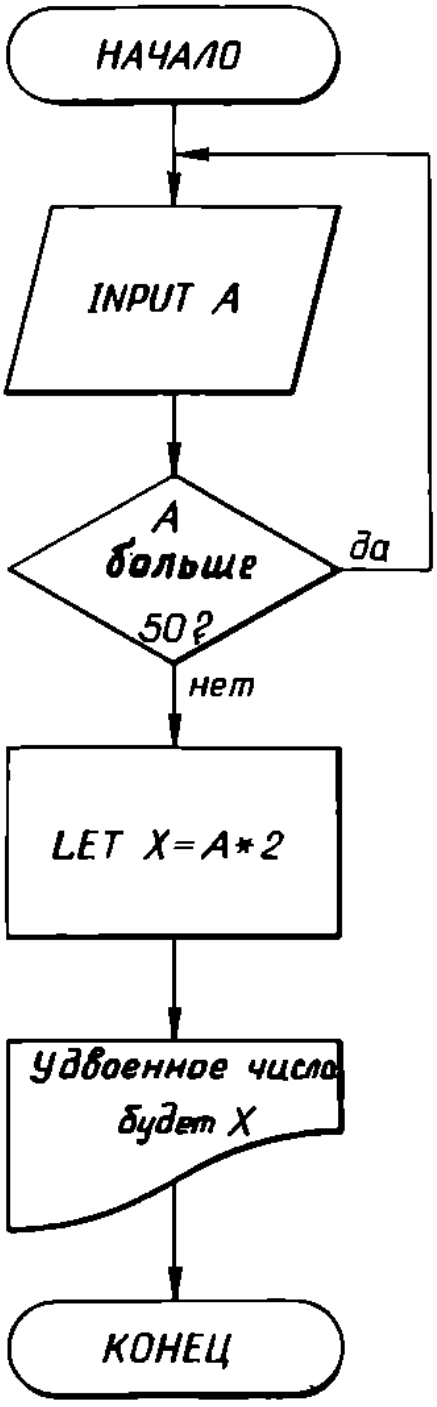
Эта программа имеет вид:

```
10  PRINT "ВВЕДИ ЧИСЛО МЕЖДУ 0 И 50"
20  INPUT A
30  IF A>50 THEN 10
40  LET X = A *2
50  PRINT "УДВОЕННОЕ ЧИСЛО БУДЕТ "; X
60  END
```

А схема этой программы приведена справа иа следующей странице.

СХЕМЫ

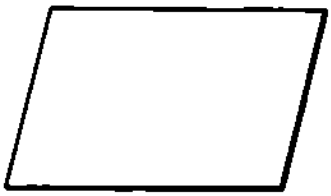

НАЧАЛО
A=0
PRINT A
A=A+1
A>10 ?
нет
да
КОНЕЦ


НАЧАЛО
INPUT A
A
больше
50?
да
нет
LET X=A*2
Удвоенное число
будет X
КОНЕЦ

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

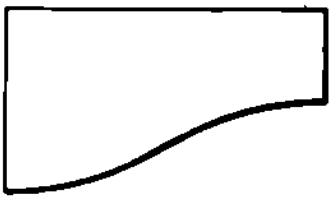
- Для изображения схем алгоритмов и программ используются стандартные символы:

Начало или конец программы

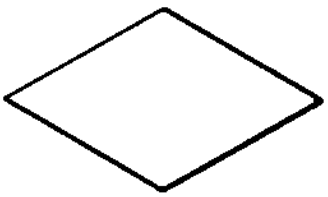


Ввод или вывод данных (INPUT)

Присвоение переменным значений (LET) или выполнение вычислений



Печать результатов (PRINT)



Логические условия (IF...)

- Необходимо научить ребенка составлять и рисовать схемы алгоритмов. Здесь нет детального описания этого, так как это не является основной целью книги.

- Когда ребенок овладеет всеми командами бейсика, приведенными в этой книге, попытайтесь выполнить с ним схемы алгоритмов программ, которые мы уже разбирали.

ИГРЫ

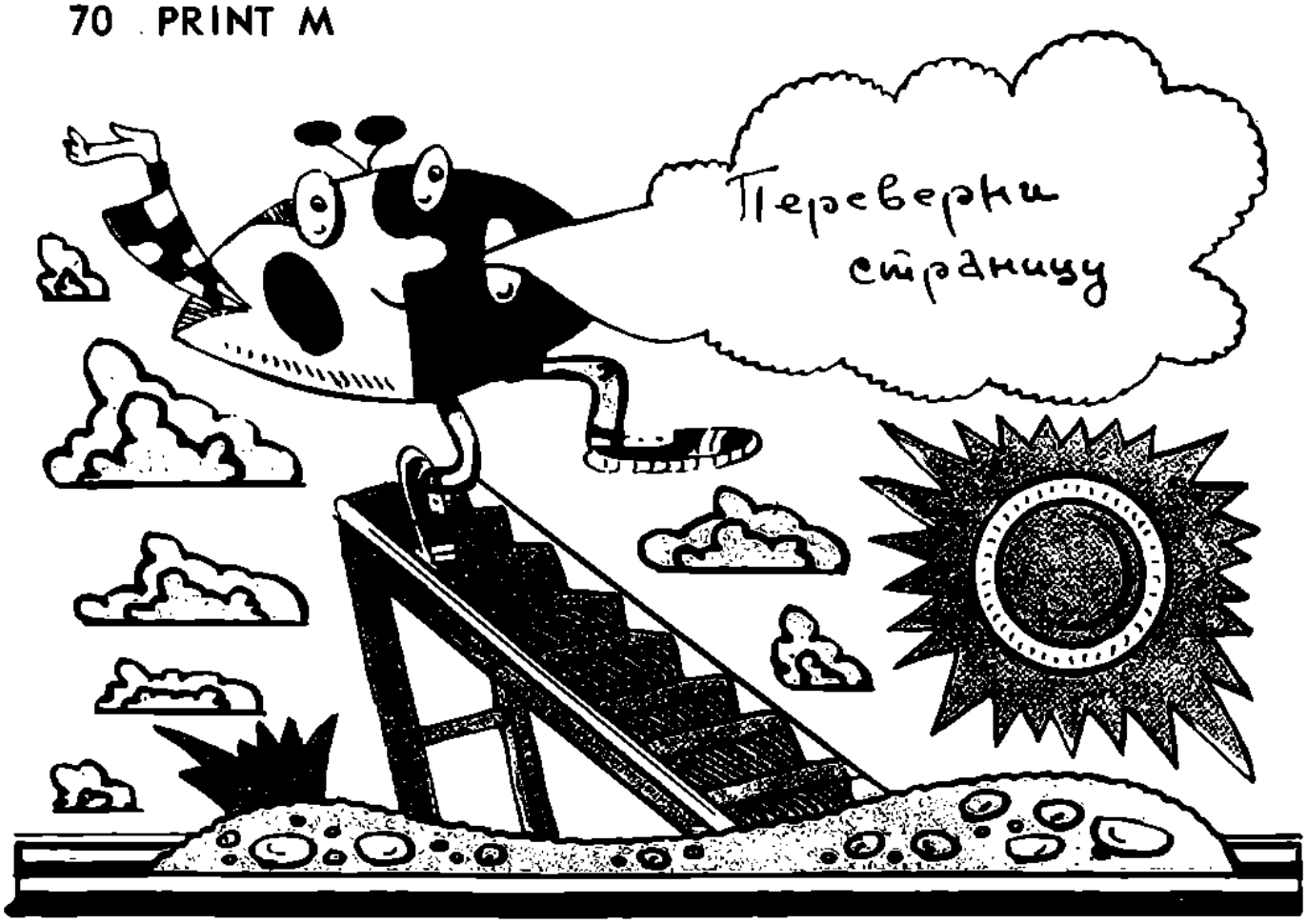
В этом пункте мы рассмотрим некоторые игры или случаи применения программ, которые могут быть интересны, полезны и увлекательны.

Может быть, они помогут тебе сделать собственные игры. Во всех программах используются команды и понятия, которые ты уже хорошо знаешь. Попробуй сначала проанализировать их, чтобы понять, как они работают.

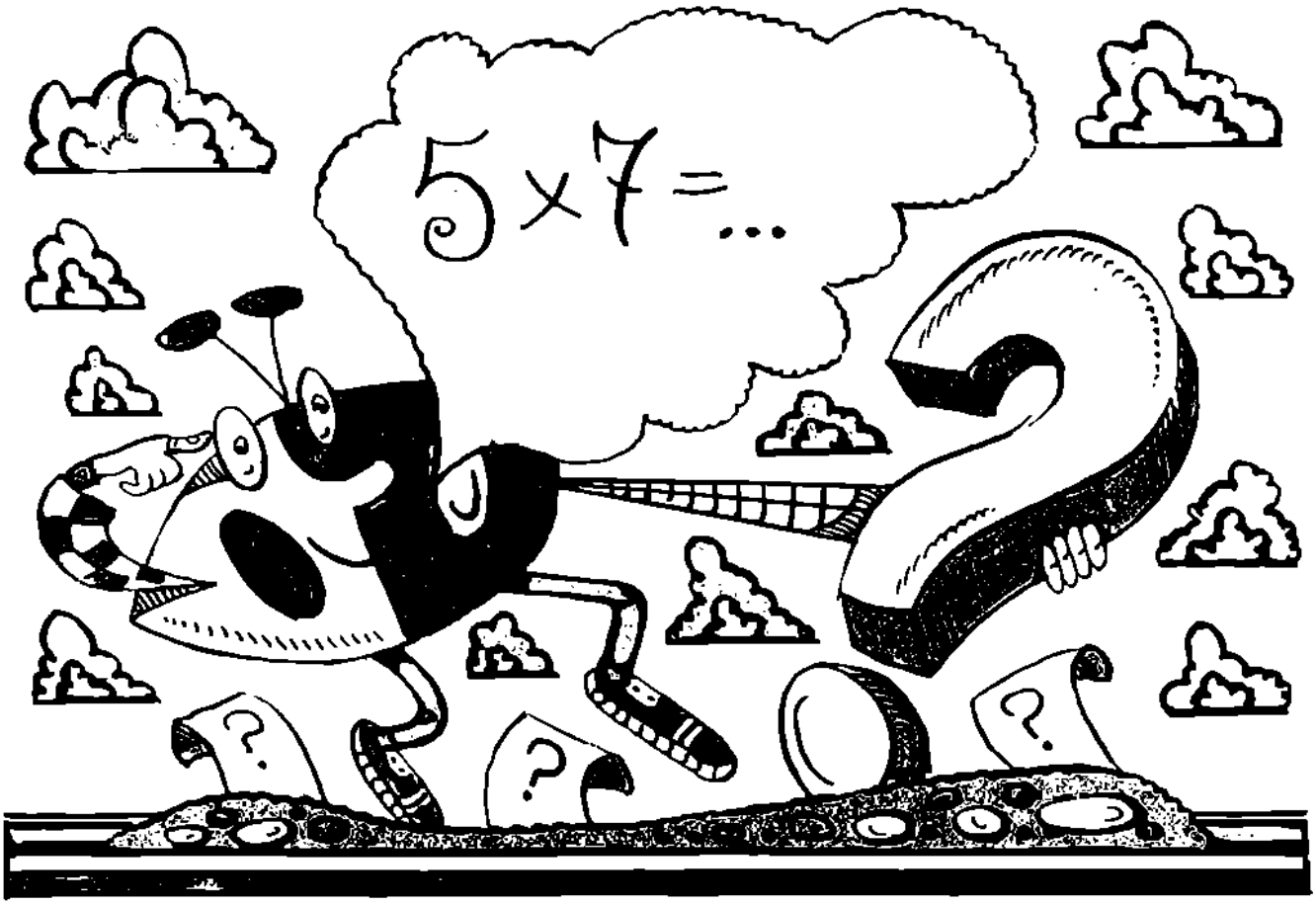
УЧИМСЯ УМНОЖАТЬ

Эта программа поможет тебе попрактиковаться в знании таблицы умножения.

```
10  PRINT "ЗАДАЙ КАКОЕ-ТО ЧИСЛО";
20  INPUT N
30  PRINT N
40  FOR I = 1 TO 10
50  PRINT N; " * "; I; "=";
60  INPUT M
70  PRINT M
```



```
 80  IF M = N * I THEN 110
 90  PRINT "ТЫ ОШИБСЯ. ПОПРОБУЙ СНОВА!"
100  GOTO 50
110  PRINT "ОЧЕНЬ ХОРОШО!"
120  NEXT I
130  END
```



ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- В этой программе используются комаиды PRINT, INPUT, FOR/NEXT, IF и GOTO.

ИГРАЕМ В ПРЯТКИ С АРТУРО И КОМПЬЮТЕРОМ

Как мы тебе обещали в начале книги, поиграем в прятки с Артуро. Программа очень простая. Если ты ее проанализируешь, то поймешь, как она действует, и убедишься, что ты бы смог сам сделать подобную программу.

Игра заключается в следующем. Артуро будет прятаться в саду. Он может спрятаться только в одном из 10 мест. Обозначим их номерами от 1 до 10. Если Артуро спрячется в доме,

то игра прекращается: как только ты начинаешь его искать — он проиграл.

Когда ты найдешь Артуро, компьютер скажет, сколько ты попыток сделал, чтобы найти его. Каждый раз, когда ты начинаешь играть, Артуро прячется в другом месте. Посмотрим, как быстро ты найдешь Артуро!

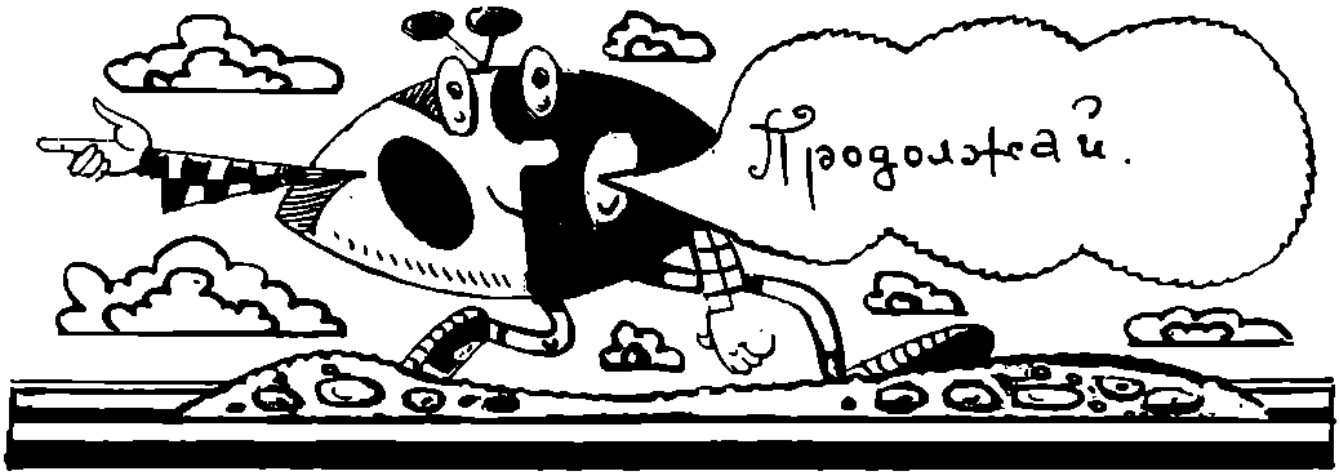
```
  5  REM ПОТАЙНОЕ МЕСТО
 10  LET A=0
 20  LET R=INT(RND(X)*10)+1
 25  PRINT "НАПИШИ НОМЕР МЕСТА"
 30  PRINT "ГДЕ АРТУРО?"
 35  LET A=A+1
 40  INPUT N
 50  PRINT N
 60  IF R=3 THEN 200
 70  IF R=N THEN 100
 80  PRINT "ЗДЕСЬ ЕГО НЕТ"
 90  GOTO 25
100  PRINT "ТЫ ЕГО НАШЕЛ!"
101  PRINT
102  PRINT
103  PRINT "В "; A;" ПОПЫТОК"
105  GOTO 220
200  PRINT "НЕ СЧИТАЕТСЯ: АРТУРО СПРЯТАЛСЯ В ДОМЕ"
210  PRINT "НАЧИНАЙ СНОВА ИСКАТЬ"
220  END
```

ОТГАДЫВАНИЕ ЧИСЕЛ

Эта программа похожа на предыдущую. Компьютер загадывает число между 1 и 100. Ты должен угадать это число. Когда компьютер тебе отвечает, он дает направление поиска. Отгадай число с наименьшим количеством попыток!

```
  5  REM УГАДАЙ
 10  LET A=0
 20  LET R=INT(RND(X)*100)+1
 30  PRINT "ЗАГАДЫВАЮ ЧИСЛО МЕЖДУ 1 И 100"
 40  PRINT "ПОПЫТАЙСЯ ОТГАДАТЬ ЕГО"
 50  PRINT "ДАЙ СВОЕ ЧИСЛО"
 60  INPUT X
```

```
   63  PRINT
   65  PRINT X
   70  LET A = A + 1
   80  IF X<R THEN 2000
   90  IF X = R THEN 3000
 1000  PRINT
 1005  PRINT
 1010  PRINT "ОШИБСЯ. МОЕ ЧИСЛО МЕНЬШЕ"
 1020  GOTO 50
 2000  PRINT
 2010  PRINT
```




```
 2020  PRINT "ОШИБСЯ. МОЁ ЧИСЛО БОЛЬШЕ"
 2030  GOTO 50
 3000  PRINT
 3010  PRINT
 3020  PRINT "ТОЧНО!!!"
 3030  PRINT
 3040  PRINT "МОЕ ЧИСЛО ";R
 3050  PRINT
 3060  PRINT "ТЫ ЕГО ОТГАДАЛ ЗА "; A;
                "ПОПЫТОК"
 3070  PRINT
 3080  IF A >= 5 THEN 3100
 3090  PRINT "ХОРОШИЙ РЕЗУЛЬТАТ"
 3093  PRINT
 3095  GOTO 3105
 3100  PRINT "НЕПЛОХО, НО ПОПРОБУЙ УЛУЧШИТЬ
          РЕЗУЛЬТАТ"
 3105  PRINT
 3110  END
```

УМНЫЙ КОМПЬЮТЕР

Компьютер может угадывать твои мысли. Проверь это, работая со следующей программой. Возьми ручку, бумагу и делай то, что тебе скажет компьютер.

```
 10  PRINT "БУДУ УГАДЫВАТЬ ТВОЕ ЧИСЛО"
 20  PRINT
 30  PRINT "ЗАДУМАЙ ЧИСЛО"
 40  PRINT "НО НЕ ГОВОРИ ЕГО МНЕ"
 50  GOSUB 300
 60  PRINT "УМНОЖЬ ЕГО НА 6"
 70  GOSUB 300
 80  PRINT "ПРИБАВЬ 10 К РЕЗУЛЬТАТУ"
 90  GOSUB 300
100  PRINT "РАЗДЕЛИ РЕЗУЛЬТАТ НА 2"
110  GOSUB 300
120  PRINT "ОТНИМИ 5 ОТ РЕЗУЛЬТАТА"
130  GOSUB 300
140  PRINT "СКАЖИ МНЕ, ЧТО ПОЛУЧИЛОСЬ"
150  INPUT X
160  LET Z = X + 5
170  LET Y = Z * 2
180  LET H = Y - 10
```


```
190  LET N = H/6
200  PRINT
210  PRINT
225  PRINT
210  PRINT
220  PRINT "ТЫ ЗАДУМАЛ ЧИСЛО "; N
```

```
225  PRINT
230  STOP
300  PRINT "ЕСЛИ ГОТОВ"
310  PRINT "НАЖМИ ENTER"
320  INPUT A$
330  PRINT
340  PRINT
350  RETURN
```

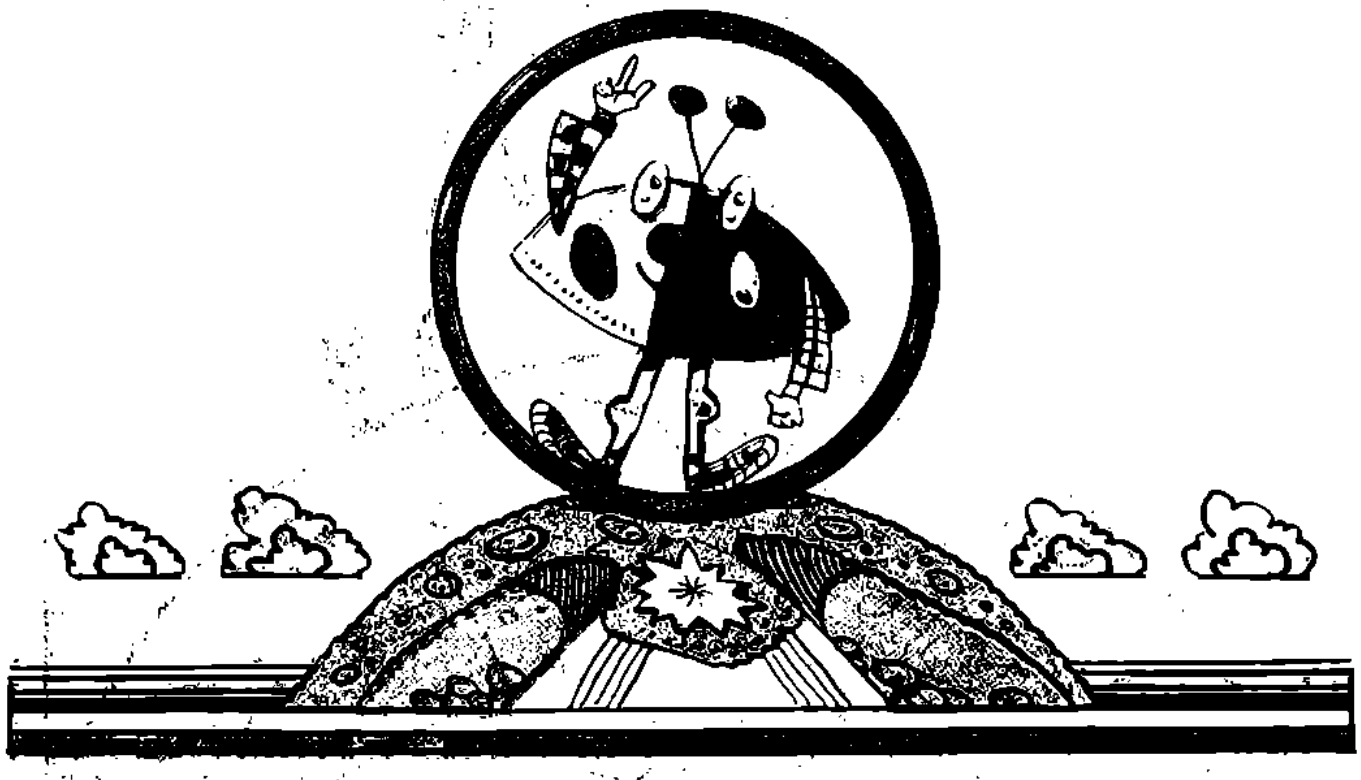
ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

● В этой программе использованы команда GOSUB и все другие, знакомые уже вам команды.

Ну вот... мы уже закончили. Ты уже настоящий программист (маленький, конечно). Тебе еще много надо учиться, чтобы знать хорошо язык программирования бейсик..

Но не пугайся, так как то, что ты выучил и понял из этой части книги, является основой, и все остальное будет легче освоить. Сейчас тебе нужно практиковаться в работе с компьютером. Составляй собственные программы и без стеснения пробуй делать их с использованием разных команд, разиыми способами.

Если захочешь узнать больше о программироваиии и языке бейсик, то Артуро тебе в этом поможет во второй части книги, которая называется «Углубленный бейсик для детей».



Часть II

УГЛУБЛЁННЫЙ БЕЙСИК ДЛЯ ДЕТЕЙ

Привет! Возможно ты уже знаешь меня. Я Артуро и в первой части книги мы вместе изучили много интересного о языке программирования бейсик. Сейчас ты настоящий программист. Давай еще изучим некоторые вещи, чтобы ты стал специалистом. Когда мы закончим изучать книгу, ты сможешь составлять очень сложные и полезные программы.

Если ты не изучал со мной первую часть книги, потому что у тебя были некоторые знания об этом языке, то посмотри страницу, где дан краткий обзор того, что мы уже знаем.

Пошли дальше?

Здесь описаны команды бейсика, которые мы изучали в первой части, чтобы ты вспомнил, что мы уже знаем.

PRINT — выводит на экран то, что ты укажешь.

LET — позволяет присвоить переменной любое значение, какое ты укажешь.

INPUT — делает так, чтобы компьютер ожидал данные, которые ты можешь ввести с помощью клавиатуры компьютера во время выполнения программы.

GOTO — дает указание компьютеру перейти на то место программы, которое ты укажешь.

IF ...THEN — приказывает компьютеру выполнить то или иное действие в зависимости от выполнения заданного условия.

FOR/NEXT — заставляет компьютер выполнять циклы (то есть повторять одни и те же действия определенное количество раз).

GOSUB — переводит компьютер на выполнение определенной подпрограммы. RETURN возвращает его в основную программу.

READ/DATA — оператор DATA запоминает в программе некоторые данные, которые затем читает оператор READ.

REM — вводит в программу, не нарушая ее, любые комментарии.

INT — удаляет дробную часть числа, оставляя только его целую часть (в числах, записанных в виде десятичной дроби).

RND — вводит элемент случайности в программу.

Схемы — позволяют нам с помощью графического материала увидеть структуру программы.

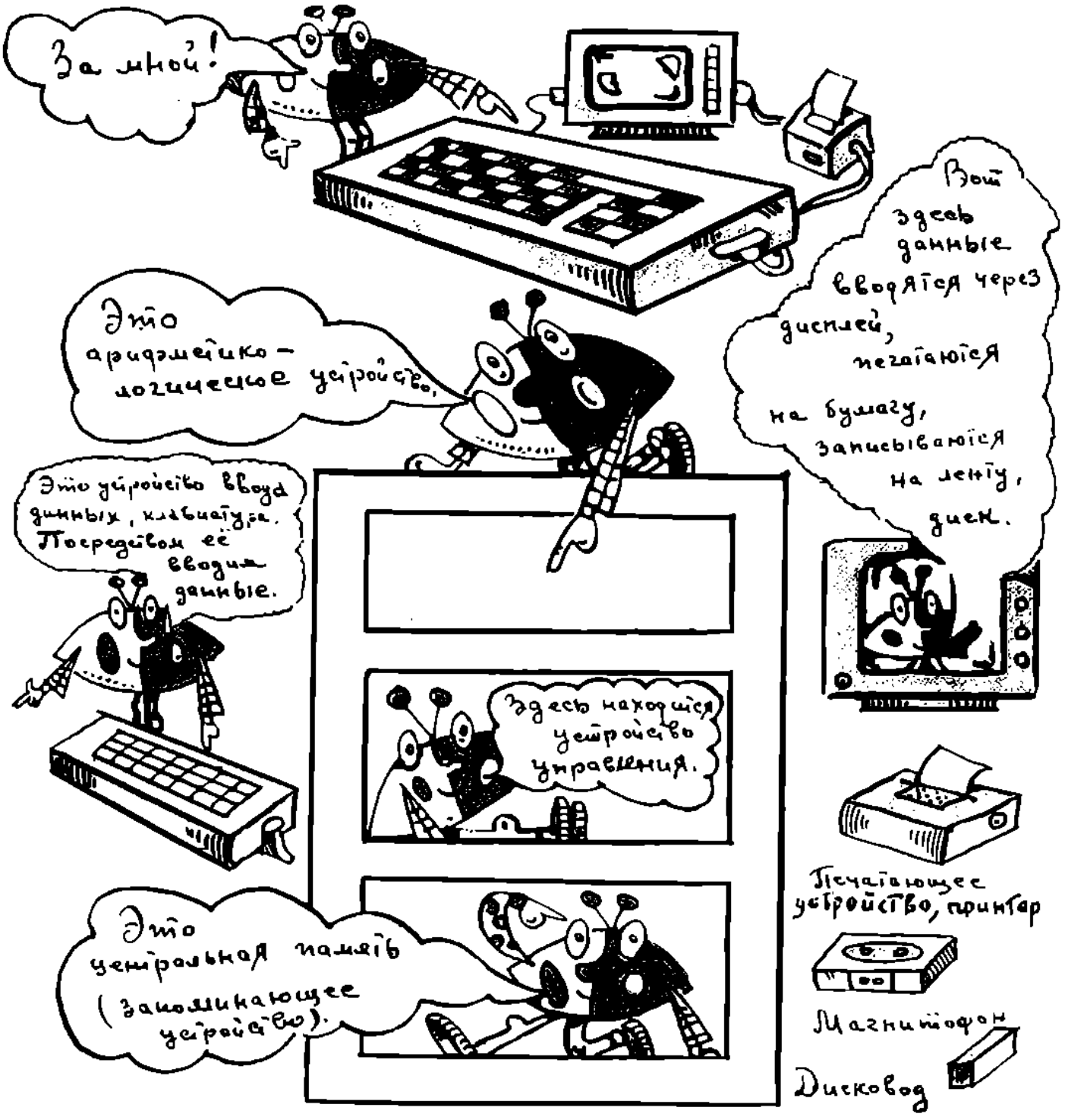
В этой части книги мы наряду с понятием *команда* используем новое понятие *оператор*. Часто оба термина употребляются как синонимы. Это связано с тем, что большинство команд можно использовать в качестве операторов программы и наоборот.

При изучении второй части книги мы узнаем больше об этих командах и ознакомимся с новыми командами бейсика, с помощью которых ты сможешь написать программы... невероятные!

ПОСМОТРИМ ВНУТРЬ

Теперь, когда ты умеешь сам писать программы, проанализируем, как и с помощью чего компьютер распознает и выполняет твои команды и операторы, то есть, как он работает внутри.

Ты уже знаешь, что компьютер запоминает все строки программы, которые ты запишешь. Войдем в компьютер и посмотрим, где он хранит даниые, которые ты вводишь, и что он делает с ними.



ПОСМОТРИМ ВНУТРЬ

Артуро потребовалось некоторое время для того, чтобы выучить названия, которые ты видишь на рисунке, но самое важное — не знать эти названия, а помнить, как работает каждая из этих частей компьютера.

Когда ты с помощью клавиатуры вводишь какие-либо данные в компьютер, то они поступают прямо в *центральную память*. Ты помнишь, что компьютеру нужно говорить почти все и что сначала ты должен объяснить ему, что он должен делать, чтобы после выполнения программы дать тебе желанный результат?

Но некоторые вещи компьютер уже знает. Обрати на это внимание. Ты ему сообщаешь операторы, команды с помощью пронумерованных строк, так, чтобы он их читал по порядку. Ты даже можешь давать операторы в неупорядоченном виде, и он сам их переставит в порядке возрастания номеров строк. Ты ведь делал такие программы, чтобы компьютер находил суммы, умножал и т. д. Но ты не обучал его ни суммировать, ни упорядочивать строки.

Внутри центральной памяти есть ПЗУ (постоянное запоминающее устройство). В ПЗУ компьютер запоминает и хранит все программы, которые необходимы для счета, упорядочения, ... Все эти программы ввел изготовитель в часть памяти компьютера, называемую ПЗУ (постоянной памятью), и они там хранятся всегда.

ПОСМОТРИМ ВНУТРЬ

Когда ты пишешь программу, в которой есть сложение, ты тем самым говоришь компьютеру: «Посмотри у себя в ПЗУ, как делается эта операция, и сделай ее».

Но внутри центральной памяти также имеется оперативное запоминающее устройство (ОЗУ). ОЗУ предназначено для запоминания всех команд, операторов и данных, которые ты вводишь в компьютер с помощью устройства ввода данных, клавиатуры. Поэтому ты можешь ввести данные и посмотреть (прочитать) их потом. Так как они хранятся в ОЗУ, то ты можешь потом к ним возвращаться и изменять их.

Когда ты выводишь программу, то ты как бы спрашиваешь компьютер: что ты имеешь в ОЗУ?

Внимание! Когда ты выключаешь компьютер, то исчезает все, что хранилось в ОЗУ, но всегда сохраняется содержимое ПЗУ, потому что в противном случае компьютер не знал бы, как с тобой работать, когда ты его попросишь сделать некоторые операции, и как понимать твои команды.

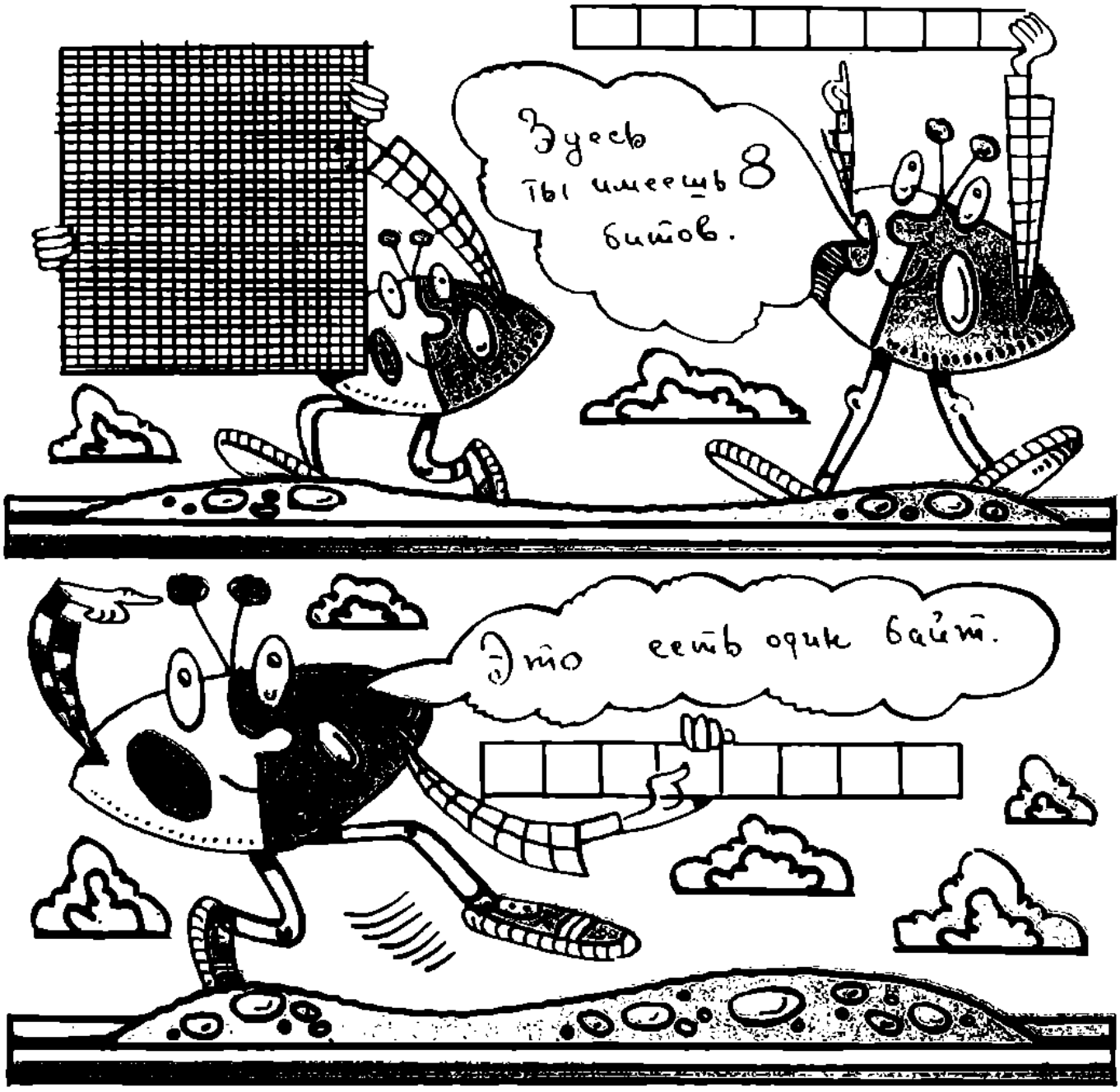
ПОСМОТРИМ ВНУТРЬ

Модели компьютеров имеют различные объемы памяти. Некоторые компьютеры обладают способностью сохранять в своей центральной памяти длиннейшие программы, или много программ, а другие хранят немного информации.

Для того чтобы узнать, много или мало памяти имеет компьютер, ввели величину памяти К. Наверно, ты слышал, как говорят: «Этот компьютер имеет 32 К памяти»?

Что такое К памяти? Смотри!

Центральная память внутри — как сетка из многих ячеек. Каждая ячейка — это один *бит*.



Каждую совокупность из 8 битов мы называем *байтом.*

В каждом байте запоминается один символ, то есть можно сказать, что компьютер использует один байт (или 8 битов) для каждого символа.

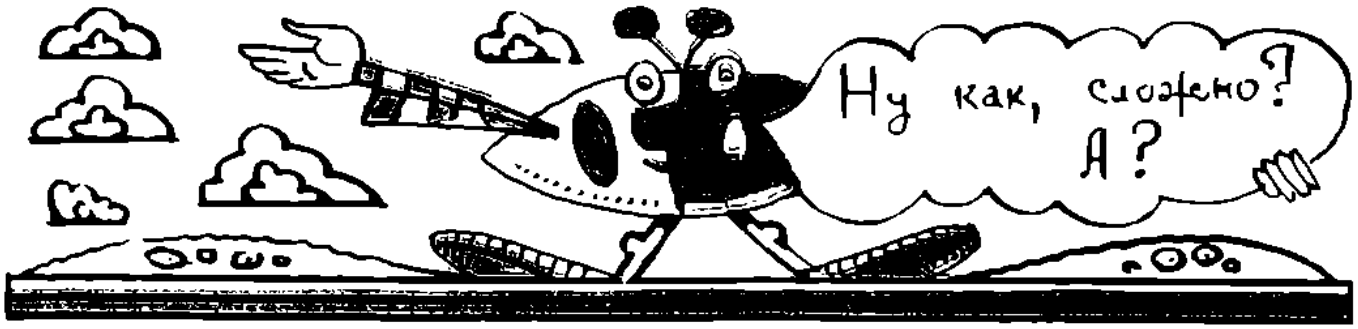
Таким образом, компьютер использует 8 ячеек (8 битов) для запоминания буквы А или В, для числа 2 или числа 7 и т. д.

1024 *байта* образуют один *Кбайт*, который мы обозначаем через К.

■ Продолжим анализировать, что же происходит внутри компьютера.

Даиные, операторы и команды поступают сначала в *устройство центральной памяти.* Оттуда передаются в *устройство управления. Устройство управления* предназначеио для распознавания и выполнения операторов и команд, которые ты даешь ему с помощью программы. *Арифметико-логическое устройство* выполняет вычисления, сравнения и т. п. Результаты этих вычислений или сравнений проходят еще раз через *устройство управления* в *центральную память*, где хранятся до тех пор, пока не будут выведены через *устройство вывода*

наружу. И тогда ты сможешь увидеть на экране результат выполнения своей программы.



Не волнуйся. Сейчас на следующем примере ты сразу все увидишь. Посмотри на программу:

```
10  LET A = 5
20  LET B = 3
30  LET C = A + B
40  PRINT C
50  STOP
```

А сейчас на рисунке на следующей странице посмотри, как эта программа интерпретируется и выполняется в различных частях компьютера.

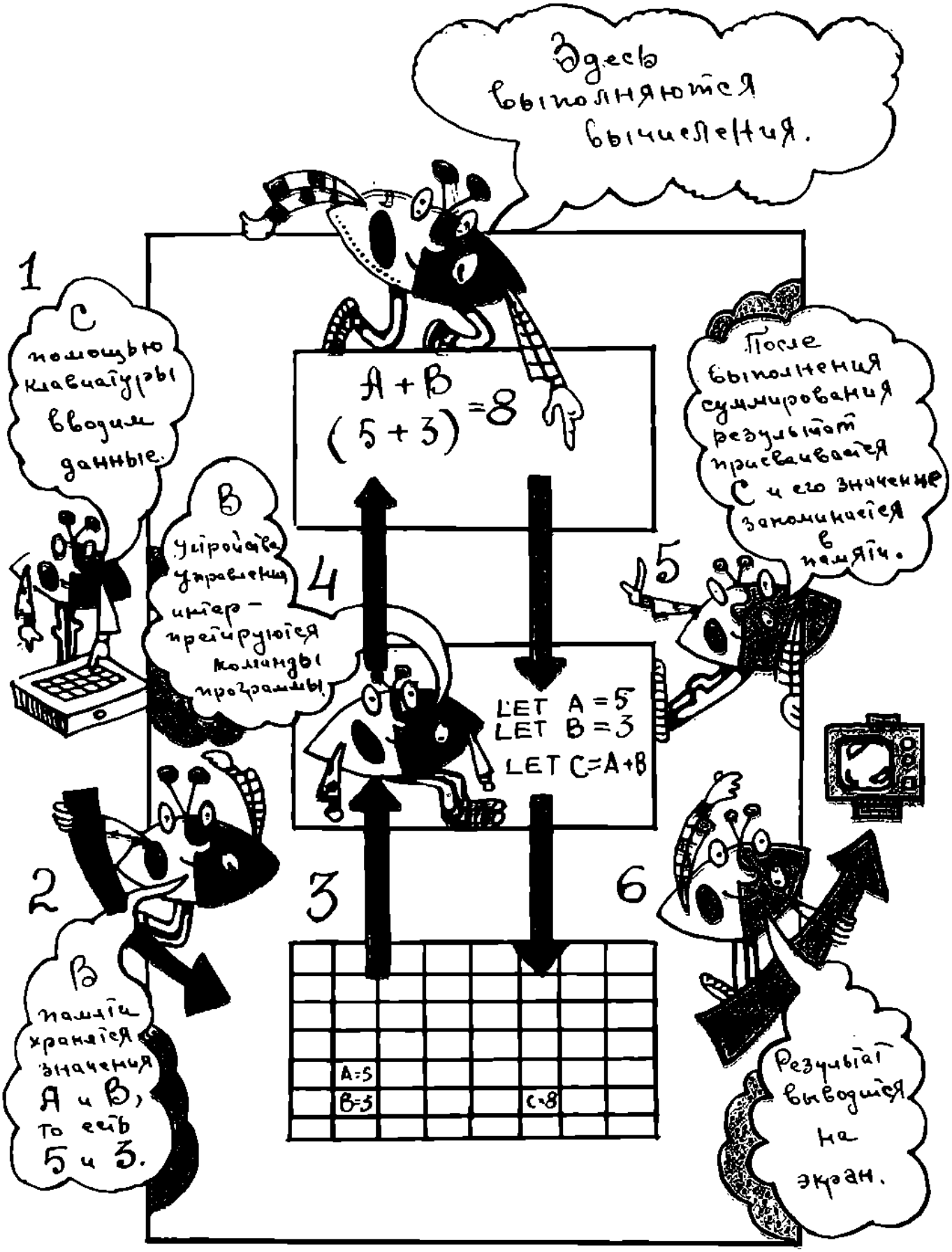
Не огорчайся, если ты не все хорошо понимаешь. На самом деле важно, чтобы у тебя сейчас было общее представление, как работает компьютер, и чтобы, когда ты услышишь какой-нибудь из этих терминов, ты знал приблизительно, о чем идет речь.

Все, что ты можешь увидеть внутри и снаружи компьютера (клавиатура, дисплей, принтер, кабели, интегральные схемы и т. д.), называется *хардвером* (от англ. HARDWARE) — физическим оборудованием (техиической составляющей компьютерной системы).

А программы, различные сервисные программы, программы-трансляторы и т. д.— все это называется *софтвером* (от англ. SOFTWARE) — программным обеспечением компьютерной системы.

ПОСМОТРИМ ВНУТРЬ

Понятия хардвер (жесткая, твердая часть) и софтвер (мягкая часть) служат для обозначения аппаратиой и программной частей компьютера.



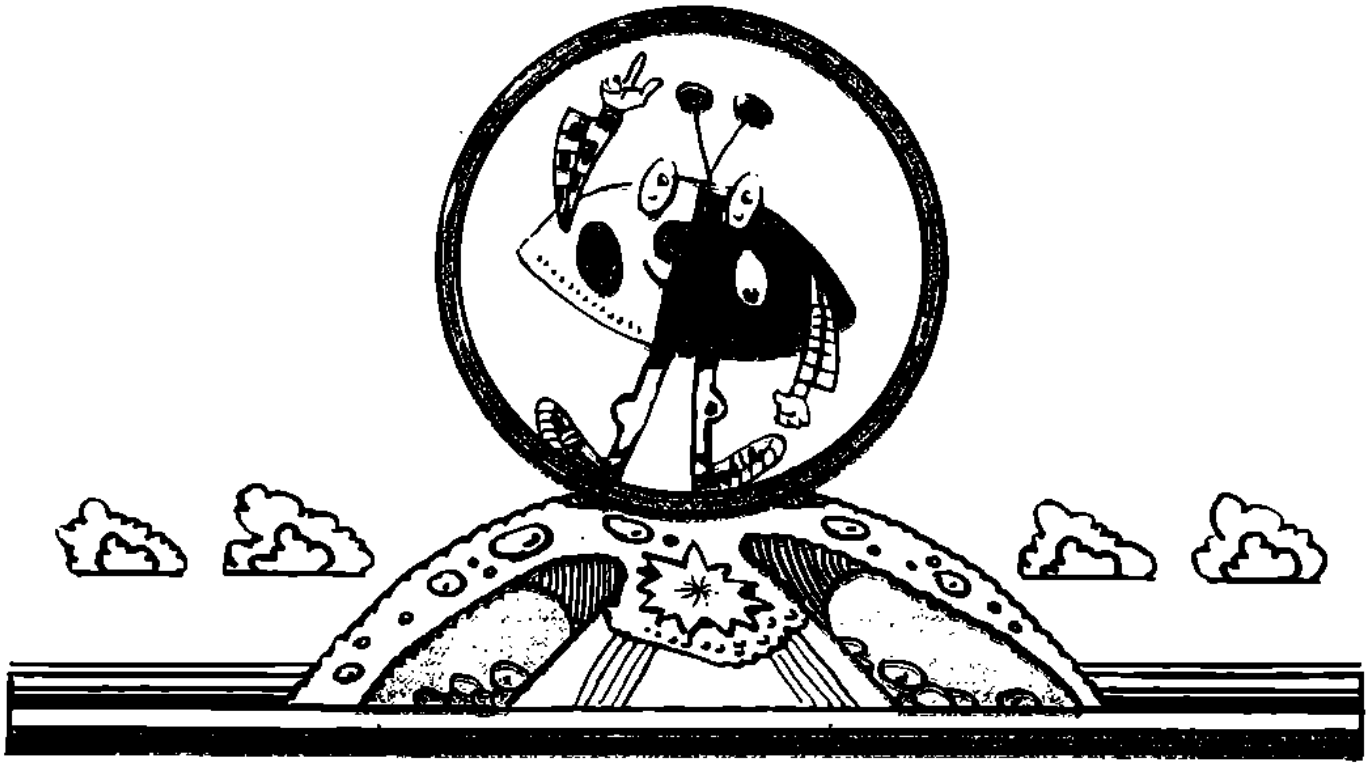
ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Не так важно, чтобы ребенок усвоил все понятия предыдущего пункта. Достаточно, чтобы он ознакомился с ними.

- Проделайте вместе с ребенком путь, указанный стрелками на рисунке (с. 105). Нарисуйте на бумаге или доске схему компьютера и попросите ребенка последовательно нарисовать стрелки, соблюдая порядок процесса.

- Компьютер работает исключительно в двоичной системе, то есть разрешаются комбинации из двух элементов (0 и 1). Так, 1 байт (или 8 битов) позволяет создать 256 различных комбинаций. Действительно, $2^8 = 256$. Например, число 30 представляется как 00011110. Число 211 представляется как 11010011.

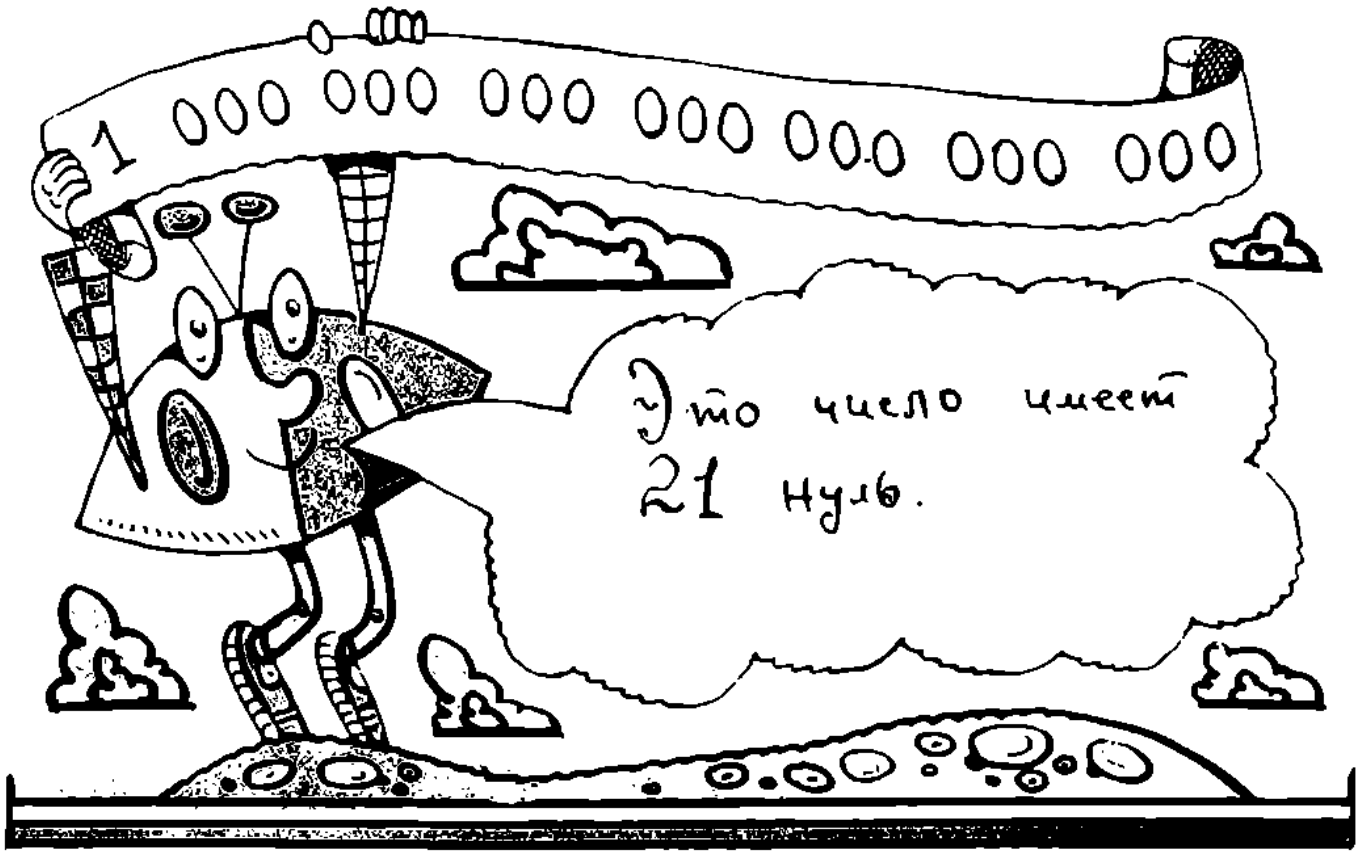
- Существуют микропроцессоры, которые способны обрабатывать 16 и даже 32 бита одновременно, что дает им большие дополнительные возможности.



ПЯТАЯ ОПЕРАЦИЯ

Артуро задумался, когда впервые услышал о пятой операции над числами. Он считал, что основных операций всего четыре: сложение, вычитание, умиожение и деление. Но, оказывается, есть еще и пятая операция, очень интересная, позволяющая легко производить сложные вычисления.

Посмотри лучше несколько простых примеров. Представь себе такое число:



Это эквивалентно числу километров, отделяющих нас от некоторых звезд. Когда астроному необходимо выполнить расчеты для отправки новых космических кораблей, то он должен оперировать многократно такими числами. Это очень тяжело и можно сделать много ошибок. Совершенно невозможно правильно работать со столькими нулями.

Более простой способ представить это число в таком виде:

$$10^{21}$$

Оно читается как «десять в 21-й степени», то есть число 10 умножается само на себя 21 раз. Число, которое нам показывает, сколько раз нужно какое-то число умножить само на себя (в нашем случае это 21), называется степенью. Поэтому операция и называется *возведение в степень*.

Другой пример. Число 8 тоже может быть записано как 2^3. Это значит, что нужно умножить 2 три раза само на себя:

$$2 * 2 * 2 = 8$$

■ Теперь мы тебе расскажем интересную историю.

Как повествует одна индийская легенда, жил-был король, который умирал от скуки. Однажды он велел созвать всех ученых королевства, чтобы они придумали игру, которая развлекла бы его. Многие пытались, но безуспешно.

И вот появился странствующий человек, который предложил королю игру, настолько чудесную и интересную, что король очень обрадовался. Это была игра, которую мы называем теперь шахматами.

Избавившийся от скуки король решил наградить изобретателя шахмат. Он сказал ему: «В благодарность за твою мудрость можешь просить, чего пожелаешь». Странник ответил: «Ваше величество, я прошу только 2 зерна пшеницы за первую клетку доски, 4 зерна за вторую клетку, 8 зерен за третью, 16 зерен за четвертую и так далее — до заполнения всех 64 клеток шахматной доски».

Король остался очень доволен, так как решил, что изобретатель шахмат попросил очень мало за такую интересную игру, позвал своего первого министра и приказал ему, чтобы он дал изобретателю все, что тот попросит.

Но каково было удивление короля, когда министр доложил ему, что даже если бы все люди на Земле занимались только выращиванием пшеницы, то во всем мире не хватило бы зерен, чтобы собрать то количество, которое просит изобретатель шахмат.

Сообщаем: нужно было бы примерно

18 446 744 500 000 000 000 зерен пшеницы.

Это число такое большое, что трудно его представить. Если бы на всей Земле люди выращивали пшеницу, то должно было бы пройти 45 000 лет, чтобы собрать такое

количество пшеницы. Чтобы посчитать все зерна пшеницы, потребовалось бы по крайней мере 1170 миллионов веков или 117 000 миллионов лет. Это невероятно!

Ты уже, наверное, понял, что за каждую новую клетку изобретатель просил вдвое больше, чем за предыдущую, то есть умножал на два число зерен предыдущей клетки, и так на всех 64 клетках.

Это огромное число зерен можно было бы написать как 2^{64}. Вроде фокус, да?

Для компьютера число 2^{64} записывается как 2↑64, так как в компьютере знак ↑ означает возведение в степень.

■ Давайте напишем программу по этой легенде.

```
 5  REM  "ШАХМАТЫ"
10  PRINT "НАЖМИ S ДЛЯ ДРУГОЙ КЛЕТКИ"
20  PRINT
30  LET C=0
40  LET A=1
50  LET A=A*2
60  LET C=C+1
70  PRINT "КЛЕТКА", "ЗЕРЕН"
80  PRINT C, A
90  INPUT S$
100 IF S$="S" THEN GOTO 50
```

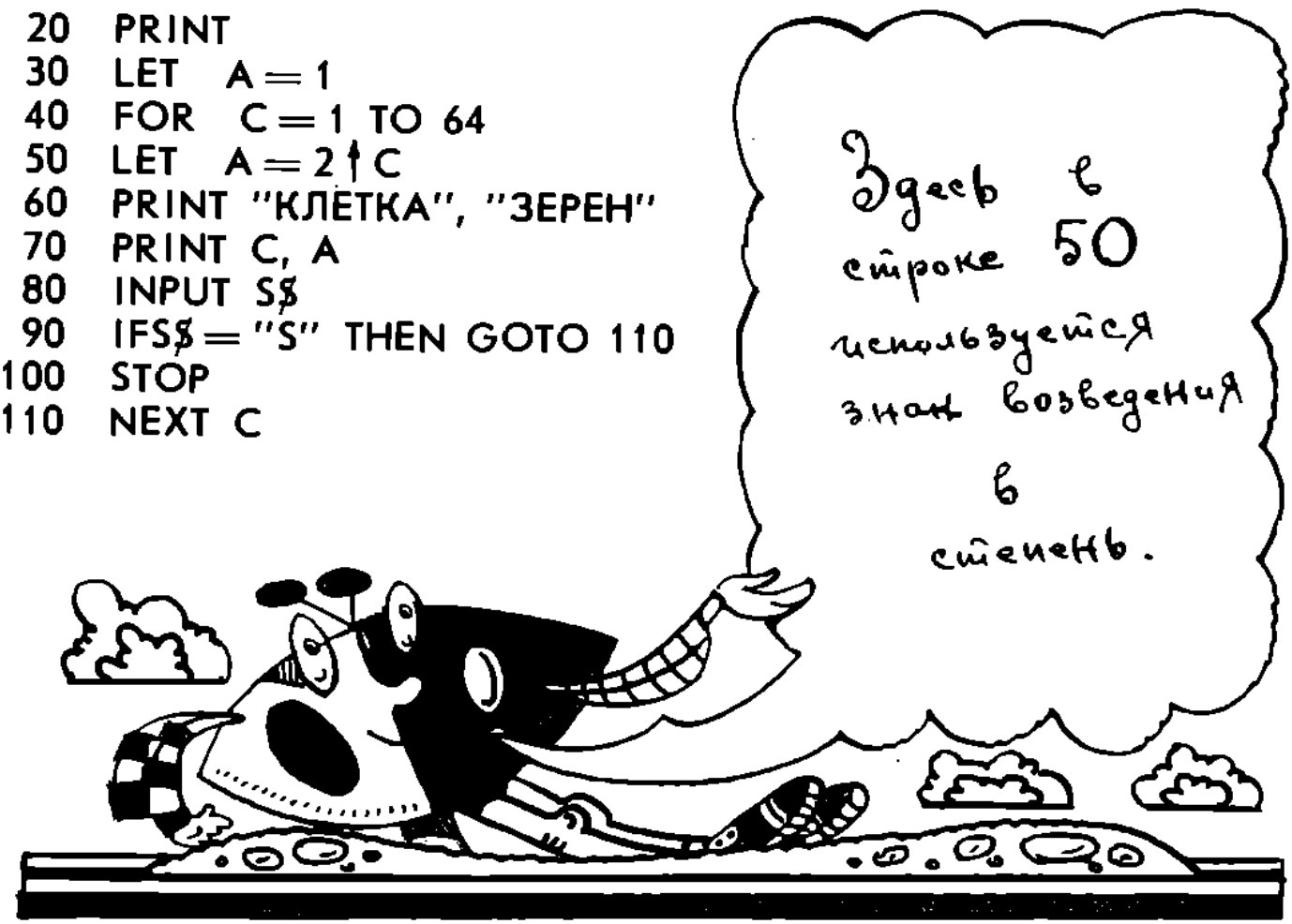
Нет необходимости объяснять, как работает эта программа, так как все операторы очень простые, и ты уже их изучал в первой части книги.

Мы можем также написать эту программу, используя цикл FOR/NEXT.

```
 5  REM "ШАХМАТЫ"
10  PRINT "НАЖМИ S ДЛЯ ДРУГОЙ КЛЕТКИ"
```

ПЯТАЯ ОПЕРАЦИЯ

```
 20  PRINT
 30  LET  A = 1
 40  FOR  C = 1 TO 64
 50  LET  A = 2 ↑ C
 60  PRINT "КЛЕТКА", "ЗЕРЕН"
 70  PRINT C, A
 80  INPUT S$
 90  IF S$ = "S" THEN GOTO 110
100  STOP
110  NEXT C
```



Выполни программу. Ты увидишь, как количество зерен с каждым разом будет больше, и когда ты достигнешь примерно 30-й клетки, компьютер выведет какие-то странные цифры.

Для клетки 30, например, ты получишь на экране:

1.07374182E + 9

Это означает, что компьютер тоже написал это число *на своем языке,* который он использует для обозначения очень больших чисел.

ПЯТАЯ ОПЕРАЦИЯ

Символ E+ 9 — это одно и то же, что сказать: необходимо сдвинуть запятую в числе на 9 знаков вправо. Или сказать, что нужно 9 раз умножить число 1.07374182 на 10 для получения числа зерен, которые имеются на 30-й клетке шахматной доски.

Клетка	Математическое выражение	Выражение в информатике	Число
30	2^{30}	2↑30 (как ты напишешь) 1.07374182E+9 (как тебе даст компьютер)	1073741820

Когда дойдешь до конца программы, увидишь, что компьютер тебе даст (используя свою форму представления) общее число зерен на 64 клетках доски.

Этот же результат, но без просмотра клетки за клеткой, ты бы мог получить, введя соответственно:

```
PRINT 2↑64
```

Как видишь, возведение в степень очень удобно, когда имеешь дело с большими числами.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Вполне вероятно, что ребенка не заинтересует эта тема. Поэтому нужно постараться интересно объяснить этот материал, используя легенду о шахматах. Возможно, что некоторые дети не смогут понять термин «числовая величина», который был рассмотрен в примере. Это не имеет значения, как не имеет значения и то, что они не поймут сейчас до конца понятие степени.

ПЯТАЯ ОПЕРАЦИЯ

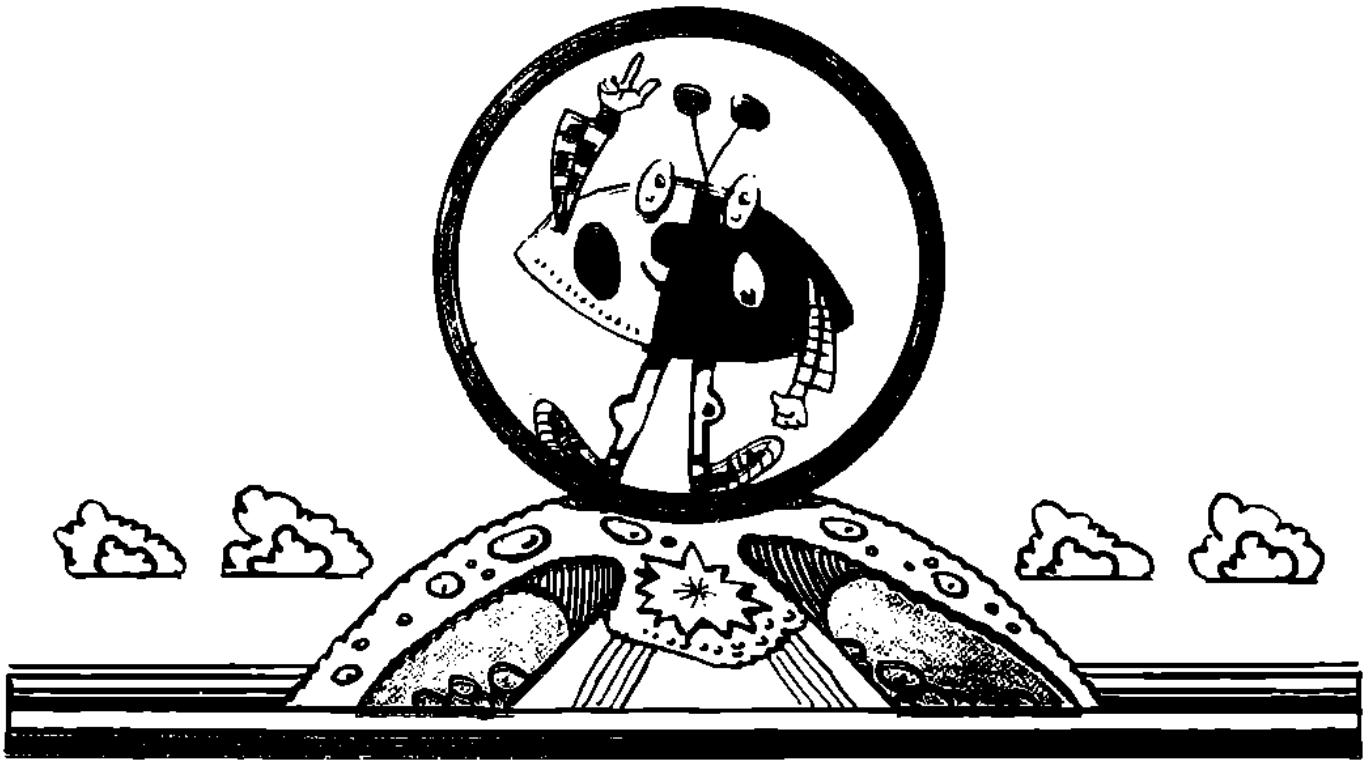
- Мы включили эту операцию в пункт для того, чтобы ребенок ознакомился с ней и чтобы для него не было неожиданностью действие возведения в степень, когда он будет читать следующий пункт, посвященный арифметическим операциям.

- Незнанне определения степени абсолютно не помешает понять темы, изучаемые в этой части книги.

- Объясните ребенку, что выражение на экране, которое может использовать компьютер для изображения возведения в степень, есть E+..., а для ввода с клавиатуры ребенок может использовать символ ↑.

- В программе нет комментариев, так как все использованные операторы хорошо известны ребенку. Если вы заметили, что ребенок с трудом понимает, прочитайте ему еще раз те команды, которые ему должны быть хорошо знакомы.

- Если даже при использовании экспоненциального представления (с E) число оказывается очень большим для компьютера, то он выдаст соответствующее сообщение.



АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

Ты уже знаешь, что машина способна выполнить определенную операцию, если ты укажешь ее с помощью оператора программы. При написании программы очень внимательно продумывай последовательность команд, потому что компьютер прочитает их в этой же последовательности. Так же внимателен будь к программе, если хочешь, чтобы компьютер выполнил арифметическую операцию. Нельзя записывать действия сложения, вычитания, умножения, ... вперемежку, не учитывая порядка выполнения операций. Компьютер очень аккуратен и всегда выполняет действия в определенной последовательности.

■ Посмотрим, как бы он вычислял следующее выражение:

```
PRINT 3—2+8—4
```

Компьютер суммирует и вычитает числа одно за другим в той последовательности, какую ты указал в программе.

Внесем небольшое изменение:

```
PRINT 3—2+8/4
```

Для компьютера деление имеет больший приоритет, чем сложение и вычитание, поэтому он сначала будет делить числа, находящиеся справа и слева от косой черты, а затем выполнит действия сложения и вычитания.

■ Как ты уже заметил, существуют определенные правила, по которым работает компьютер. Допустим, мы имеем математическую задачу, в которой есть все арифметические действия (сложение, вычитание, деление, умножение и возведение в степень).

Компьютер:

во-первых, выполнит все операции возведения в степень слева направо;

во-вторых, выполнит все действия умножения и деления слева направо;

в-третьих, выполнит действия сложения и вычитания слева направо.

Но когда компьютер встретит действие, записанное внутри скобок, то сначала он выполнит все операции в скобках (в том порядке, который он знает), а затем будет выполнять остальные операции согласно правилам, о которых мы уже сказали.

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

■ Предположим, что тебе не хочется выполнять действия в такой последовательности: делить 8 на 4, прибавлять к результату 3, а затем вычитать 2, а ты хочешь сначала из 3 вычесть 2, добавить 8, а затем все это разделить на 4.

Ты можешь указать компьютеру, в какой последовательности выполнять действия, написав:

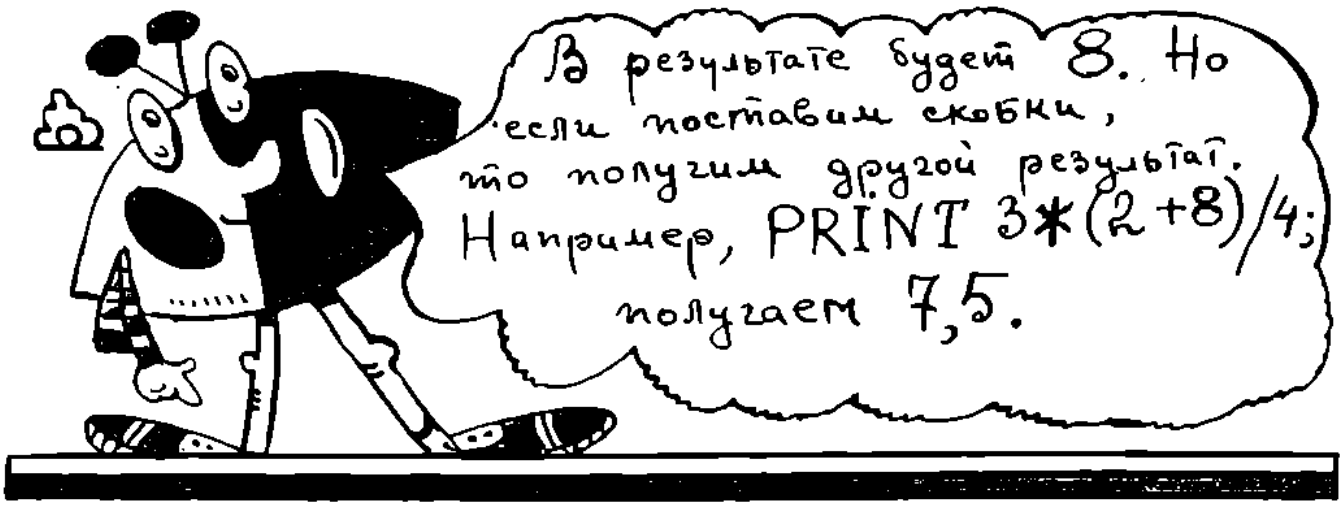
```
PRINT (3—2+8)/4
```

Компьютер сделает прежде всего то, что заключено в скобки ( ), поэтому результат теперь будет — 2.25.

■ Внесем еще одно изменение:

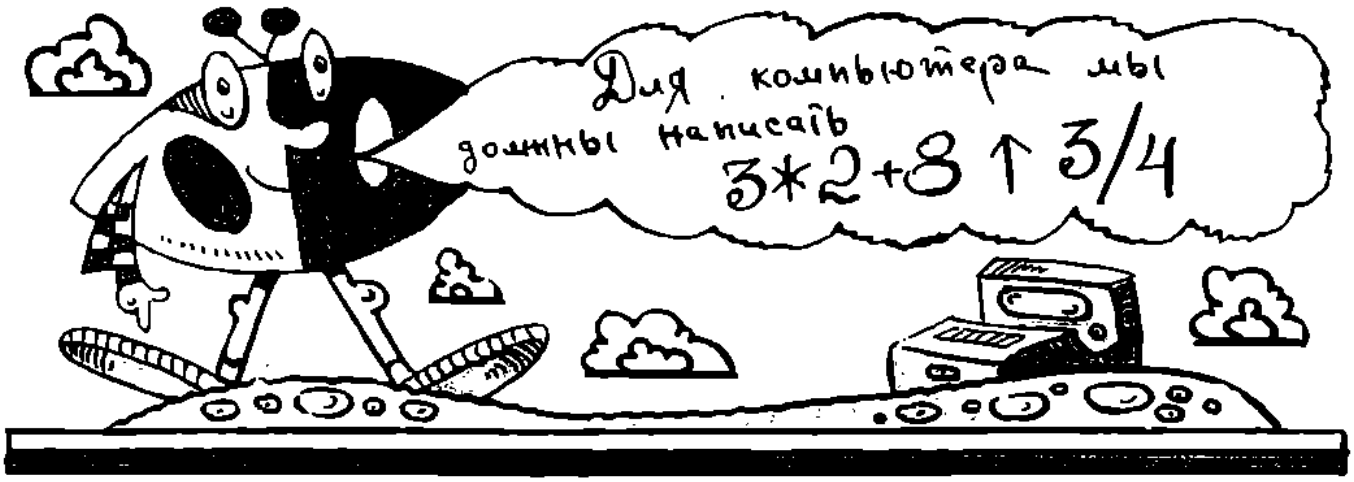
```
PRINT 3*2+8/4
```

Компьютер умножит 3 на 2, затем разделит 8 на 4. Результат умножения сложит с результатом деления. В этом случае умножение выполняется раньше, чем деление, так как обычно действия выполняются слева направо.



■ Выполним новое вычисление:

```
PRINT 3*2+8^3/4
```



Для компьютера приоритетнее возведение в степень, чем сложение, умножение или деление, поэтому сначала он вычислит 8^3 (что равно 512). Затем по порядку слева направо будет действовать так:

$3*2+512/4$ Сначала умножит 3 на 2, затем поделит 512 на 4.

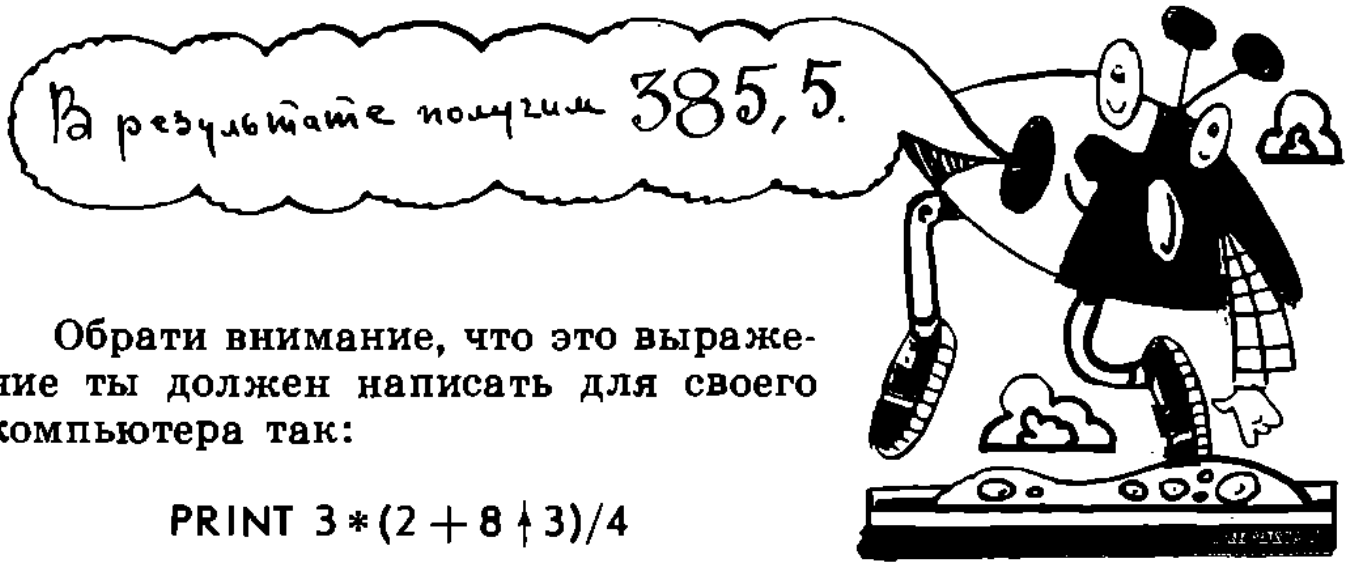
$6 + 128$ В конце сложит результаты, полученные из предыдущих действий.

В конечном результате получим 134.

■ Рассмотрим другой пример со скобками:

PRINT $3*(2+8^3)/4$

Сначала возведем 8 в третью степень.
К результату прибавим 2.
Полученный результат умножим на 3.
Результат разделим на 4.



Обрати внимание, что это выражение ты должен написать для своего компьютера так:

PRINT 3*(2+8↑3)/4

Проследи шаг за шагом порядок действий и ты увидишь, что компьютер сначала выполнит вычисления внутри скобок, а затем продолжит действия согласно правилам, которые мы уже знаем.

■ Что произойдет, если мы немного изменим расположение скобок?

Напишем так:

PRINT $3*(2+8)^3/4$

Сначала сложим 2 и 8.
Результат возведем в третью степень
Результат умножим на 3.
Результат разделим на 4.

Как ты уже заметил, в зависимости от чисел и действий, заключенных в скобки, результат получается различный. Важно представить порядок действий, который ты хочешь сделать, прежде чем написать их в форме, понятной для компьютера. В противном случае компьютер даст неверный результат.

■ Рассмотрим один пример. У Артуро было 15 конфет, 3 он съел. Оставшиеся конфеты ему нужно разделить между шестью друзьями.

Если для компьютера напишем:

PRINT 15—3/6,

то компьютер согласно известным тебе правилам даст результат 14.5. Как видишь, этого не может быть. Артуро должен написать:

PRINT (15—3)/6

Составим программу, используя вместо цифр переменные:

```
10  PRINT "СКОЛЬКО КОНФЕТ У АРТУРО?"
20  INPUT A
30  PRINT "СКОЛЬКО ОН СЪЕЛ?"
40  INPUT B
50  PRINT "СКОЛЬКО У АРТУРО ДРУЗЕЙ?"
60  INPUT C
70  LET X = (A—B)/C
80  PRINT  "АРТУРО ДАСТ " ;X;" КОНФЕТ КАЖДОМУ ИЗ
    СВОИХ "; C; " ДРУЗЕЙ"
```

Обрати внимание на важность правильного написания строки 70. Измени ее на:

```
70  LET X = A—B/C
```

и ты увидишь, что компьютер выведет неверный результат.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Иерархия арифметических операций на деле очень проста и легка для понимания. Может быть, ее труднее объяснить, чем понять.

- Ребенок не должен бояться ошибок в написании формул. Все очень просто, и нужна практика для того, чтобы освоить тему.
  Выполняйте вместе различные действия и вычисления с постепенным их усложнением.

- Убедите ребенка в важностн ясного представления прноритетности расчетов, которые он должен реализовать для того, чтобы написать свое арифметическое выражение в форме, пригодной для вычислений на компьютере. Лучше всего, чтобы он это делал сначала на бумаге.

- Рассмотрите с ребенком преимущества написания программ в той форме, как она дана в этом пункте, вместо того чтобы сразу вывести PRINT (15—3)/6, хотя оба варианта программы дадут нам одинаковый результат. Использование программ с переменными и ввод их значений с помощью оператора INPUT позволяет нспользовать их для разных случаев.

- Если сочтете необходимым, то можно использовать такие выражения: $(3 * (2 + 8)\dagger 3)/4$, где появились внутренние и внешние скобки. Компьютер в этом случае всегда сначала выполннт действия во внутренних скобках.

CLEAR и CLS

Ты уже знаешь, что прежде чем начать новую программу ты должен стереть все данные, которые компьютер имеет в своей памяти.

Обычно ты используешь команду NEW. С ее помощью компьютер подготавливает себя для записи новой программы. Командой NEW мы стираем в памяти компьютера все имеющиеся у него данные и программы.

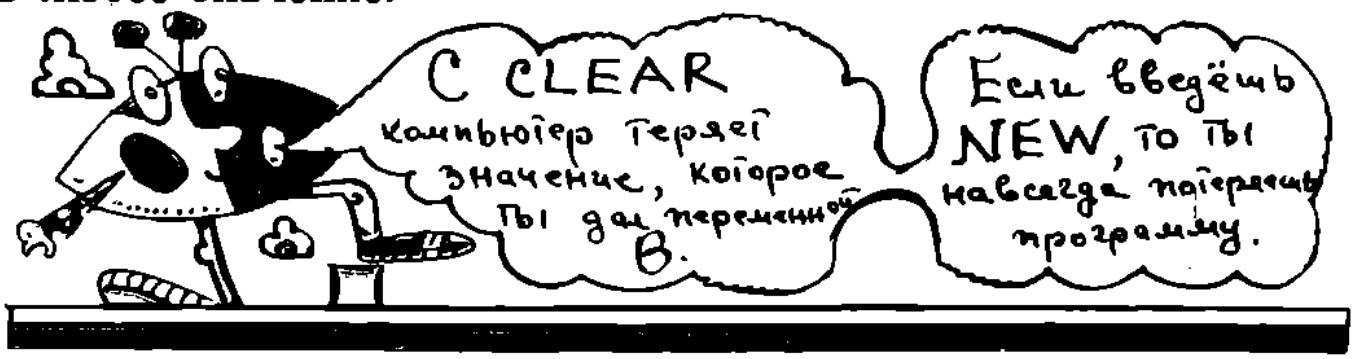
Теперь испробуем некоторые новые команды. Введи следующую программу:

```
10  LET A = 5
20  PRINT "ВВЕДИ ЛЮБОЕ ЧИСЛО"
30  INPUT B
40  LET M = A*B
50  PRINT "РЕЗУЛЬТАТ РАВЕН "; M
```

Сейчас на экране ты имеешь эту небольшую программу. Введи команду CLEAR и посмотри, что произойдет. Программа исчезла с экрана.

Теперь набери LIST. Программа снова появится на экране. Это происходит потому, что команда CLEAR не стирает программу в памяти.

Запусти программу и дай B, например, значение 6. На экране ты можешь увидеть результат. Введи CLEAR, а затем просмотри программу (с помощью LIST). Проанализируй ее. Она такая же, как сначала. A имеет значение 5, но B не равно 6, зато если ты запустишь программу еще раз, то можешь дать B любое значение.



Другим оператором для стирания с экрана без потери программы является CLS. Но заметь, что NEW и CLEAR — команды, то есть, другими словами, они выполняются немедленно

без программы, а CLS должен быть включен как строка программы. Если во время выполнения программы компьютер доходит до строки с оператором CLS, он стирает все, что в этот момент находится на экране. Добавь в программу строку:

```
45 CLS
```

Выполни программу и сравни результат.

Поэтому необходимо, чтобы в начале каждой программы в первой строке был оператор CLS. Благодаря этому ты будешь иметь автоматическую очистку экрана перед выполнением программы. Добавь в нашу программу: 5 CLS

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Попрактикуйтесь с ребенком, пока он полностью не поймет эти понятия и команды.

- В нашей программе B есть некоторая переменная, а A — константа, и ее значение мы определили в 10-й строке программы.

- Имейте в виду, что:
  команда RUN присваивает всем переменным значение 0 и стирает или не стирает содержимое экрана в зависимости от типа компьютера;
  команда NEW стирает программу из памяти (с ОЗУ);
  команда CLEAR стирает содержимое экрана, а не памяти, но значения переменных при этом становятся равными нулю;
  оператор CLS стирает содержимое экрана, но не изменяет значения переменных и не стирает программ из памяти.

- Необходимо, чтобы, начиная с этого момента, ребенок привыкал систематически использовать оператор CLS.

INPUT (немного подробнее)

Ты, конечно, уже знаешь, как использовать оператор INPUT. До сих пор мы использовали комбинацию PRINT и INPUT, чтобы иметь возможность «разговаривать» с компьютером. Сейчас мы рассмотрим, как с помощью оператора INPUT можно экономить количество строк в программе. Введем снова программу со страницы 119. Теперь изменим строку 20 и запишем:

```
20  INPUT "ВВЕДИ ЛЮБОЕ ЧИСЛО"; B
```

Сотри строку 30. При выполнении программы на экране появится то же сообщение, что и раньше. Ты поменял только одну строку, и она выполнила то, что раньше выполняли два оператора: PRINT и INPUT.

Полная программа выглядит так:

```
 5  CLS
10  LET A = 5
20  INPUT "ВВЕДИ ЛЮБОЕ ЧИСЛО"; B
30  LET M = A*B
40  PRINT "РЕЗУЛЬТАТ РАВЕН "; M
```

■ Сделаем еще одну программу, чтобы попрактиковаться в использовании этой формы INPUT. Поможем Артуро заполнить анкету:

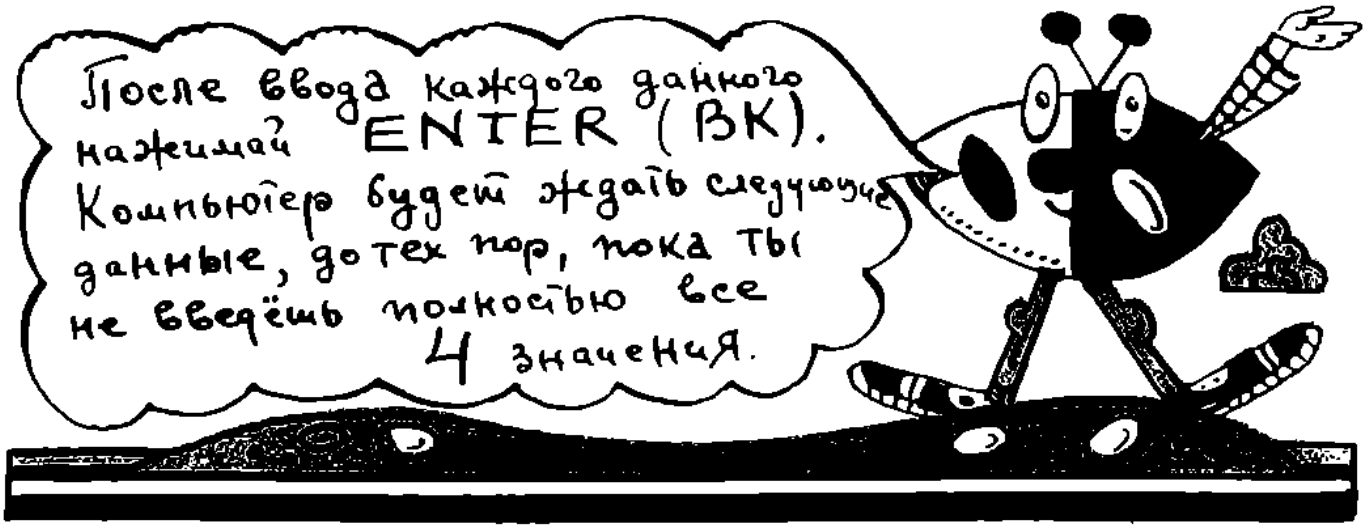
```
 5  CLS
10  INPUT "ИМЯ"; N$
20  INPUT "ОТЧЕСТВО"; A$
25  INPUT "ФАМИЛИЯ"; C$
30  INPUT "АДРЕС"; D$
40  CLS
50  PRINT N$; " "; A$;" "; C$
60  PRINT "ЖИВЕТ ПО АДРЕСУ"; " "; D$
70  END
```

Но мы можем сделать более короткую программу, поскольку INPUT позволяет нам ввести больше одной переменной в той же строке программы. Предыдущую программу мы можем написать так:

```
 5  CLS
10  INPUT "ВВЕДИ СВОЕ ИМЯ, ОТЧЕСТВО, ФАМИЛИЮ И
    АДРЕС"; N$, A$, C$, D$
```


```
15  CLS
20  PRINT "АНКЕТНЫЕ ДАННЫЕ"
30  PRINT
40  PRINT N$:"  "; A$; " "; C$; " ЖИВЕТ ПО АДРЕСУ "; D$
```

INPUT (немного подробнее)



Как видишь, INPUT — очень удобный оператор. Обрати внимание на интересную вещь в строке 40. Между N$ и A$ мы оставили один пробел для того, чтобы тексты отделялись друг от друга (переменные выводились отдельно).

Если необходимо отделить два текста друг от друга или цифровую переменную от текста в кавычках, всегда оставляй один пробел внутри кавычек. Так мы уже делали прежде.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Попрактикуйтесь с ребенком в использовании этой формы оператора INPUT и постарайтесь, чтобы он понял, что в одной и той же строке можно делать так, чтобы компьютер ожидал ввода и цифровых, и символьиых переменных. Переменные должны быть разделены запятыми.

- Точно так же попрактикуйтесь в использовании пробелов внутри оператора PRINT, добиваясь, чтобы тексты были отделены друг от друга. Не все компьютеры действуют одинаково; так, некоторые из них оставляют пробелы с каждой стороны цифровых данных.

FOR/NEXT (немного подробнее)

Давай поработаем еще немного с этим оператором, который ты уже знаешь. Использовать FOR/NEXT в программе так полезно и так удобно, что лучше изучить его более глубоко и подольше попрактиковаться в его использовании.

Теперь мы будем использовать этот оператор очень часто. Сначала повторим немного, как он работает, и рассмотрим некоторые интересные особенности его наилучшего применения.

Введи программу:

```
10  FOR  X=1 TO 3
20  PRINT X
30  NEXT X
```

Эта программа так проста, что нет необходимости тебе ее объяснять.

Ты можешь также написать ее следующим образом:

```
10  FOR X=1 TO 3:PRINT X:NEXT X
```

Может быть, ты еще не знаешь, что внутри одной строки программы можно записать несколько операторов. Единственное, что нужно при этом делать — это отделять один оператор от другого двоеточием (:). Это справедливо для любых операторов.

■ Есть более сложная форма оператора цикла FOR/NEXT:

```
10  FOR X=1 TO 10 STEP 2
```

Это означает, что X также принимает значения от 1 до 10, но уже с шагом два. Если написать STEP 3, то шаг будет равен трем и т. д.

■ Как и другие операторы в бейсике, FOR/NEXT можно использовать в комбинации с другими операторами для облегчения программирования. Например, запишем:

FOR/NEXT (немного подробнее)

```
10  INPUT A
20  INPUT B
30  INPUT C
40  FOR X = A TO
    B STEP C
50  PRINT X
60  NEXT X
```

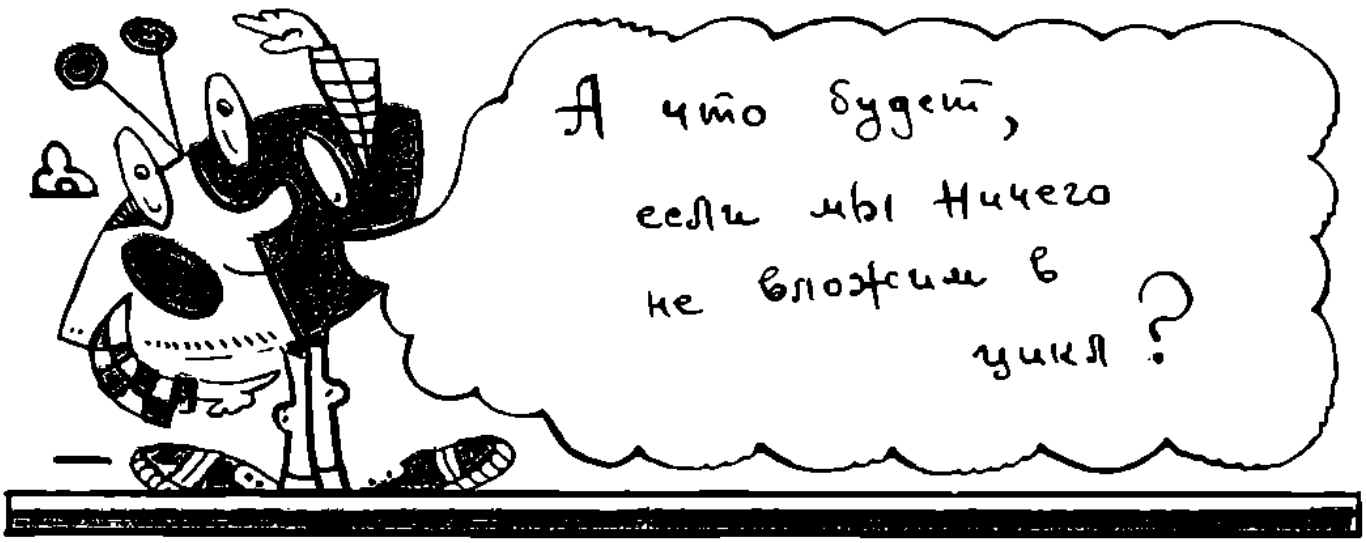
Запусти программу на выполнение и дай переменным A, B и C значения, какие хочешь, например 0, 20 и 3. Ты получишь список чисел, иачинающихся от 0, через три единицы. Обрати внимание, что последнее значение равно 18. Согласно инструкции STEP следующее значение должно быть 21, но ты сообщил компьютеру, что последнее значение X должно быть 20, поэтому после 18 не будет выведено никакого числа.

■ Еще одно ты должен иметь в виду: в случае необходимости, ты можешь выйти из цикла FOR/NEXT. Добавь в предыдущую программу:

```
 45  IF X = 9 THEN GOTO 100
100  PRINT "Я ВЫШЕЛ ИЗ ЦИКЛА, ПОТОМУ ЧТО X ПРИНЯЛ
     ЗНАЧЕНИЕ 9"
```

Значит, с операторами типа GOTO или IF... THEN ты можешь выйти из цикла, и цикл прервется. Единственное, что ты не должен делать,— это входить в цикл не через его начало, так как в этом случае компьютер зафиксирует ошибку.

■ Рассмотрим еще некоторые формы использования FOR/NEXT. В основном, внутри цикла ты ставишь серию операторов, повторяющихся определенное число раз.



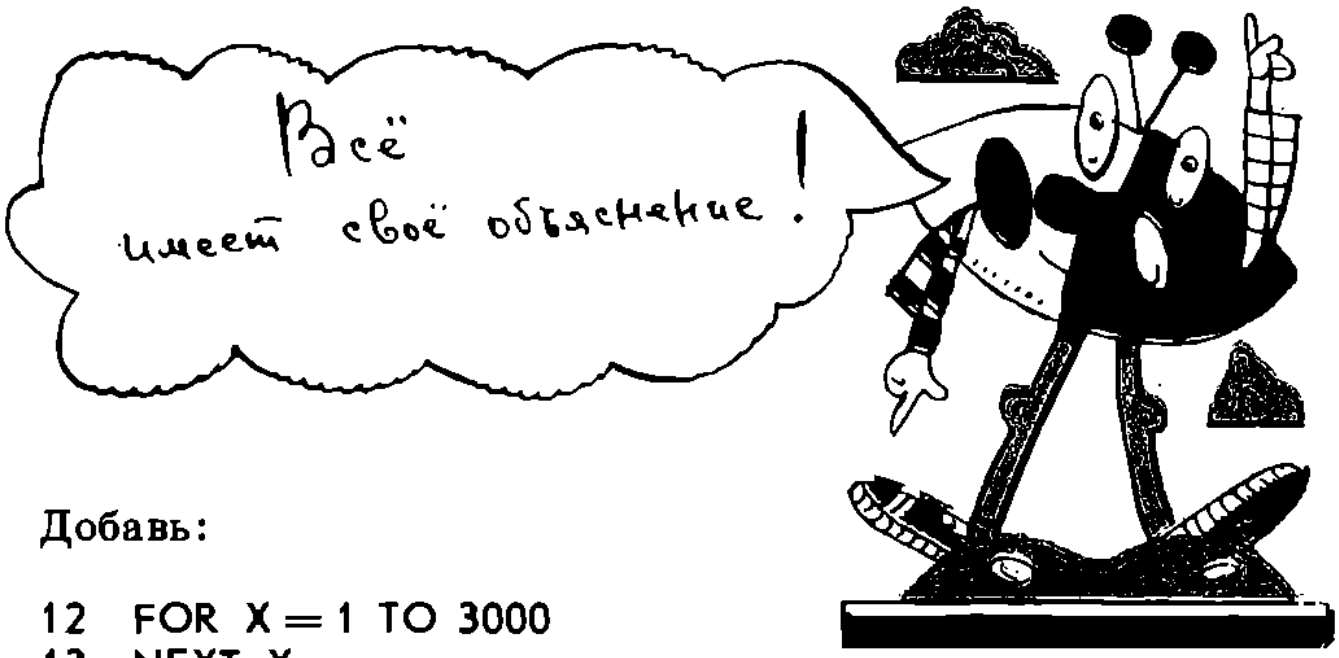
Но... ничего и не произойдет, то есть получим некоторый пустой цикл. Артуро не очеиь хорошо понял, зачем иам нужен пустой цикл, который иичего не делает. Тем ие менее он имеет свое назначение. Даже если цикл пустой, он все равио выполняется компьютером, и в это время не выполняются другие работы. Поэтому мы можем использовать этот вид цикла, чтобы замедлить действия программы.

Например, введем:

```
 5 CLS
10 PRINT "ЭТА СТРОКА ИСЧЕЗНЕТ ЧЕРЕЗ НЕКОТОРОЕ
   ВРЕМЯ"
20 CLS
30 PRINT "ВСЕ СТРОКИ ИСЧЕЗЛИ"
```

При выполнении программы ты не успеешь прочитать на экране первую фразу, так как после ее вывода компьютер прочитает строку 20 и сотрет ее. Поэтому удобно использовать в этой программе один пустой цикл, чтобы задержать строку 20 иа экране и прочитать ее.

FOR/NEXT (немного подробнее)



Добавь:

```
12  FOR X = 1 TO 3000
13  NEXT X
```

Выполни программу. Отлично! Такой цикл иазывается также *циклом ожидания.* Ты можешь управлять временем задержки, измеияя оператор FOR. Например, изменить на FOR X = 1 TO 1000 или как-то иначе.

Хочу напомиить тебе, что вместо двух строк 12 и 13 ты можешь написать только одну строку:

```
12  FOR X = 1 TO 3000:NEXT X
```

■ Посмотри другой пример:

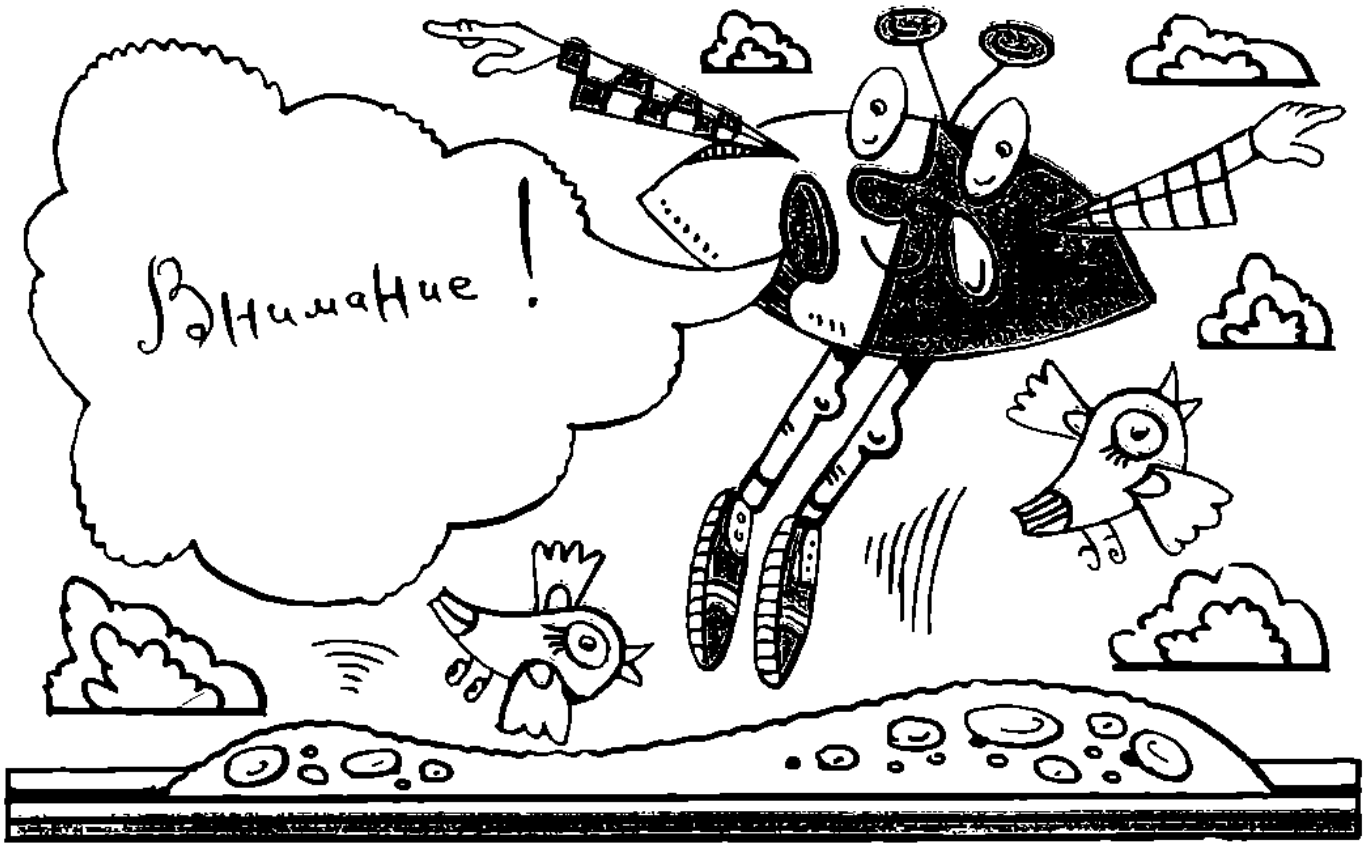
```
10  FOR X = 1 TO 10
20  PRINT "ПРОВЕРКА ЦИКЛА"
30  NEXT X
```

Выполни программу, и на экране 10 раз появится текст со строки 20.

Добавим еще:

```
25  FOR Y = 1 TO 2000
26  NEXT Y
```

Ты получишь тот же текст на экране, но он будет постепенно появляться, потому что, выполнив строку 20, компьютер выполняет цикл ожидания перед чтением строки 30 и затем берет следующее значение X.



■ Мы оставили пока без внимания один очень важный момент: у нас есть возможность поставить один цикл внутри другого. Такие циклы называются *внутренними* или, лучше, *вложенными циклами.*

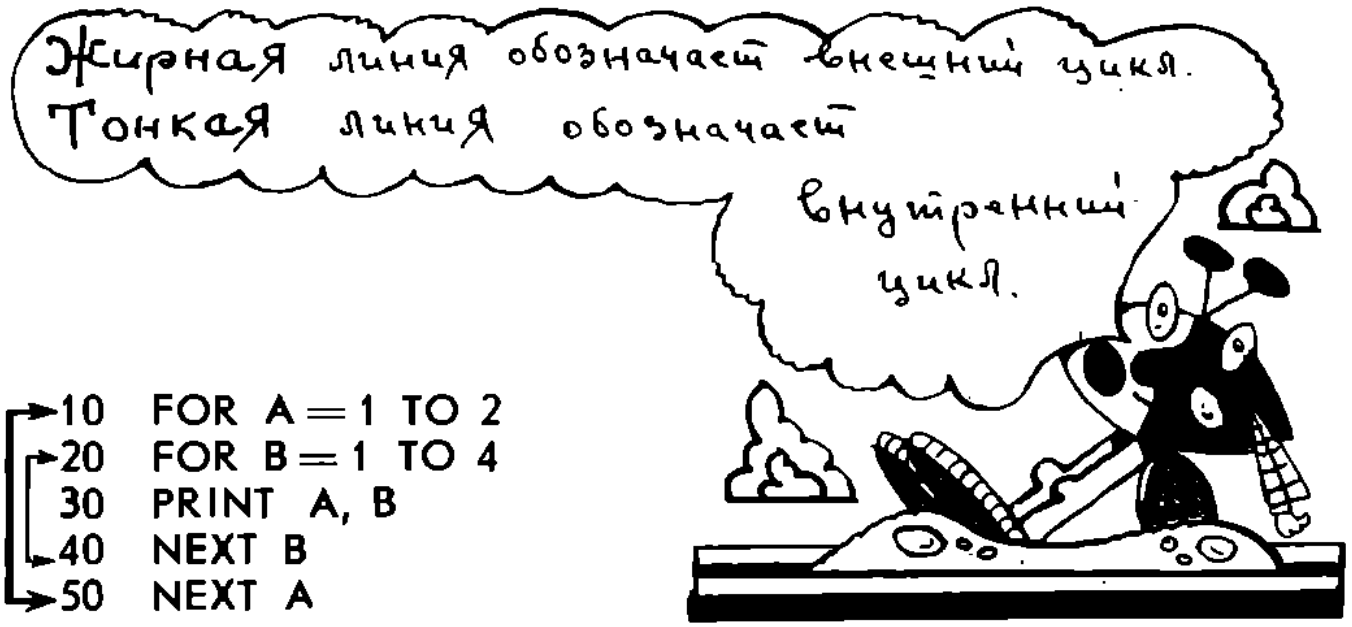
Почему вложенные? Потому что внешний цикл представляет собой гнездо, в которое вкладывается внутренний цикл.

И другая очеиь важная вещь: в нашем примере компьютер выполняет сначала цикл ожидания (внутренний цикл), а затем берет следующее значение из внешнего цикла.

■ Что произойдет, если вместо вложенного пустого цикла мы поставим нормальный цикл? Давай посмотрим это на примере.

FOR/NEXT (немного подробнее)

Набери программу:




```
10  FOR A = 1 TO 2
20  FOR B = 1 TO 4
30  PRINT A, B
40  NEXT B
50  NEXT A
```

Внешний цикл ограничен строками 10 и 50. Внутренний цикл — это строки 20, 30 и 40. Обрати внимание на важную деталь: сначала ты пишешь NEXT B (для закрытия внутреннего цикла), а затем NEXT A (для закрытия внешнего цикла).

Не ошибись, не поменяй их местами, потому что программа не заработает. Посмотрим, как работает программа. Внимательно и ие спеша следи за объяснениями.

В строке 10 дается начальное значение A (A = 1).

В строке 20 дается иачальное значение B (B = 1) и начинает выполняться виутренний цикл. A равно 1.

В строке 30 выводятся эти значения.

В строке 40 указываем компьютеру, что необходимо продолжать выполнение внутреннего цикла и брать следующее зиачеиие B(B=2). В этот момент A все еще равно 1.

Так как все еще выполняется внутренний цикл, компьютер возвратится на строку 30 и выведет теперь значения A и B (A=1 и B = 2).

Таким образом компьютер продолжает выполнение внутреннего цикла до тех пор, пока не даст переменной B все значения. При B = 4 компьютер закончит внутренний цикл и перейдет на строку 50, где присвоит следующее значение A. Теперь A = = 2.

Затем с этим значением он вернется, чтобы повторить весь внутренний цикл сначала. И так далее, пока не переберет все значения A.

Для того чтобы рассмотреть это яснее и подробнее, почему бы не попросить компьютер помочь нам? Измени строку 30:

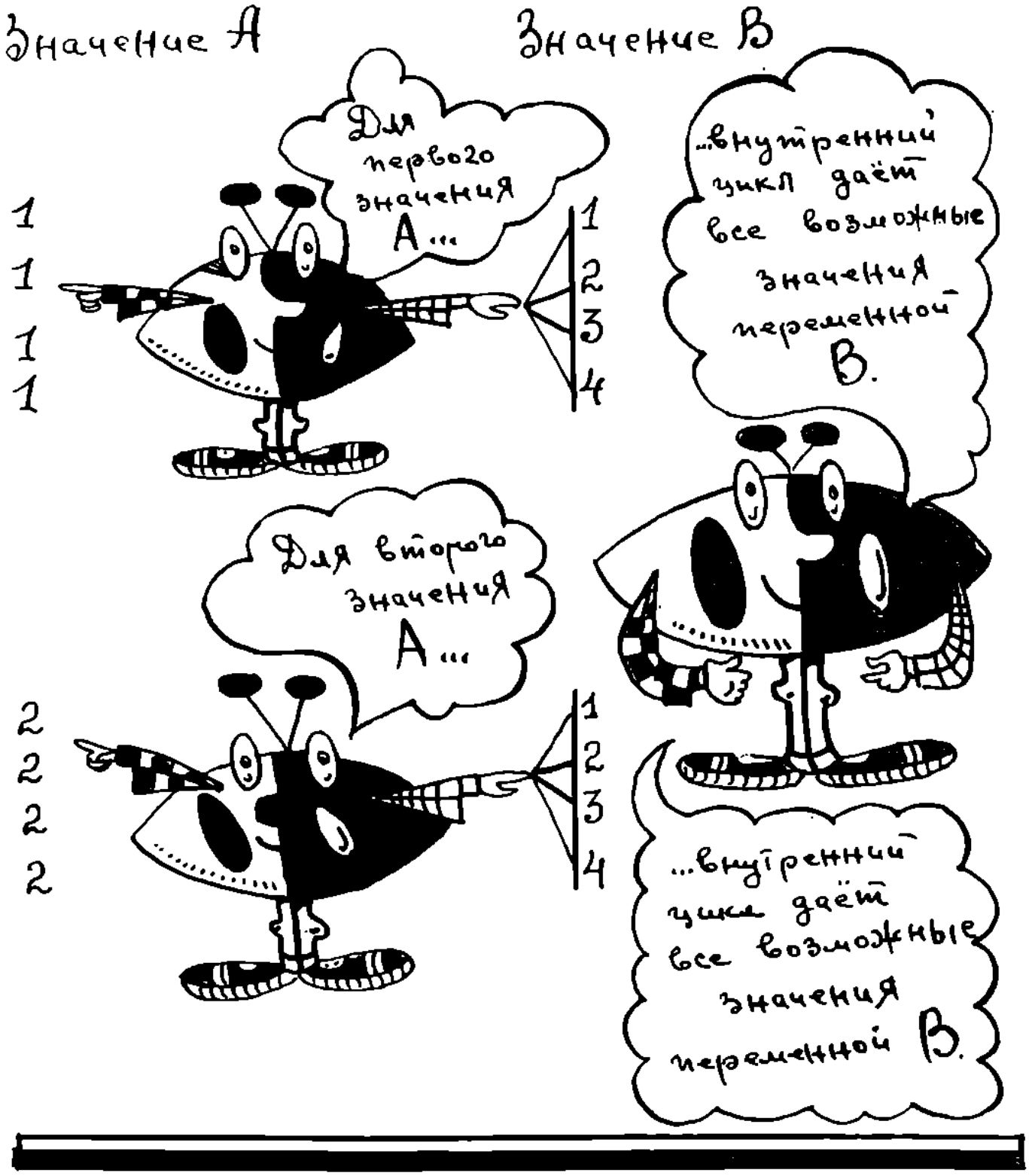
```
30  PRINT "A РАВНО ";A;"... ТЕПЕРЬ B РАВНО "; B
```

Добавь строку 35:

```
35  IF B = 4 THEN PRINT "ЗАКОНЧИЛИ ВНУТРЕННИЙ ЦИКЛ
                ДЛЯ ЗНАЧЕНИЯ A, РАВНОГО ";A
```

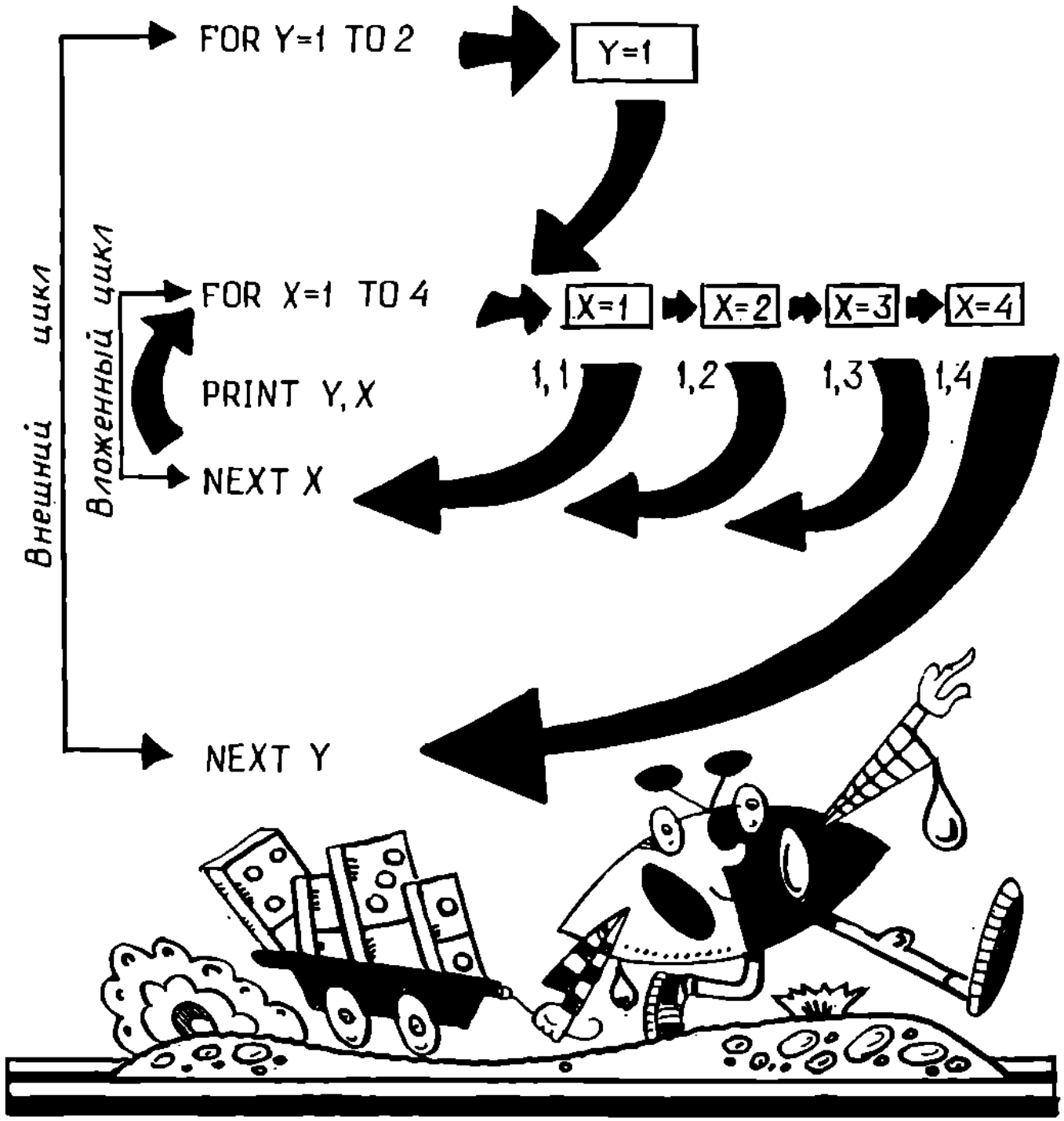
Выполни программу. На экране получишь различные значения, которые принимают одна за другой переменные A и B.

FOR/NEXT (немного подробнее)



Посмотри теперь на рисунок на следующей странице. Проследи путь, который совершает компьютер, когда выполняет внутренний цикл.

FOR/NEXT (немного подробнее)



Возвратись и прочитай еще раз все объяснения, при этом смотри на экран и на программу. И ты убедишься, что это просто и легко понять.

■ Посмотрим некоторые практические программы, в которых использованы вложенные циклы. Например, для получения списка всех фишек домнно введн:

```
10  FOR X=0 TO 6
20  FOR  Y=0 TO 6
30  PRINT  X; "!"; Y
40  NEXT Y
50  NEXT X
```

Так как цифры очень быстро меняются на экране и необходимо их зафиксировать, добавь один пустой цикл для замедления:

```
35  FOR A=1 TO 500:NEXT A
```

и ты сможешь читать появляющиеся значения.

■ Одна из форм программ для получения таблицы умножения будет:

```
10  CLS
20  PRINT "ТАБЛИЦА УМНОЖЕНИЯ"
30  PRINT "НАЖМИ ENTER ИЛИ ВК ПОСЛЕ КАЖДОГО РЕ-
    ЗУЛЬТАТА УМНОЖЕНИЯ"
40  FOR  A=1 TO 2500:NEXT A:CLS
50  FOR  X=1 TO 10
60  FOR  Y=1 TO 10
70  LET  C=X*Y
80  PRINT X; " УМНОЖИТЬ НА ";Y;" РАВНЯЕТСЯ "; C
90  INPUT E$
100 CLS
110 NEXT Y
120 NEXT X
```

FOR/NEXT (немного подробнее)

Посмотри, как в строке 40 введен цикл замедления для задержки стирания экрана. Внешний цикл образуют строки с 50 по 120. Вложенный цикл — строки с 60 по 110.

Обратн внимание на «хитрость» в строке 90. Посредством ее мы заставляем компьютер ожидать какие-то данные. Это делаем для того, чтобы экран не заполнялся таблицей умножения полностью. Когда мы вводим данные (все равно какие), достаточно нажать клавишу ENTER и компьютер продолжит выполнять программу. В разделе «Строки символов» мы посмотрим одну из форм достижения того же эффекта с использованием команды INKEY$.

Выполни программу.

Хорошо, пока достаточно. Как видишь, можно делать много разного с циклами. Даже подготовить календарь игр баскетбольных команд, составленных из друзей Артуро. Ты бы решился помочь ему? Если ты хочешь посмотреть, как это делается, посмотри программу на странице 209.

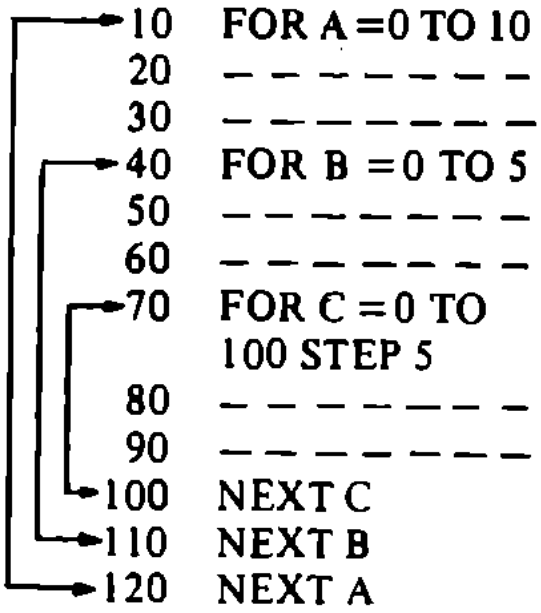
FOR/NEXT (немного подробнее)

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

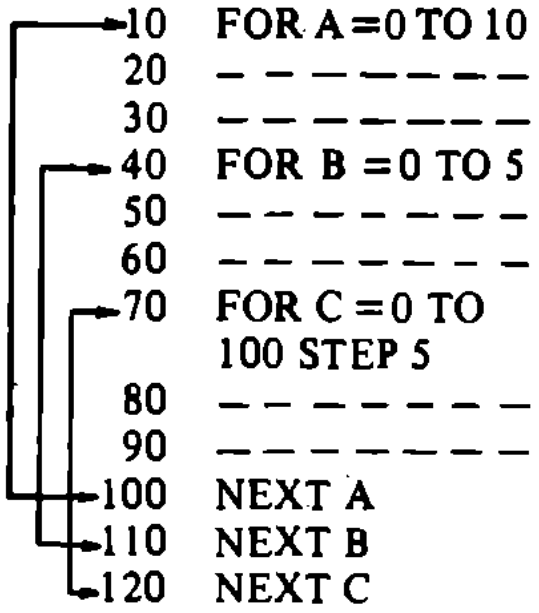
- Желательно, чтобы ребенок попрактнковался в использовании этого оператора, обладающего чрезвычайнымн возможностями н часто употребляемого в бейсике.

- Покажите ему, что он может использовать иеограинченное чнсло циклов внутрн одной программы, но должен правильно располагать предложение NEXT — так, чтобы циклы не скрестились бы, иначе произойдет ошнбка.

ПРАВИЛЬНО



НЕПРАВИЛЬНО



- При нспользованнн циклов следует иметь в виду, что не все компьютеры обладают одннаковой скоростью, поэтому цикл FOR X = 1 TO 1000 может замедлить выполнение программы на большее или меньшее время в завнсимостн от типа ЭВМ.

ЛОГИЧЕСКИЕ ОПЕРАЦИИ

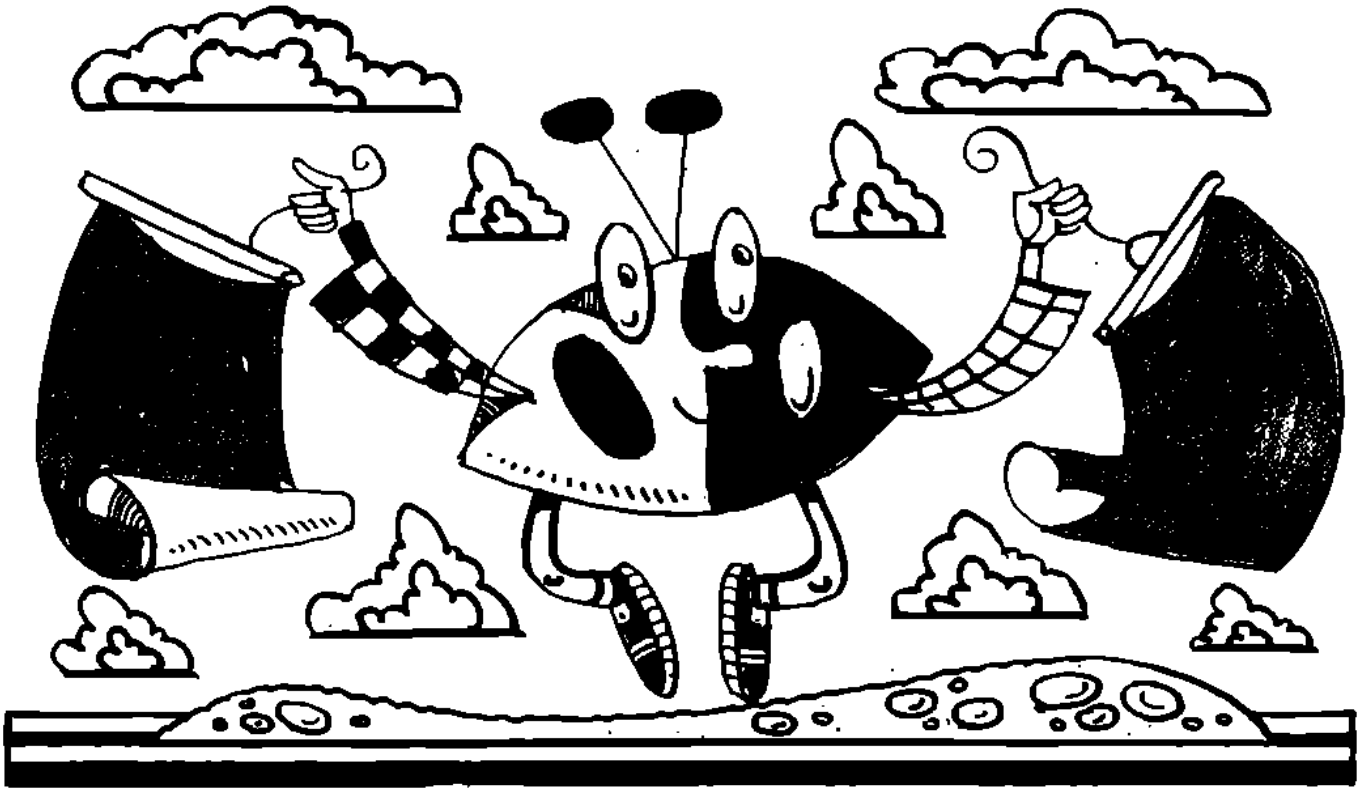
Ты помнишь, что, используя IF...THEN, мы ставили условия компьютеру? Например, мы говорим ему следующее:

```
IF A=B  THEN  GOTO  1
IF M>5  THEN  PRINT "ПРАВИЛЬНО"
IF A$<>"АРТУРО" THEN STOP
```

Ты, конечно, заметил, что в каждой из этих трех строк программы мы ставили только одно условие компьютеру. Бейсик, однако, позволяет тебе поставить больше одного условия, что увеличивает возможности программирования. Рассмотрим, как можно использовать логические операции.

Самыми распространенными являются следующие:

AND,	что обозначает	И
OR,	что обозначает	ИЛИ



■ Мы тебе покажем несколько примеров, чтобы ты понял, как они работают.

Артуро хочет играть в саду, и мама ему говорит:

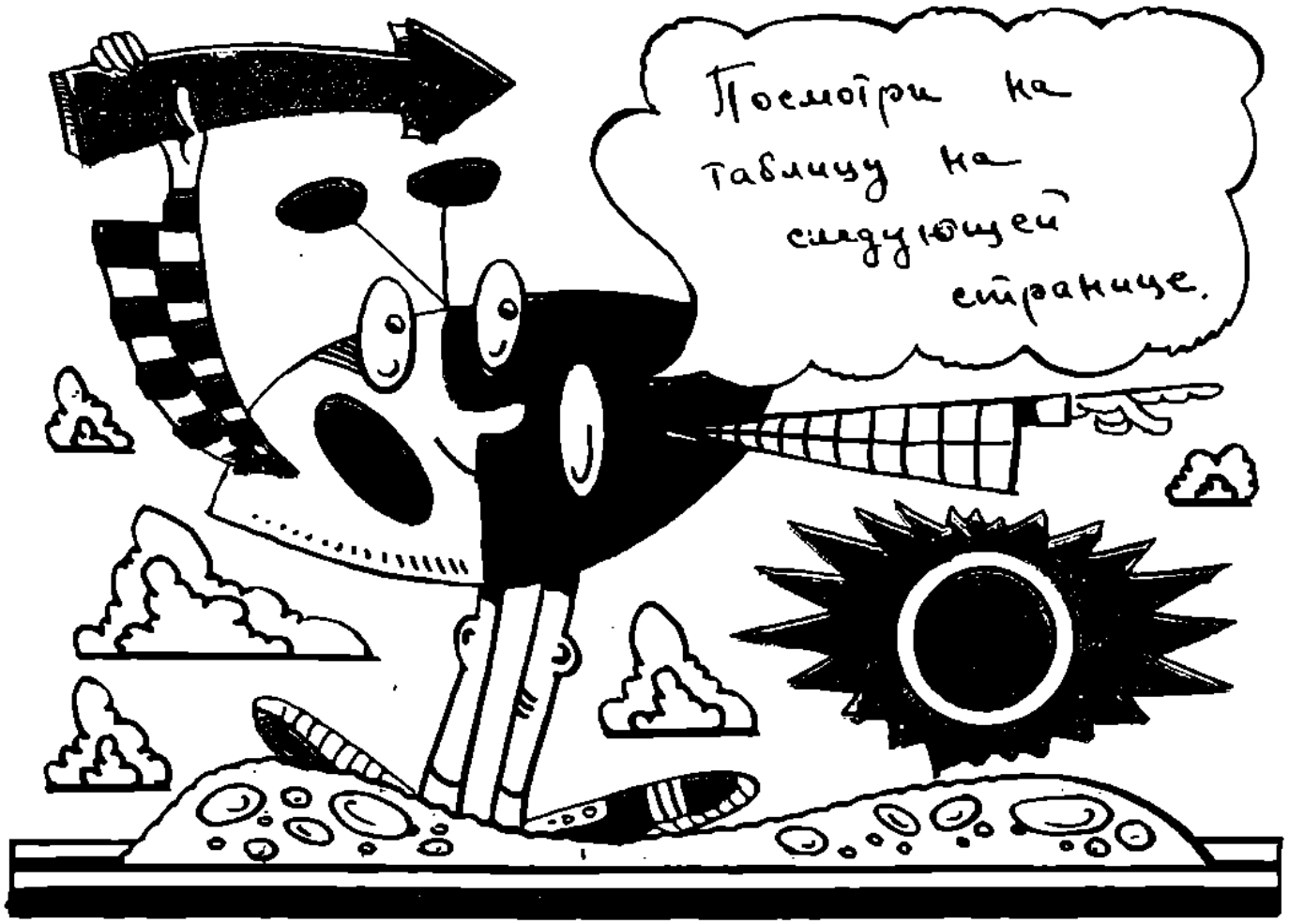
«Если у тебя есть мяч и время, то поиграй».

Мама Артуро задала ему два условия.

1-е условие — это наличие мяча. Выполняется?

2-е условие — это наличие времени. Выполняется?

Сейчас у Артуро есть мяч и есть время для игры. Поскольку оба условия выполняются, он может поиграть. Но так как не всегда эти условия будут выполняться, то составим таблицу и посмотрим, что может произойти.



1-е условие Есть мяч?	2-е условие Есть время?	Выполнение 1-го И (AND) 2-го условня	Результат
Да	Да	Да	Артуро играет
Да	Нет	Нет	Артуро ие играет
Нет	Да	Нет	Артуро не играет
Нет	Нет	Нет	Артуро не играет

Как видно из таблицы, только в том случае, если Артуро имеет мяч И время, он выйдет играть. В бейсике И обозначается AND. Начертим схему алгоритма, а затем составим и программу, чтобы воспроизвести эту ситуацию.

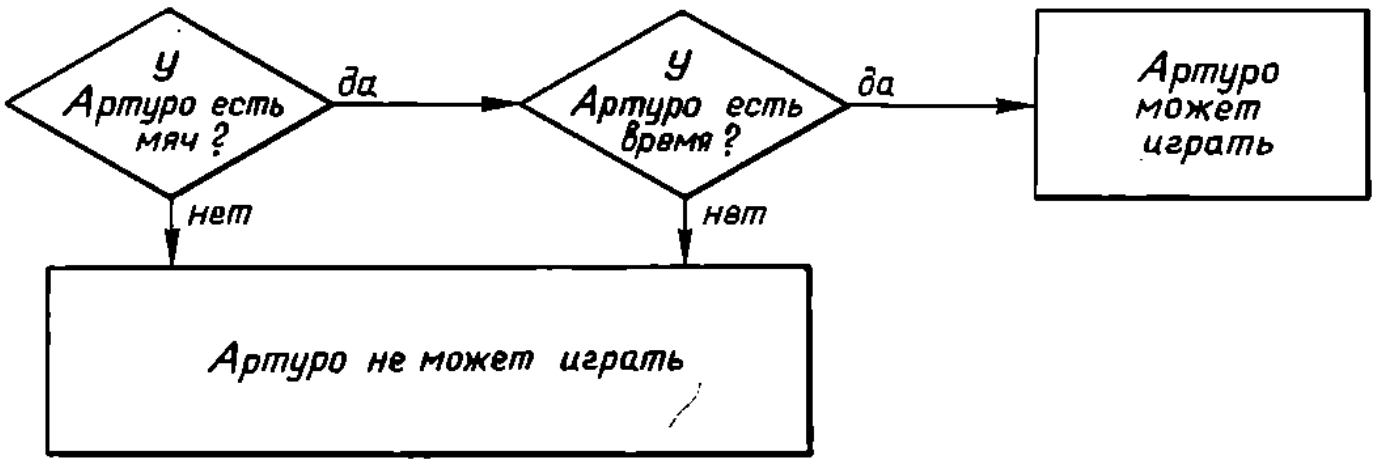
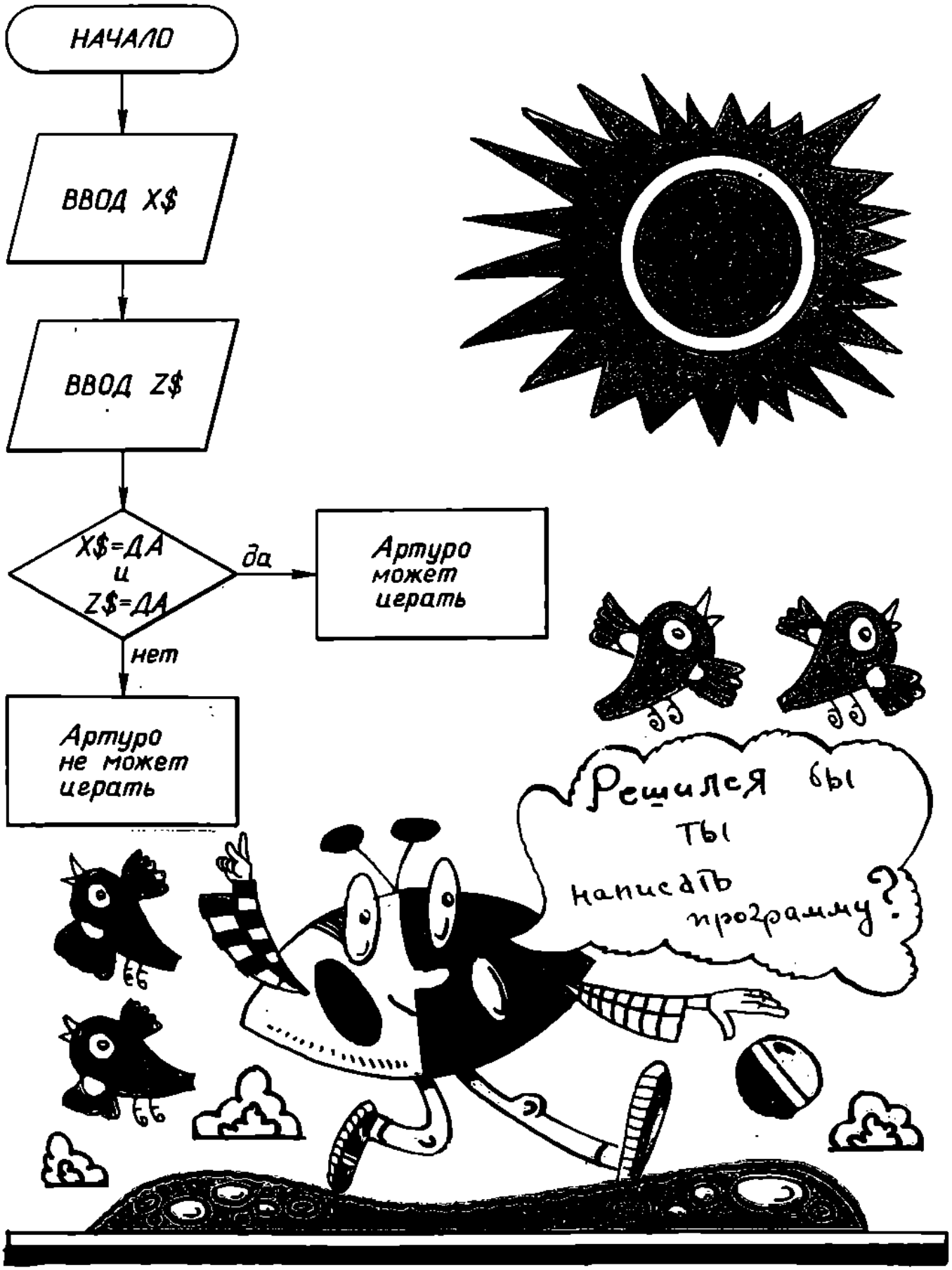


Схема алгоритма будет следующая:



Программа будет иметь вид:

```
 10  CLS
 20  PRINT "ОТВЕЧАЙ ДА ИЛИ НЕТ НА ВОПРОСЫ"
 30  PRINT
 40  INPUT "У АРТУРО ЕСТЬ МЯЧ?"; X$
 50  INPUT "У АРТУРО ЕСТЬ ВРЕМЯ?"; Z$
 60  IF X$ = "ДА" AND Z$ = "ДА" THEN 120
 70  PRINT
 80  PRINT "ТАК КАК НЕ ВЫПОЛНЯЮТСЯ"
 90  PRINT "ДВА УСЛОВИЯ"
100  PRINT "АРТУРО НЕ МОЖЕТ ИГРАТЬ"
110  GOTO 30
120  PRINT
130  PRINT "ТАК КАК ВЫПОЛНЯЮТСЯ"
140  PRINT "ДВА УСЛОВИЯ"
150  PRINT "АРТУРО МОЖЕТ ИГРАТЬ"
155  PRINT " — — — — — — — — — "
160  GOTO 30
```

Обрати внимание на строку 60. Ты сообщаешь компьютеру, что только в случае, если и X$ и Z$ одновременно будут равны «ДА» — он должен перейти на строку 120, с которой компьютер сообщит нам, что оба условия выполняются и Артуро может играть.

Выполни программу, отвечая на вопросы, которые тебе ставит компьютер. Сравни полученные результаты с таблицей на странице 138.

О! Есть такое, что тебе очень поможет при выполнении программы. Может случиться так, что когда ты введешь ответ на какой-то вопрос, то компьютер совершит ошибку. Например, может случиться, что в предыдущей программе ты нажмешь «ДЕ» или «ДУ» вместо «ДА».

Эти ошибки могут очень сильно влиять на результат выполнения программы. Проверь это, отвечая, например, «ДА» на первый вопрос и «ДУ» на второй. Ты увидишь, что компьютер выдаст ошибочный результат, так как он понял, что ты не написал «ДА», и понял это как «НЕТ». Твоих знаний уже достаточно, чтобы проанализировать оператор в строке 60 и последующие строки программы.

Если оба ответа не будут точными «ДА» и «ДА», то компьютер будет выполнять программу дальше, вместо того, чтобы перейти на строку 120.

Поэтому целесообразно ввести в программу строку-пароль, которая подсказала бы нам наличие ошибки такого типа.

В нашей последней программе мы можем использовать строку-пароль, которая содержит логическую операцию AND, хотя в этом случае мы ее используем не для решения задачи Артуро (играть или не играть ему в саду), а для того, чтобы выявить возможные ошибки при нажатии на клавиши.

Добавим в программу следующие строки:

```
 55  IF X$<> "ДА" AND X$ <> "НЕТ" OR Z$ <>
     "ДА" AND Z$ <> "НЕТ" THEN 165
165  PRINT
170  PRINT "ТЫ ДАЛ НЕПРАВИЛЬНЫЙ ОТВЕТ, ОТВЕТЬ ЕЩЕ
     РАЗ"
180  GOTO 30
```

Строка 55 означает, что если X$ отлично от «ДА» и отлично от «НЕТ» **или** Z$ отлично от «ДА» **и** отлично от «НЕТ», то твой ответ неправильный, так как программа тебя просила отвечать только «ДА» или «НЕТ». Обрати внимание, что в этой строке используется новая логическая операция OR (что обозначает ИЛИ).

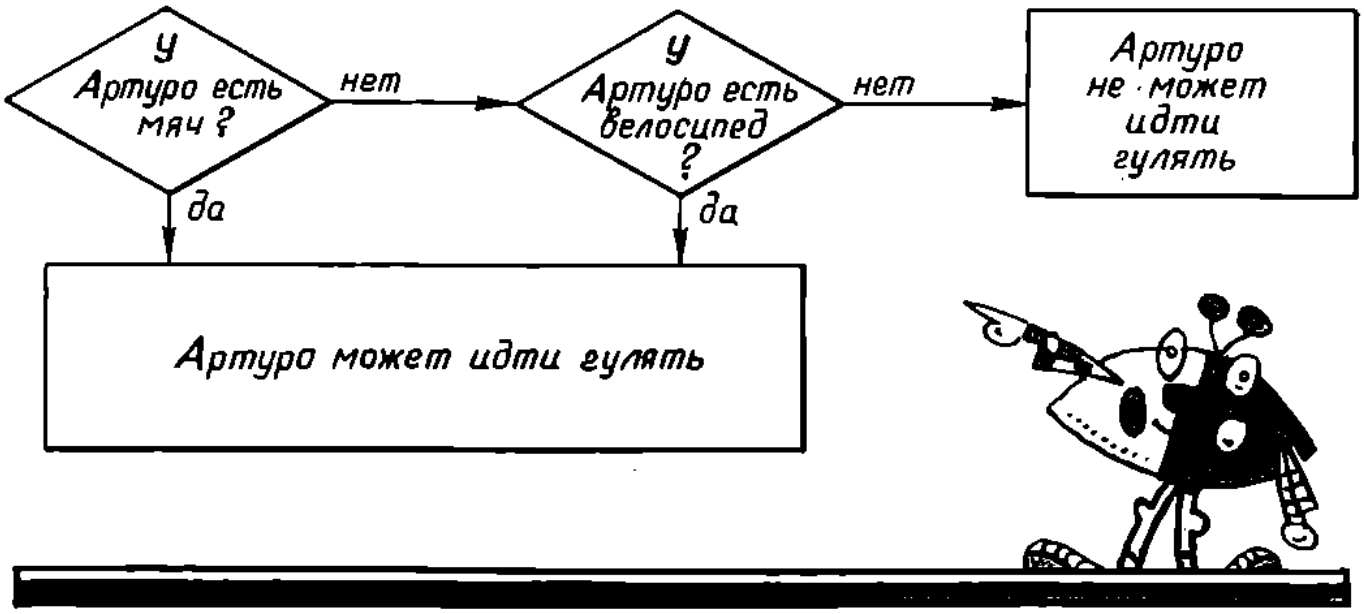
Давай ознакомимся с ней подробнее на одном примере. Мама говорит Артуро:
«ЕСЛИ У ТЕБЯ ЕСТЬ МЯЧ ИЛИ ЕСТЬ ВЕЛОСИПЕД, ТЫ МОЖЕШЬ ИДТИ ГУЛЯТЬ».

Мама Артуро поставила перед ним два условия, но, как видишь, достаточно, чтобы выполнялось только одно из двух условий, и он может идти гулять. Сделаем опять таблицу, чтобы посмотреть возможности, которые имеет Артуро для того, чтобы, выйти гулять:

1-е условие Есть мяч?	2-е условие Есть велосипед?	Выполняется 1-е **или** (OR) 2-е условие	Результат
Да	Да	Да	Артуро выходит гулять
Да	Нет	Да	Артуро выходит гулять
Нет	Да	Да	Артуро выходит гулять
Нет	Нет	Нет	Артуро ие выходит гулять

Как видишь, достаточно, чтобы у Артуро была одна из двух вещей, и он сможет выйти гулять. В бейсике ИЛИ пишется как OR.

В этом случае схема будет следующей:



Ты можешь написать также программу, чтобы рассмотреть возможности Артуро. В предыдущей программе в таком случае нужно изменить следующие строки:

```
 50  INPUT "АРТУРО ИМЕЕТ ВЕЛОСИПЕД?"; Z$
 60  IF X$ = "ДА" OR Z$ = "ДА" THEN 120
 85  PRINT "НИ ОДНО ИЗ"
 90  PRINT "ДВУХ УСЛОВИЙ"
130  PRINT "ТАК КАК ВЫПОЛНЯЕТСЯ ОДНО ИЗ"
140  PRINT "ДВУХ УСЛОВИЙ"
```

Теперь в строке 60 ты сообщаешь компьютеру: если выполняется условие, что X$ ИЛИ Z$ равно «ДА», то нужно перейти на строку 120; начиная с которой компьютер сообщает, что при выполнении одного из двух условий Артуро может выйти погулять. Выполни программу и сравни результаты с приведенными в таблице (с. 142). Обрати внимание, что здесь мы продолжаем использовать строку 55 («пароль»), как и в предыдущем примере.

Эти примеры наверняка помогут тебе лучше понять таблицы и функциоиирование логических операций. Эти таблицы часто используются, и, когда тебе в дальнейшем понадобиться с ними работать, они покажутся тебе простыми и легкими.

■ А сейчас мы поставим перед тобой очень простую, но... необычную задачу.

Артуро считает, что он разведчик, что у него есть пакет программ с секретными данными и он не хочет, чтобы кто-нибудь увидел этот пакет. Поэтому он решил поставить секретную клавишу-пароль в каждой своей программе, чтобы никто, кроме него, не смог бы пользоваться программами. Ты можешь сделать то же самое, поставив перед каждой программой примерно такие строки:

```
 1  LET C=0
 2  CLS:PRINT "ЭТО СЕКРЕТНАЯ ПРОГРАММА. УКАЖИТЕ
    ПРАВИЛЬНО ПАРОЛЬ":PRINT
 3  INPUT "СООБЩИТЕ НОМЕР И СЕКРЕТНОЕ СЛОВО";
    M, L$
 4  LET C=C+1: IF C=3 THEN 7
 5  IF M=25 AND L$="ДЕРЕВО" THEN 10
 6  GOTO 2
 7  PRINT "ВЫ НЕ ЗНАЕТЕ ПАРОЛЬ. МОЖЕТ БЫТЬ, ВЫ
    ШПИОН. У ВАС БЫЛО ТРИ ПОПЫТКИ. ЭТА ПРОГРАММА
    УНИЧТОЖАЕТСЯ ЧЕРЕЗ 5 СЕКУНД"
 8  FOR Y=1 TO 3000: NEXT Y
 9  CLS:STOP
10  . . . . .(ДАЛЬШЕ ИДЕТ ТВОЯ ПРОГРАММА)
```

Счетчик С в строке 4 считает число попыток. Когда С принимает значение 3, компьютер переходит на строку 7, где, после печати предупредительного текста, стирает программу с экрана. Обрати внимание на цикл ожидания в строке 8. В строке 9 можешь заменить команду STOP на NEW, но в этом случае программа полностью сотрется из памяти.

Придумай свой собственный пароль и измени, как хочешь, значения M и L$ в строке 5.

Сделаем еще одну замену. Поменяем в строке 5 AND на OR и выполним программу. Ты убедишься, что в этом случае достаточно правильно назвать один из двух паролей для того, чтобы компьютер позволил тебе выполнить основную программу.

О! В строке 5 мы написали в конце THEN 10 так как мы предполагаем, что твоя главная программа начинается со строки 10. Сделай изменения, если твоя программа начинается с другой строки.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Логические операции действуют только виутри двух возможностей: истинно (обозначается как 1) и ложно (обозначается как 0). Для аиализа возможностей, предоставляемых логическими операциями, используются *таблицы истинности*. Таблица истинности для логической операции AND такова:

X	Y	X AND Y
1	1	1
0	0	0
0	1	0
1	0	0

Таблица истиности для логической операции OR такова:

X	Y	X OR Y
1	1	1
0	0	0
1	0	1
0	1	1

- Эти понятия часто примеияются не только в информатике, но и в логике, математической логике. Полезио, чтобы ребенок усвоил их, и этого легче достичь, если теорию подкреплять примерами, как в этом пункте.

- Подбирая и используя примеры, учитывайте способности и иитересы ребеика.

- Существуют другие логические операции: NOT, XOR, NOR, но их фуикции более сложные и мы ие рассматривали такие операции, поскольку усвоить их могут только дети, старше тех, которым предназначена эта книга.

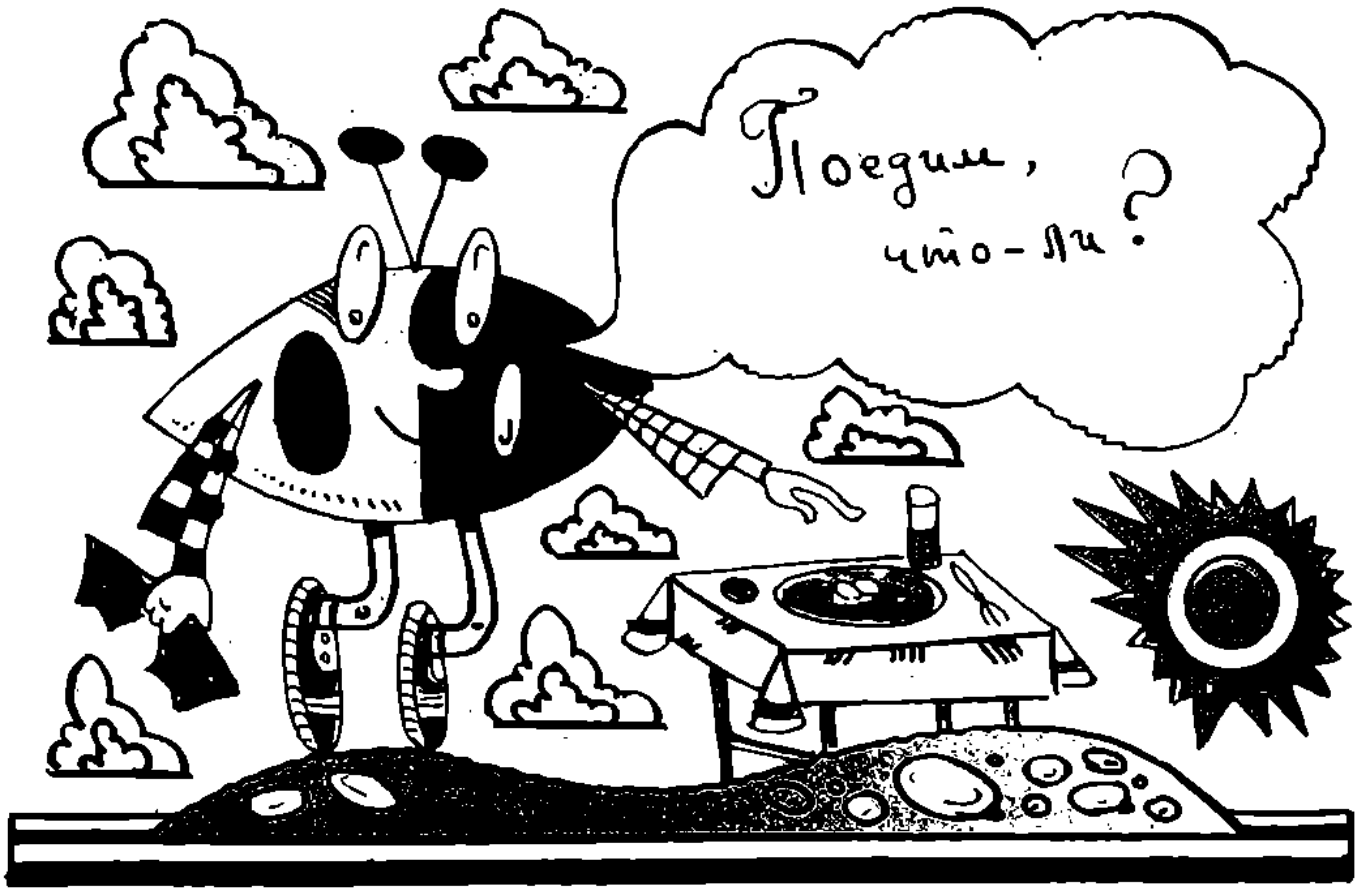
IF/THEN (немного подробнее)

А теперь покажем тебе, что такое *меню* и какие различные формы может принимать оператор IF/THEN.

Рассмотрим все это на примере. Введи следующую программу:

```
 10  CLS
 20  INPUT "ВВЕДИ ПЕРВОЕ ЧИСЛО"; A
 30  INPUT "ВВЕДИ ВТОРОЕ ЧИСЛО"; B
 40  PRINT "НАЖМИ 1 ДЛЯ УМНОЖЕНИЯ"
 50  PRINT "НАЖМИ 2 ДЛЯ ДЕЛЕНИЯ"
 60  PRINT "НАЖМИ 3 ДЛЯ СЛОЖЕНИЯ"
 70  PRINT "НАЖМИ 4 ДЛЯ РАБОТЫ С ДРУГИМИ ЧИС-
     ЛАМИ"
 80  PRINT "НАЖМИ 5 ДЛЯ ОКОНЧАНИЯ РАБОТЫ"
 90  INPUT "КАКУЮ ОПЕРАЦИЮ ХОЧЕШЬ ВЫПОЛНИТЬ?";
     X
100  IF X <> 1 AND X <> 2 AND X <> 3 AND
     X <> 4 AND X <> 5 THEN 220
110  IF X = 1 THEN 160
120  IF X = 2 THEN 180
130  IF X = 3 THEN 200
140  IF X = 4 THEN 10
150  IF X = 5 THEN 230
160  PRINT "РЕЗУЛЬТАТ УМНОЖЕНИЯ ";A;" НА ";B;" РА-
     ВЕН ";A*B
170  GOTO 90
180  PRINT "РЕЗУЛЬТАТ ДЕЛЕНИЯ ";A;" НА ";B;" РА-
     ВЕН ";A/B
190  GOTO 90
200  PRINT "РЕЗУЛЬТАТ СЛОЖЕНИЯ ";A;" ПЛЮС ";B;"
     РАВЕН ";A+B
210  GOTO 90
220  PRINT "ВЫ НЕПРАВИЛЬНО УКАЗАЛИ ВАШ ВЫБОР.
     НАЖМИТЕ ПРАВИЛЬНО": GOTO 90
230  STOP
```

Как видишь, эта программа очень проста и легка. В строках с 40 по 90 ты записал так называемое *меню*.



Нет, мы не будем есть. Но точно так же, как в ресторане, тебе дадут меню, чтобы ты выбрал еду, которая тебе нравится.

В начало программы ты можешь поставить меню, чтобы на этапе выполнения ты смог выбрать путь, который тебя больше интересует.

Таким образом, ты получишь более понятную и ясную программу. Введи меню во все сделанные тобой программы и ты сможешь выбирать между различными вариантами. Но не забудь поставить строку-пароль, чтобы избежать ошибок при вводе.

Обрати внимание на строку 160. Мы не присваивали специально ни одной переменной результат умножения A на B, а непосредственно указали компьютеру вывести A*B. То же самое происходит в строках 180 и 200.

■ Хорошо. А теперь будем использовать ON GOTO. Не все компьютеры позволят тебе пользоваться этим оператором,

так что сначала прочитай инструкцию для своего компьютера и выясни, сможешь ли ты пользоваться этим оператором.

Сотри строки 110, 120, 130,140, 150 и введи

```
110  ON X GOTO 160, 180, 200, 10, 230
```

Посмотри, результат выполнения новой программы такой же.

С оператором ON GOTO ты как будто говоришь компьютеру: «В зависимости от значения X ты перейдешь на строки...».

X всегда принимает значения 1, 2, 3 и так далее до возможного числа путей (вариантов), которые мы хотели бы ввести в программу.

Другая возможность, которую тебе дает ON,— это использование его с подпрограммами, то есть с формой ON X GOSUB...

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Операторы ON GOTO, или ON GOSUB не являются осиовиыми — оии просто облегчают процесс программироваиия. Ребенок может практиковаться с ними для приобретеиия непринуждеиности и скорости в программировании.

- Имейте в виду, что не все компьютеры имеют эти операторы. Если ваш не имеет, все равио объясните ребеику важиые понятия, описанные в этой главе, такие, как иаписание программы с меию (строки 40—90), испольэование строки выявления ошибок при вводе данных во время выполнения программы (строка 100), прямой вывод результата арифметических операций без предварительиого присваивания его какой-либо переменной (строки 160, 180, 200).

- Если вы убедитесь, что ребеиок относительно легко усваивает эти понятия, вы можете расширить его зиания об операторе IF...THEN, объяснив его в более полной форме: IF (условие) THEN (оператор) ELSE (оператор), то есть если условие выполияется, то выполияется оператор, следующий за служебным словом THEN; если не выполняется, то будет выполияться оператор, следующий за ELSE, иапример: IF A=5 THEN PRINT "УГАДАЛ" ELSE PRINT "ПРОМАХНУЛСЯ".

READ и DATA

Артуро собирается организовать мини-олимпиаду и хочет узнать, кто из его друзей и в каком из видов программы будет участвовать. Чтобы все было хорошо организовано, он хочет узнать эти данные с помощью компьютера.

Чтобы сделать это наиболее удобным способом, Артуро будет писать программу с использованием оператора READ/DATA. Поможем ему? Давай посмотрим шаг за шагом, как это делается.

Артуро имеет друзей, которые специализируются в таких трех видах спорта: БЕГ, БАСКЕТБОЛ и ПЛАВАНИЕ. Итак, первое, что мы делаем,— это подготавливаем операторы DATA с именами друзей Артуро и видами спорта, в которых они специализируются:

```
1  DATA ЛУИС, БЕГ
2  DATA ХУАН, БАСКЕТБОЛ
3  DATA КАРЛОС, ПЛАВАНИЕ
4  DATA ХОСЕ, БАСКЕТБОЛ
5  DATA ПЕДРО, БАСКЕТБОЛ
6  DATA ХОРХЕ, БЕГ
7  DATA МИГЕЛЬ, ПЛАВАНИЕ
8  DATA АРТУРО, БАСКЕТБОЛ
```

Эти данные затем будут прочитаны с помощью оператора READ. Но это мы посмотрим потом. Теперь сделаем МЕНЮ как уже знаешь:

```
10  CLS
20  PRINT "НАЖМИ 1 ДЛЯ БЕГУНОВ"
30  PRINT "НАЖМИ 2 ДЛЯ БАСКЕТБОЛИСТОВ"
40  PRINT "НАЖМИ 3 ДЛЯ ПЛОВЦОВ"
50  PRINT "НАЖМИ 4 ДЛЯ ОКОНЧАНИЯ ПРОГРАММЫ"
60  PRINT: INPUT "НАЖМИ НОМЕР СВОЕГО ВЫБОРА"; X
70  IF X>4 OR X<1 THEN PRINT
    "ВЫ ПЛОХО ВВЕЛИ СВОЙ ВЫБОР": GOTO 60
```

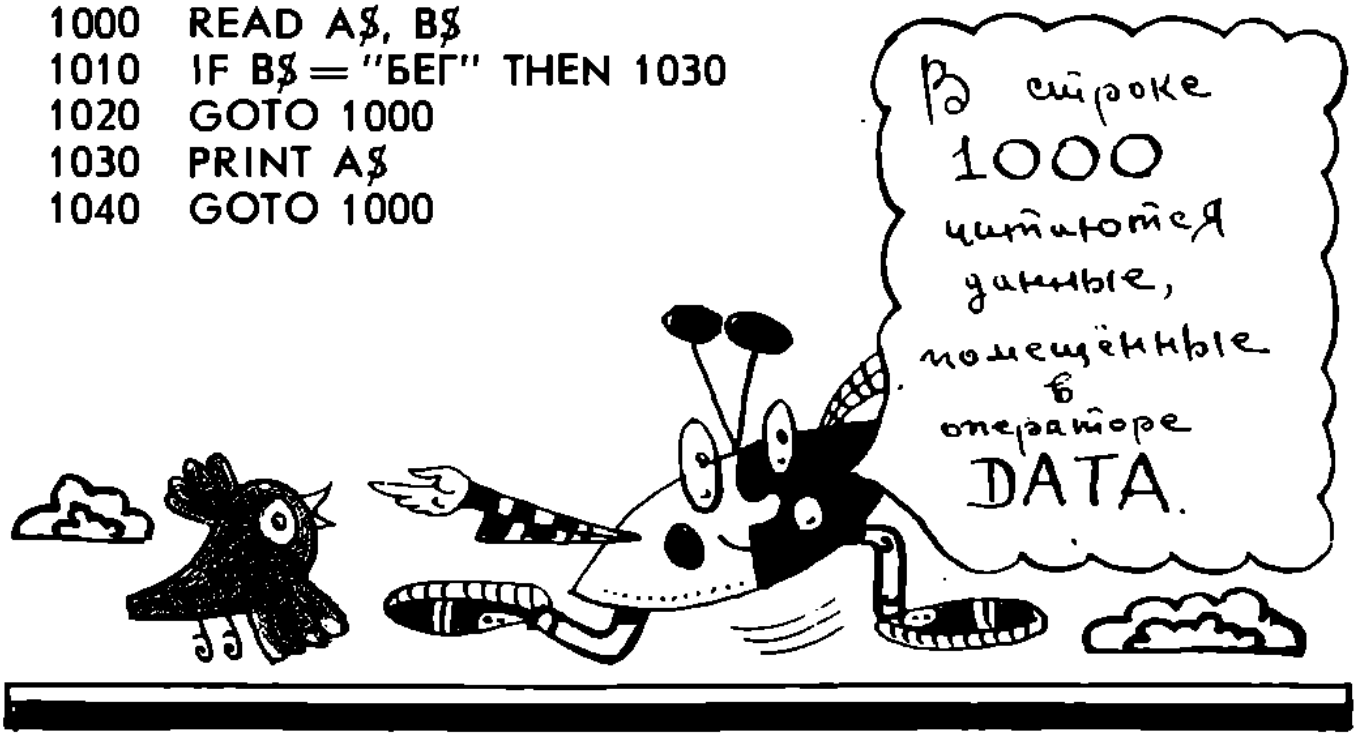
Обрати внимание на оригинальный и простой способ выявления ошибок при вводе в строке 70 (строка-«пароль»).

В строках 80, 90 и 100 сообщаешь компьютеру, чтобы он перешел в разные подпрограммы, в зависимости от выбранного тобой варианта:

```
 80  IF X=1 THEN GOSUB 1000
 90  IF X=2 THEN GOSUB 2000
100  IF X=3 THEN GOSUB 3000
110  IF X=4 THEN PRINT
     "КОНЕЦ ПРОГРАММЫ": GOTO 200
200  END
```

Напишем подпрограмму для поиска бегунов:

```
1000  READ A$, B$
1010  IF B$="БЕГ" THEN 1030
1020  GOTO 1000
1030  PRINT A$
1040  GOTO 1000
```



Проанализируем, как работает эта программа. В строке 1000 читаем данные, хранящиеся в строке DATA в основной программе. В строке 1010 сообщаем компьютеру, что когда B$ будет равно "БЕГ", то нужно идти на строку 1030, в которой выводится A$ (имя друга Артуро, который может выступать в этом виде спорта). Затем возвращаемся на строку 1000, где продолжаем читать еще не прочитанные данные из DATA.

Если B$ отлично от "БЕГ", компьютер продолжит выполнять программу и в строке 1020 получит приказ вернуться на строку 1000 для продолжения поиска данных.

Добавь также:

```
150  GOTO 20
```

Таким образом, когда ты захочешь организовать поиск, тебе не нужно всякий раз вводить команду RUN.

Выполни программу в том виде, как она есть. Но осторожно! Сейчас ты можешь выбрать только вариант 1, потому что ты не написал ещё подпрограммы других вариантов поиска.

На экране появится список друзей Артуро, которые могут выступать в беге. Кроме того, будет сообщение о том, что у компьютера нет больше данных для чтения в строке 1000. Поэтому необходимо вводить фиктивные данные:

```
9  DATA КОНЕЦ, КОНЕЦ
```

а в подпрограмму добавить строку

```
1015  IF B$ = "КОНЕЦ" THEN PRINT
      "ВСЕ БЕГУНЫ НАЙДЕНЫ": RETURN
```

тем самым заканчивая подпрограмму.

С RETURN первая строка, которую выполнит компьютер дальше, будет 150.

Теперь давай-ка напишем подпрограмму, которая выдаст список баскетболистов.

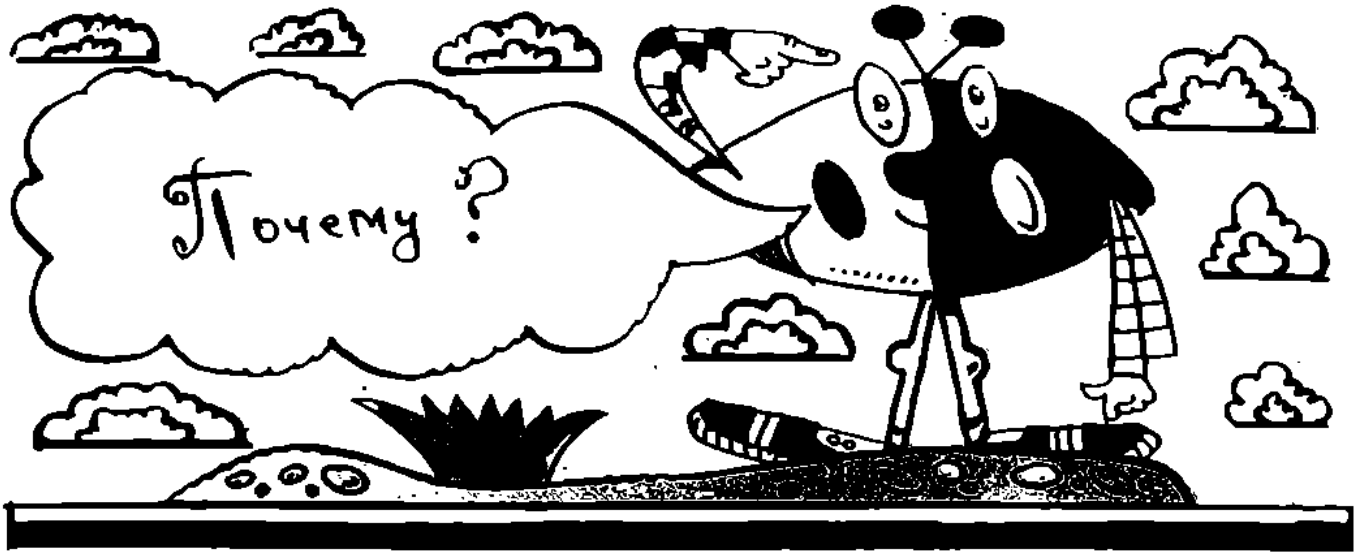
```
2000  READ A$, B$
2010  IF B$="БАСКЕТБОЛ" THEN 2030
2015  IF B$="КОНЕЦ" THEN PRINT
      "ВСЕ БАСКЕТБОЛИСТЫ НАЙДЕНЫ": RETURN
2020  GOTO 2000
2030  PRINT A$
2040  GOTO 2000
```

Нет необходимости объяснять ее, поскольку она работает точно так же, как предыдущая программа.

Но внимание! Выполни теперь программу. Выбери вариант, который требуется (1 или 2). Компьютер выведет тебе на экран список друзей, которые могут выступать по тому виду спорта, который ты указал.

Теперь выбери следующий вариант. И компьютер не выведет тебе ни одного имени и даст тебе сообщение об ошибке:

"Не имею данных для чтения".



Объяснение этому достаточно простое. Когда READ читает значения, хранящиеся в DATA, это означает, что каждый раз он как будто в каждом прочитанном значении ставит точку или фиктивный знак, чтобы обозначить, что оно уже прочитано.

RESTORE

Когда ты попросишь пойти по второму варианту и компьютер начнет читать данные, чтобы еще раз повторить отбор, он определит, что все данные уже прочитаны, то есть имеется фиктивный знак, и он их проигнорирует. Поэтому и появится сообщение, что компьютер не имеет данных.

Это мы ему можем сказать оператором RESTORE. Этот оператор заставляет компьютер читать DATA еще раз. Поэтому для того, чтобы всегда, когда нам нужно, можно было читать все DATA, ставим оператор RESTORE во всех наших подпрограммах.

Добавим строки:

```
 900  RESTORE : CLS
1900  RESTORE : CLS
```

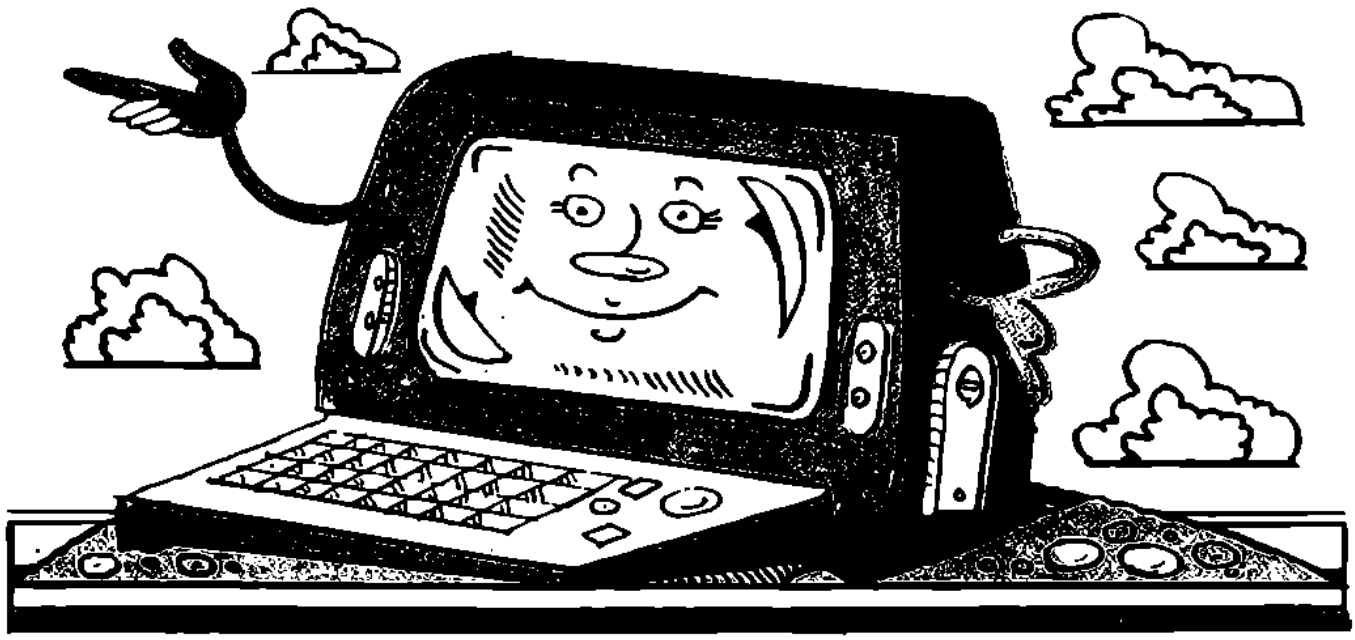
Изменим строки 80, 90 и 100 следующим образом:

```
 80  IF X = 1 THEN GOSUB 900
 90  IF X = 2 THEN GOSUB 1900
100  IF X = 3 THEN GOSUB 2900
```

Добавь, наконец, подпрограмму для вывода списка пловцов:

```
2900  RESTORE : CLS
3000  READ A$, B$
```

```
3010  IF B$="ПЛАВАНИЕ" THEN 3030
3015  IF B$="КОНЕЦ" THEN PRINT
      "ВСЕ ПЛОВЦЫ НАЙДЕНЫ": RETURN
3020  GOTO 3000
3030  PRINT A$
3040  GOTO 3000
```



ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Операторы READ, DATA, RESTORE очень полезны, так как намного упрощают работу по прнсвоению значений переменных. Полезно, чтобы ребенок уже в этой фазе обучення нормально нспользовал их как вспомогательное средство программнрования.

- Проанализируйте предыдущую программу шаг эа шагом. Не переходите на новую фазу, пока ребенок хорошо не поймет предыдущую.

- Некоторые компьютеры требуют, чтобы данные, включенные в оператор DATA, были в кавычках. В другнх это не так.

- Обратите вниманне на строки 1010, 2010, 3010. Вудьте внимательны. Не сделайте ннкакой ошнбки и не введите пробелы при напнсанин и вводе текстов в кавычках, которые прнсванваются переменным. Если вы все же сделалн это и затем прн выполненни выберете

вид спорта для поиска, то компьютер будет искать в DATA значение, которого не существует, так как компьютер интерпретирует пробел как символ.

- Учитывая те же критерии и внеся соответствующие изменения, вы можете расширить список видов спорта, добавив другие виды, или дать больше вариантов выбора (например, возраст участников). Если у вас будут трудности, то посмотрите программу на страиице 212.

- Если писать RESTORE, а за иим номер строки, которая содержит DATA, то компьютер будет читать данные с той строки, номер которой мы указали. Например, в нашей программе, RESTORE 4 проигнорирует имеиа ЛУИС, ХУАН, КАРЛОС при втором и следующем чтении.

- Данные для программы были включены в неё в строках 1—9 для большей ясности. Но таким же образом можно их поставить и в конце программы или даже в виде: DATA ЛУИС, БЕГ, ХУАН, БАСКЕТБОЛ, КАРЛОС, ПЛАВАНИЕ, ХОСЕ, ... и т. д.

DIM

Давайте ознакомимся с одним оператором, который позволяет сэкономить много времени. С тех пор как Артуро научился пользоваться им, он может использовать шире все возможности своих программ. И действительно, Артуро экономит много времени.

Как же он узнал оператор DIM? Все произошло в один день, когда наш друг захотел классифицировать некоторые из своих вещей.

Сначала он сделал следующие списки:

Книги	Игрушки	Одежда
ПРИКЛЮЧЕНИЯ	МЯЧ	БРЮКИ
РАЗВЕДЧИКИ	ВЕЛОСИПЕД	РУБАШКА
ПАРТИЗАНЫ	РАКЕТКА	ШАПКА
КОСМОС	РОЛИКИ	ТУФЛИ
СКАЗКИ		ПИЖАМА
ФАНТАСТИКА		

Затем он решил обозначить каждый предмет одной переменной. И стал писать.

```
LET A$="ПРИКЛЮЧЕНИЯ"
LET B$="РАЗВЕДЧИКИ"
. . . . . .
. . . . . .
```

Артуро прав: он потратит много времени на определение всех переменных. Кроме того, если ему потом еще придется добавить новые предметы, ему может не хватить переменных. А если он захочет, чтобы компьютер дал ему список всех предметов, классифицированных по видам (книги, игрушки, одежда), то ему нужно вставить в свою программу операторы типа PRINT A$, PRINT B$, PRINT C$ и т. д.

Тогда он решил использовать так называемые *переменные с индексом.* Например, такая переменная с индексом имеет вид:

A(1), A(2), A(3), . . . и т. д.

Как видишь, все они имеют одну и ту же букву, но все они являются различными переменными.

Работа с ними имеет большие преимущества. Во-первых, присвоить этим переменным значения можно быстрее, используя цикл FOR/NEXT. Давай поработаем с книгами Артуро.

Сначала напишем:

```
 5  CLS
10  DIM L$(6)
```

Этим мы хотим сказать, что компьютер зарезервирует в своей памяти 6 ячеек для 6 переменных, которые будут иметь один и тот же указатель или ту же букву. В нашем случае переменные будут: L$(1), L$(2), L$(3), L$(4), L$(5), L$(6). Давай присвоим значения этим переменным. Поскольку все они имеют один и тот же указатель, то не надо писать 6 строк программы с LET, так как мы можем использовать цикл FOR/NEXT.

Напиши:

```
20  FOR N=1 TO 6
25  PRINT "НАПИШИ
    НАЗВАНИЕ КНИГИ"
30  INPUT L$(N)
35  CLS
40  NEXT N
```

Цикл, который начинается в строке 20, выполняется так, что N будет последовательно принимать значение от 1 до 6. Шесть раз компьютер нас спросит, какое значение мы хотим присвоить L$(N). И если ты следуешь такому же порядку, что и Артуро в начале этой главы, то переменные примут такие значения:

```
L$(1) = "ПРИКЛЮЧЕНИЯ"
L$(2) = "РАЗВЕДЧИКИ"
L$(3) = "ПАРТИЗАНЫ"
L$(4) = "КОСМОС"
L$(5) = "СКАЗКИ"
L$(6) = "ФАНТАСТИКА"
```

Компьютер уже имеет эти значения в памяти. Для пробы введи, например, PRINT L$(3) или PRINT L$(5).

Компьютер тебе ответит!

Теперь мы можем сделать так, чтобы компьютер с помощью одной команды вывел значения всех переменных, имеющих общий указатель (общее имя). Добавим в программу следующие строки:

```
50  FOR N = 1 TO 6
60  PRINT L$(N)
70  NEXT N
```

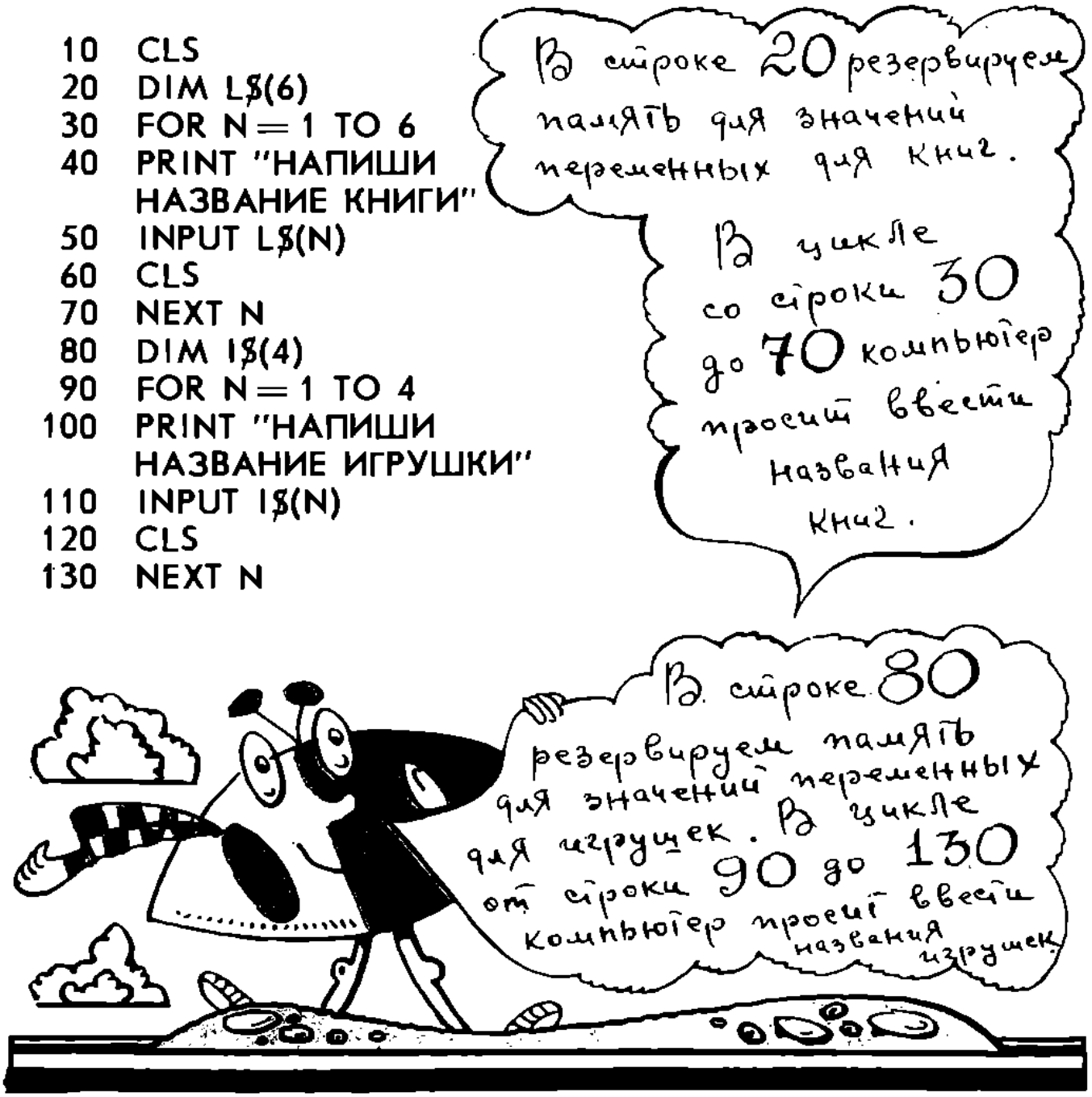
Этим циклом компьютер тебе выведет названия всех книг Артуро. Видишь, сколько времени и труда ты сэкономил! Если мы хотим сделать то же самое для игрушек и для одежды Артуро, то лучше всего использовать меню.

Советуем тебе использовать указатели с именами I$ для игрушек и R$ для одежды.

■ Давай сделаем программу, чтобы знать все имеющиеся у Артуро предметы, классифицированные по группам. Начнем сначала. Сотрем предыдущую программу.

Сначала подготовим компьютер для получения всех данных:

```
 10  CLS
 20  DIM L$(6)
 30  FOR N = 1 TO 6
 40  PRINT "НАПИШИ
     НАЗВАНИЕ КНИГИ"
 50  INPUT L$(N)
 60  CLS
 70  NEXT N
 80  DIM I$(4)
 90  FOR N = 1 TO 4
100  PRINT "НАПИШИ
     НАЗВАНИЕ ИГРУШКИ"
110  INPUT I$(N)
120  CLS
130  NEXT N
```

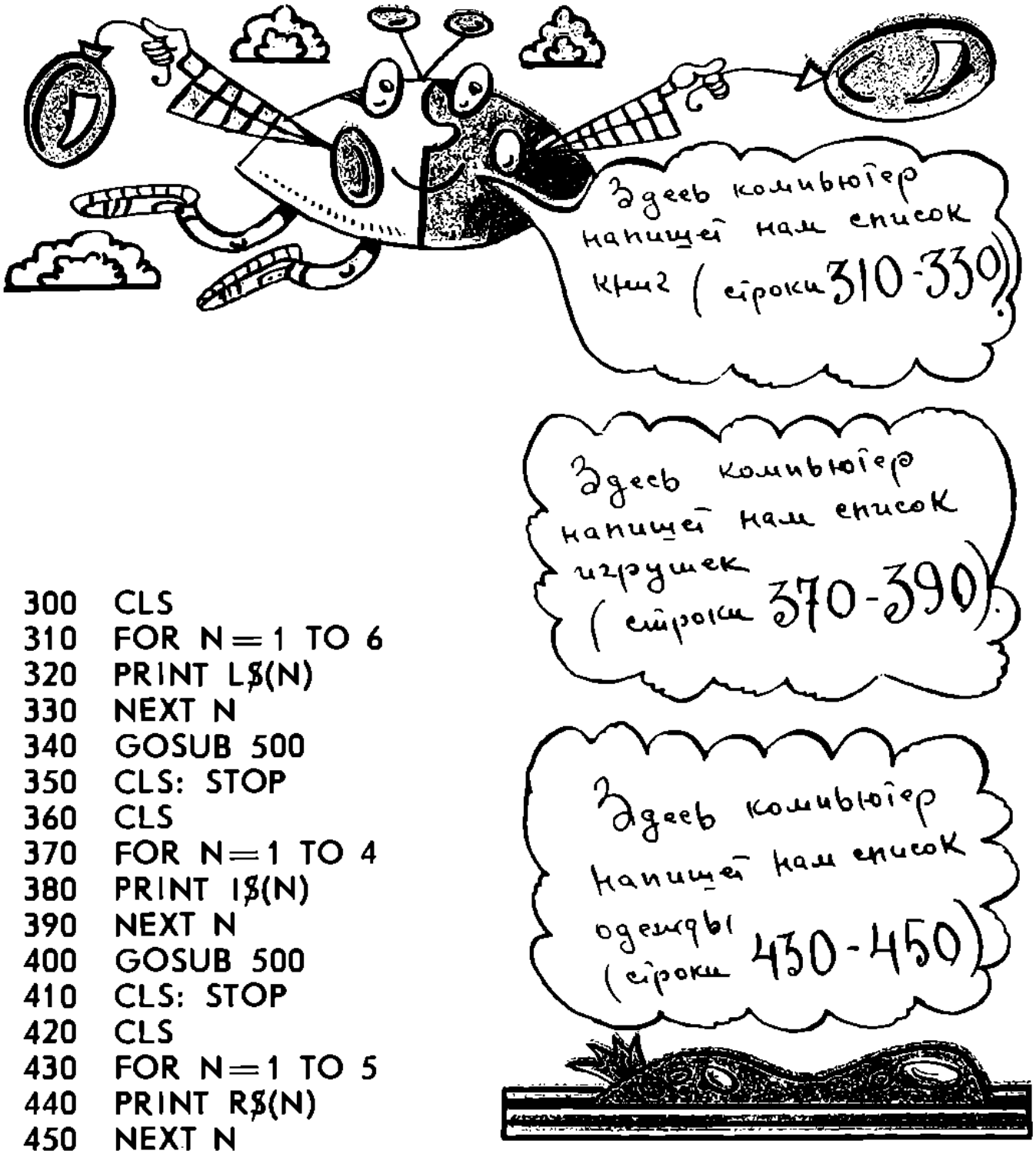


```
140  DIM R$(5)
150  FOR N=1 TO 5
160  PRINT "НАПИШИ
     НАЗВАНИЕ ОДЕЖДЫ"
170  INPUT R$(N)
180  CLS
190  NEXT N
```

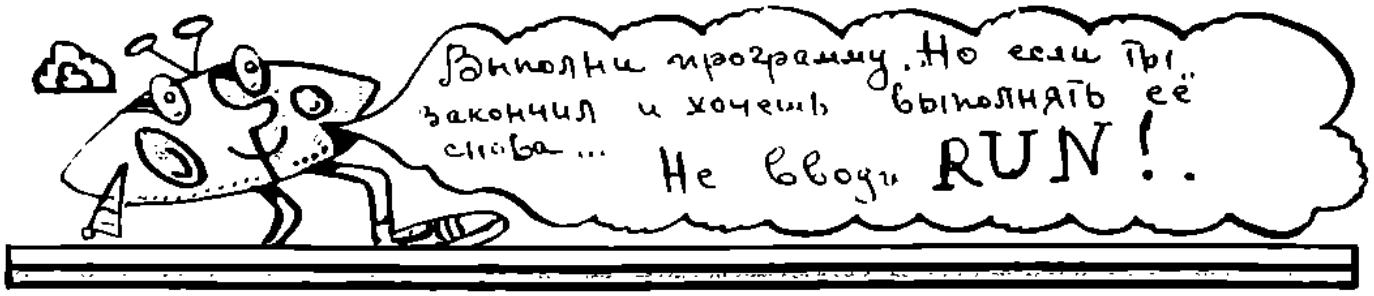
Теперь напишем меню для того, чтобы в любой момент мы могли выбрать какой-нибудь из трех имеющихся у нас списков предметов.

```
200  CLS
210  PRINT "ВОЗМОЖНОСТИ"
220  PRINT "ДЛЯ КНИГ НАЖМИ 1"
230  PRINT "ДЛЯ ИГРУШЕК НАЖМИ 2"
240  PRINT "ДЛЯ ОДЕЖДЫ НАЖМИ 3"
250  INPUT "НАЖМИ НУЖНЫЙ НОМЕР"; X
260  IF X=1 THEN 300
270  IF X=2 THEN 360
280  IF X=3 THEN 420
290  IF X<>1 AND X<>2 AND X<>3 THEN PRINT
     "ТЫ ПЛОХО ВВЕЛ НУЖНЫЙ НОМЕР": GOTO 250
```

Наконец, напишем ту часть программы, которая позволит нам вывести нужный список.



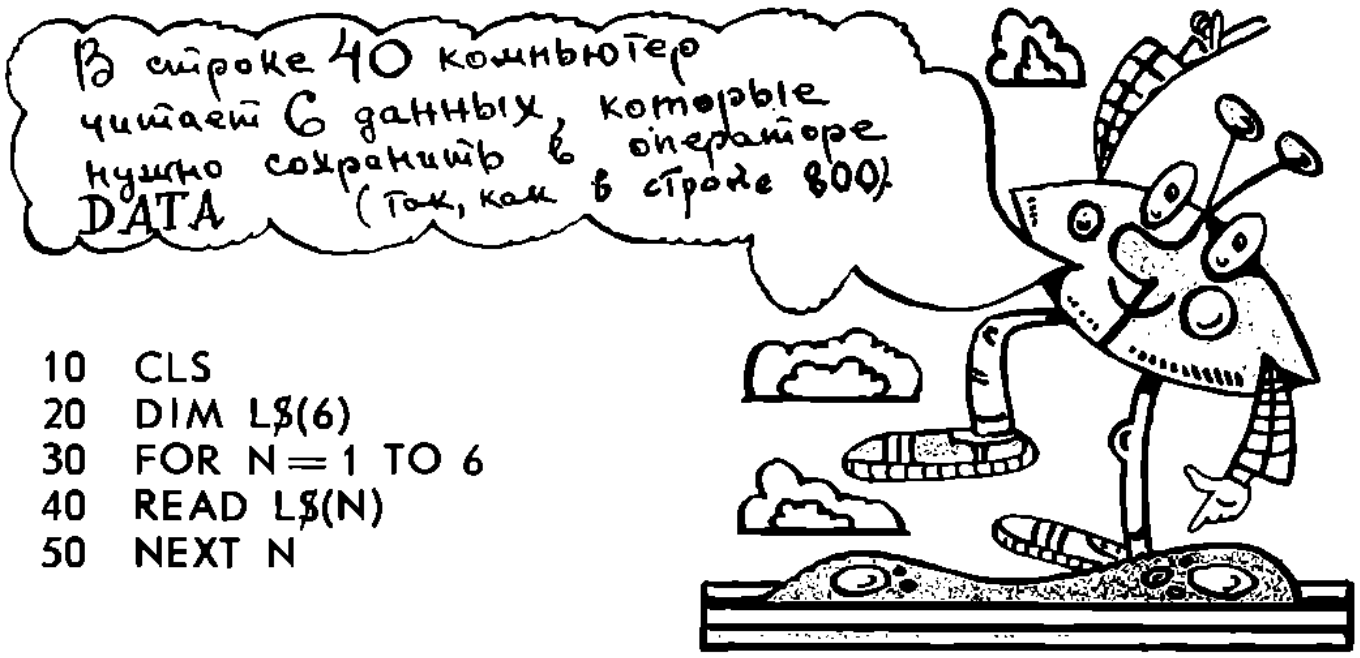

```
300  CLS
310  FOR N=1 TO 6
320  PRINT L$(N)
330  NEXT N
340  GOSUB 500
350  CLS: STOP
360  CLS
370  FOR N=1 TO 4
380  PRINT I$(N)
390  NEXT N
400  GOSUB 500
410  CLS: STOP
420  CLS
430  FOR N=1 TO 5
440  PRINT R$(N)
450  NEXT N
460  GOSUB 500
470  END
500  INPUT "НАЖМИ КЛАВИШУ К ДЛЯ ПРОДОЛЖЕНИЯ
     РАБОТЫ ИЛИ ЛЮБУЮ ДРУГУЮ КЛАВИШУ ДЛЯ ОКОН-
     ЧАНИЯ РАБОТЫ"; K$
510  IF K$="K" THEN 200
520  RETURN
```



Эта программа имеет один недостаток: каждый раз, когда ты вводишь RUN и программа выполняется, тебе необходимо снова вводить все данные: ты ведь знаешь, что RUN все переменные в памяти обнуляет.

Если ты имеешь программу, записанную на кассету вместе с введенными данными, то единственный способ сохранить данные — это ввести GOTO 200 вместо RUN. В строке 200 программа начинает работу и запрашивает вариант работы. Но это тоже очень неудобно.

■ Программу можно написать так, что компьютер сохранит в памяти все данные. Можем использовать операторы READ/DATA. Основная структура программы не изменится, но для ввода данных в память компьютера напишем:




```
10  CLS
20  DIM L$(6)
30  FOR N=1 TO 6
40  READ L$(N)
50  NEXT N
```

В какой-нибудь части программы (желательно в конце) введи оператор DATA:

```
800 DATA ПРИКЛЮЧЕНИЯ, РАЗВЕДЧИКИ, ПАРТИЗАНЫ,
    КОСМОС, СКАЗКИ, ФАНТАСТИКА
```

Продолжив работу тем же методом, ты должен внести такие же изменения для игрушек (строки 80—130) и для одежды (строки от 140 до 190). Тебе необходимо также написать еще 2 строки DATA — для игрушек и одежды.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Оператор DIM часто применяется для организации списков н таблиц. Необходимо, чтобы ребенок хорошо понял его назначение до того, как мы начнем изучать другне темы, в которых используется этот материал.

- Прнведенный в этой главе пример нетнпичен (по своей структуре) для реальной жнзни, но помогает ребенку понять назначенне этого оператора в программе.

- В большинстве компьютеров DIM прнменяется только тогда, когда необходнмо нспользовать более 10 индексированных переменных.

  При попытке использовать бóльшее количество предварительно объявленных переменных мы получим от компьютера сообщение об ошибке.

- В нашей программе употреблялись снмвольные переменные, чтобы помочь ребенку лучше разобраться в этом операторе. DIM используется точно так же н с числовыми переменнымн. Когда используются числовые переменные необходимо иметь в виду, что некоторые компьютеры требуют, чтобы прн определенни числа переменных было указано максимальное число разрядов, какое может иметь значение переменной. Так, в нашем примере в строке 20 можно напнсать:

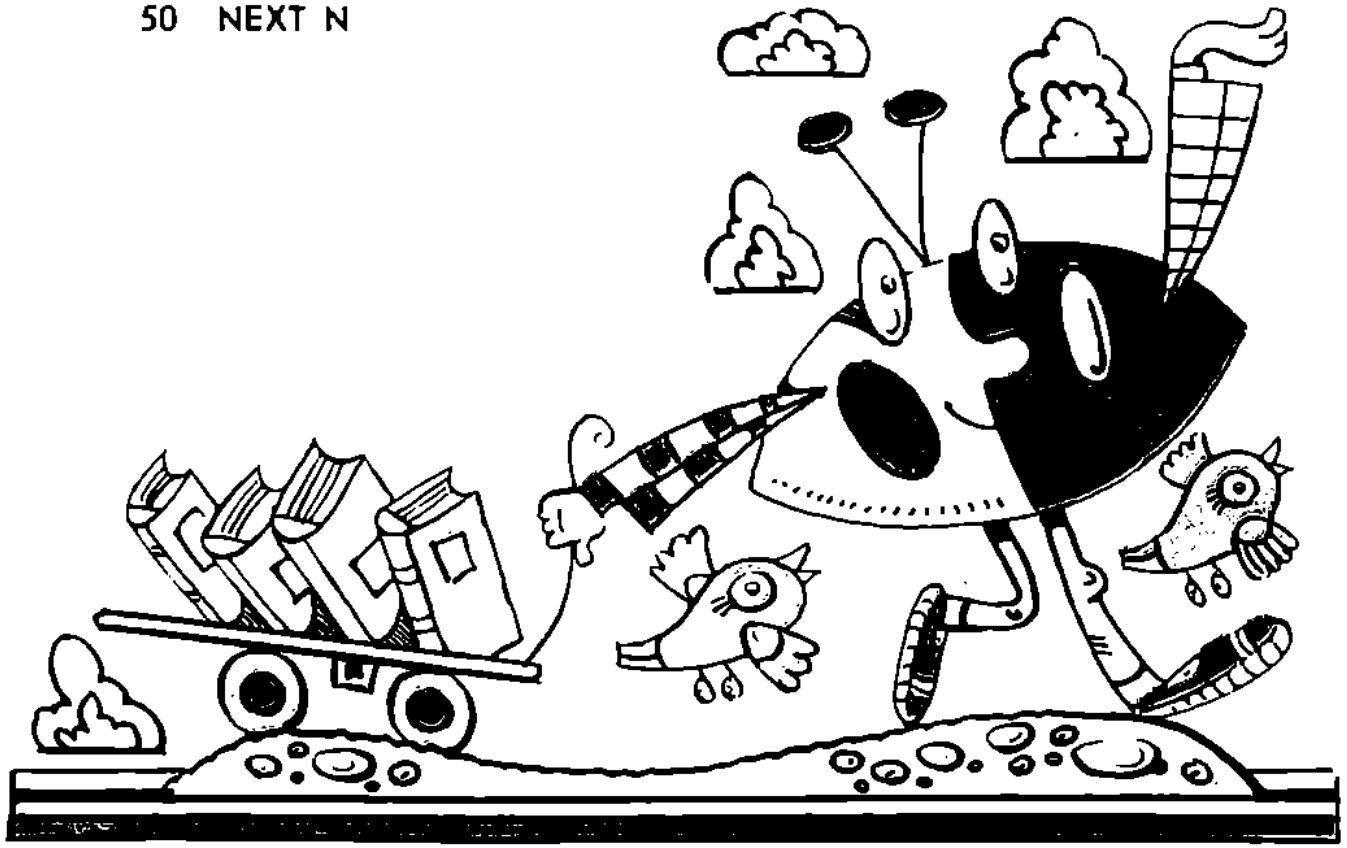
```
20 DIM L$ (6, 13)
```

С помощью этой строки мы сообщаем компьютеру, что необходимо в памяти резервнровать место для 6 переменных с общим нменем L$ и что каждая переменная может нметь макснмально 13 разрядов.

● Когда ребенок полностью ознакомится с оператором DIM н свободно его будет использовать, он сможет расширять содержание своих программ, комбинируя этот оператор с другими, напрнмер, с INPUT.

Так, некоторые строки программы могут служить для ввода с клавиатуры числа элементов каждого списка:

```
10  INPUT "СКОЛЬКО ГРУПП КНИГ ТЫ ИМЕЕШЬ?"; A
20  DIM L$ (A)
30  FOR N = 1 TO A
40  INPUT L$ (N)
50  NEXT N
```



● Оператор DIM часто нспользуется для организацин таблиц. Хотя объяснение этого не входит в цели этой книги, мы приведем маленькнй прнмер. Бейсик позволяет нспользование переменных с двумя и большим количеством нндексов.

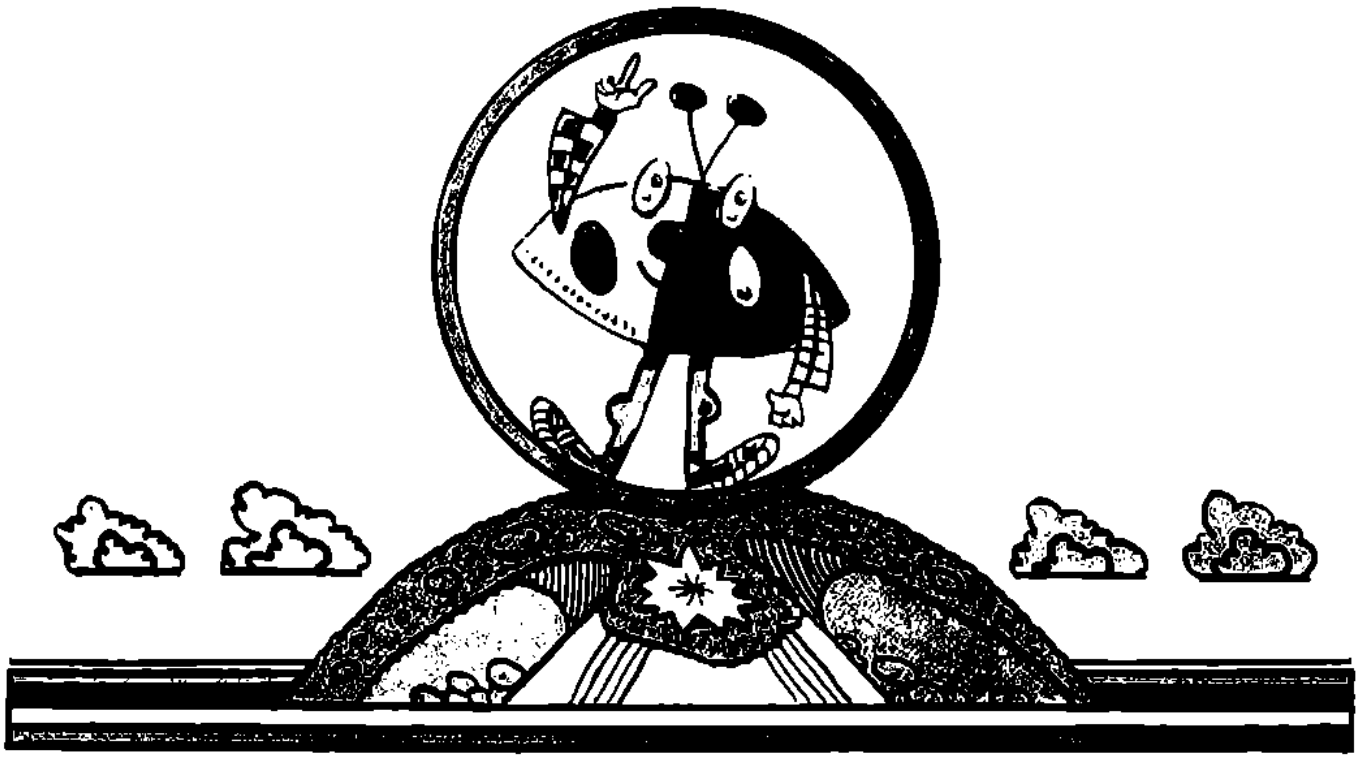
Так, программа

```
10  DIM A (2, 4)
20  FOR X = 1 TO 2
30  FOR Y = 1 TO 4
40  PRINT " ЗНАЧЕНИЕ "; X; ","; Y; " МАТРИЦЫ A"; " ";
50  INPUT A(X,Y)
60  PRINT A(X,Y)
70  NEXT Y
80  NEXT X
```

определяет все значения таблицы, похожей на следующую:

	Столбец 1	*Столбец* 2	*Столбец* 3	*Столбец* 4
Строка 1	A (1, 1)	A (1, 2)	A (1, 3)	A (1, 4)
Строка 2	A (2, 1)	A (2, 2)	A (2, 3)	A (2, 4)

Таким образом, элемент A (2, 3) находится на пересечении 2-й строки и 3-го столбца.

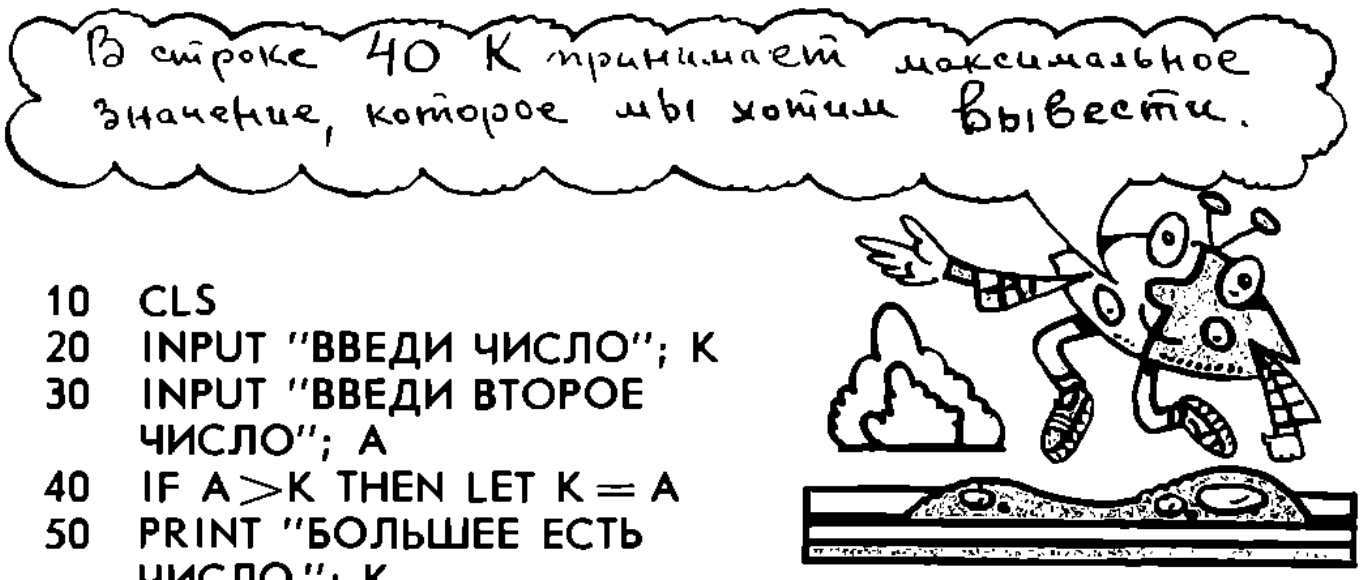


УПОРЯДОЧЕНИЕ

Одно из преимуществ, которым обладает компьютер,— то, что он может помочь тебе сделать работу, повторяющуюся много раз. Как ты знаешь, компьютер быстро вычисляет. На простую работу, занимающую у тебя много времени, компьютер потратит несколько секунд.

Мы в течение дня много раз что-нибудь упорядочиваем: этикетки, игрушки. Мы можем упорядочить имена наших друзей по алфавиту в нашем дневнике. И твой компьютер может выполнить это для тебя. Нужно только показать ему, как это делать.

■ Давай посмотрим, как это делается. Сначала потренируем компьютер, чтобы он сравнивал числа. Напиши программу сравнения двух чисел.




```
10  CLS
20  INPUT "ВВЕДИ ЧИСЛО"; K
30  INPUT "ВВЕДИ ВТОРОЕ
    ЧИСЛО"; A
40  IF A>K THEN LET K = A
50  PRINT "БОЛЬШЕЕ ЕСТЬ
    ЧИСЛО"; K
```

В строках 20 и 30 компьютер ждет ввода значений чисел. В строке 40 компьютер сравнивает их. Скажем компьютеру, что если значение переменной А больше, чем значение переменной К, то переменная К принимает значение переменной А (которое больше) и выводится.

Например, К = 5 и А = 9.

Так как А больше К (9 больше 5), то К = А (К принимает значение 9) и выводится.

Попробуй и ты, меняя значения А и К.

УПОРЯДОЧЕНИЕ

Если мы хотим, чтобы компьютер сравнивал несколько чисел, а потом вывел нам наибольшее из них, можем использовать цикл. Добавь строки:

```
24  INPUT "СКОЛЬКО ЧИСЕЛ ХОЧЕШЬ СРАВНИТЬ?"; N
25  FOR I=1 TO N
45  NEXT I
```

Теперь ты можешь сравнивать до N чисел. Компьютер в конце выведет тебе наибольшее число.

■ Предположим, что мы хотим, чтобы компьютер сравнил и распечатал все числа, которые мы ввели, по порядку (от меньшего к большему).

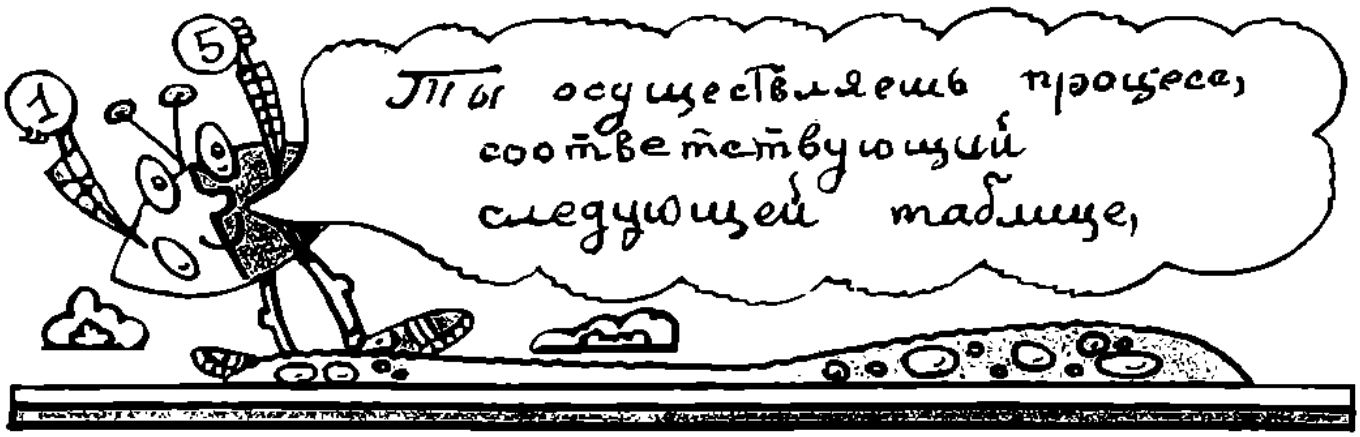
В предыдущей программе компьютер только хранит в памяти значение большего из двух сравниваемых чисел. Это как будто ты имеешь в руках фишку с числом. Тебе дают фишки с числами, которые ты сравниваешь, и выбираешь для сохранения только фишку с большим числом, а остальные отбрасываешь.

Но если ты хочешь упорядочить фишки в порядке возрастания чисел, то тебе надо их сохранить, классифицировать и упорядочить. Поэтому ты не можешь отбросить ни одной фишки или тебе не хватит чисел.

По такому принципу выполняет это и компьютер. Рассмотрим пример. Предположим, необходимо упорядочить в порядке возрастания числа, которые даны нам в таком порядке: 9, 3 и 5.

Ты со своими фишками поступил бы так: сравнил бы первую фишку со второй, выбрал бы бóльшую, упорядочил бы их (поставил бы в порядке возрастания), потом взял бы ту, которая теперь стоит на первом месте, сравнил бы ее с третьей фишкой и упорядочил бы их. Затем сравнил бы фишки, стоящие на втором и третьем местах, и поставил бы их на свои места.

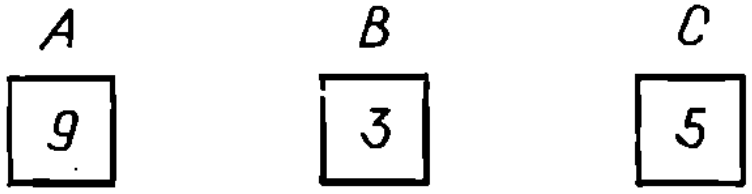
УПОРЯДОЧЕНИЕ



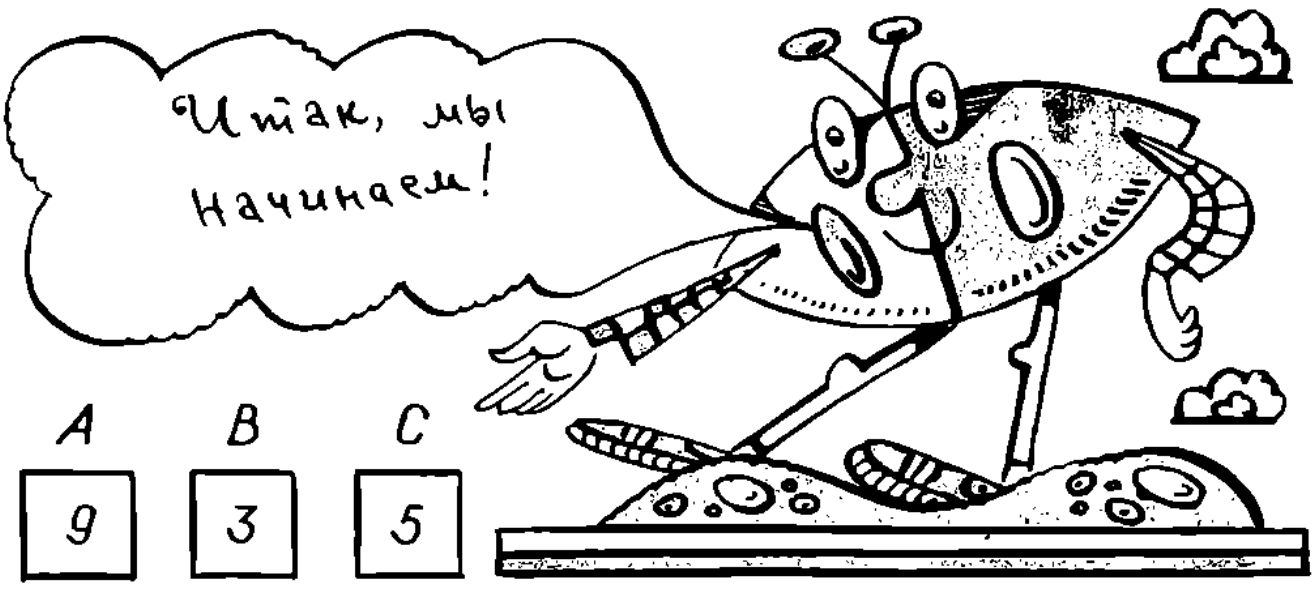
Шаги	Порядок
Сначала имеем	9—3—5
1. Сравниваем 9 и 3; так как 3 меньше, то меняем их местами	3—9—5
2. Сравниваем 3 и 5; так как 3 меньше, то оставляем как есть	3—9—5
3. Сравниваем 9 и 5; так как 5 меньше, меняем их местами	3—5—9

Упорядочение выполнено. Процесс одинаков, как для 3, так и для 1000 чисел или любого другого количества чисел.

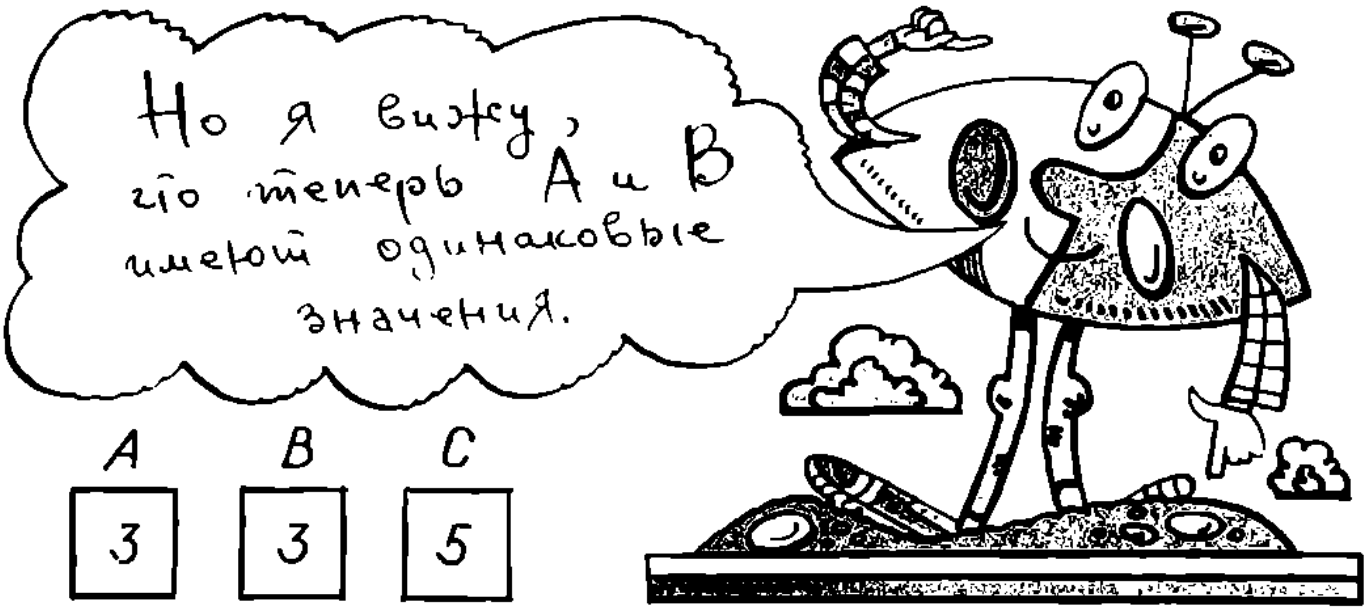
Теперь посмотри, как это выполняется компьютером. Прежде всего присвоим эти значения некоторым переменным:

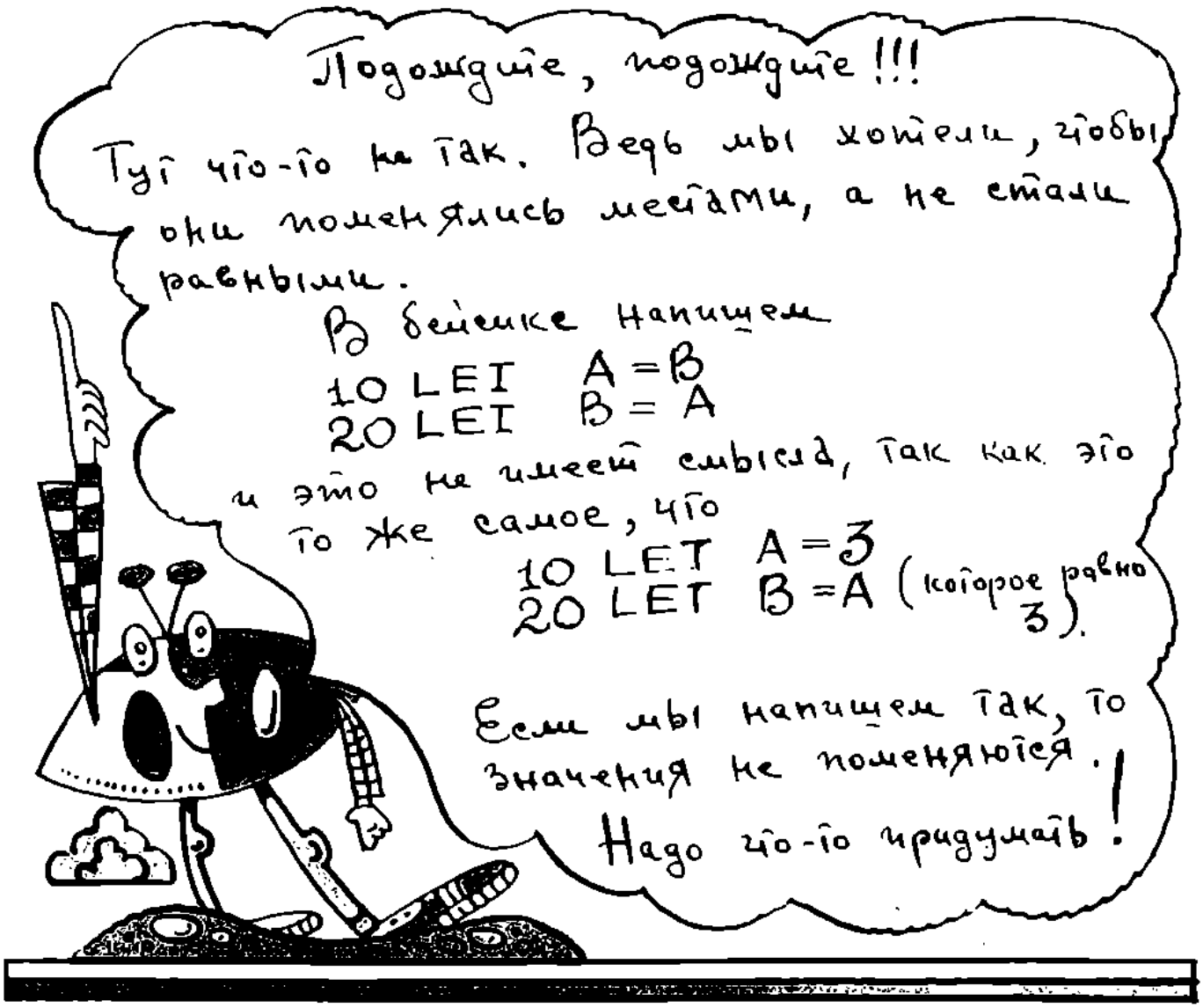


Когда работаем с компьютером, то для получения конечного результата должны дать приказ типа PRINT A, B, C. Но внимание! Если компьютер теперь выведет значения переменных, то получим 9—3—5. Таким образом, нам необходимо указать, что переменной A присваиваем наименьшее значение, переменной B — среднее значение, а C — наибольшее, то есть компьютеру придется поменять числа местами. Он так и делает это следующим образом:



Первыми сравниваем A и B. Так как 3 меньше, чем 9, то присваиваем A значение 3, то есть A берет значение B (A=B).

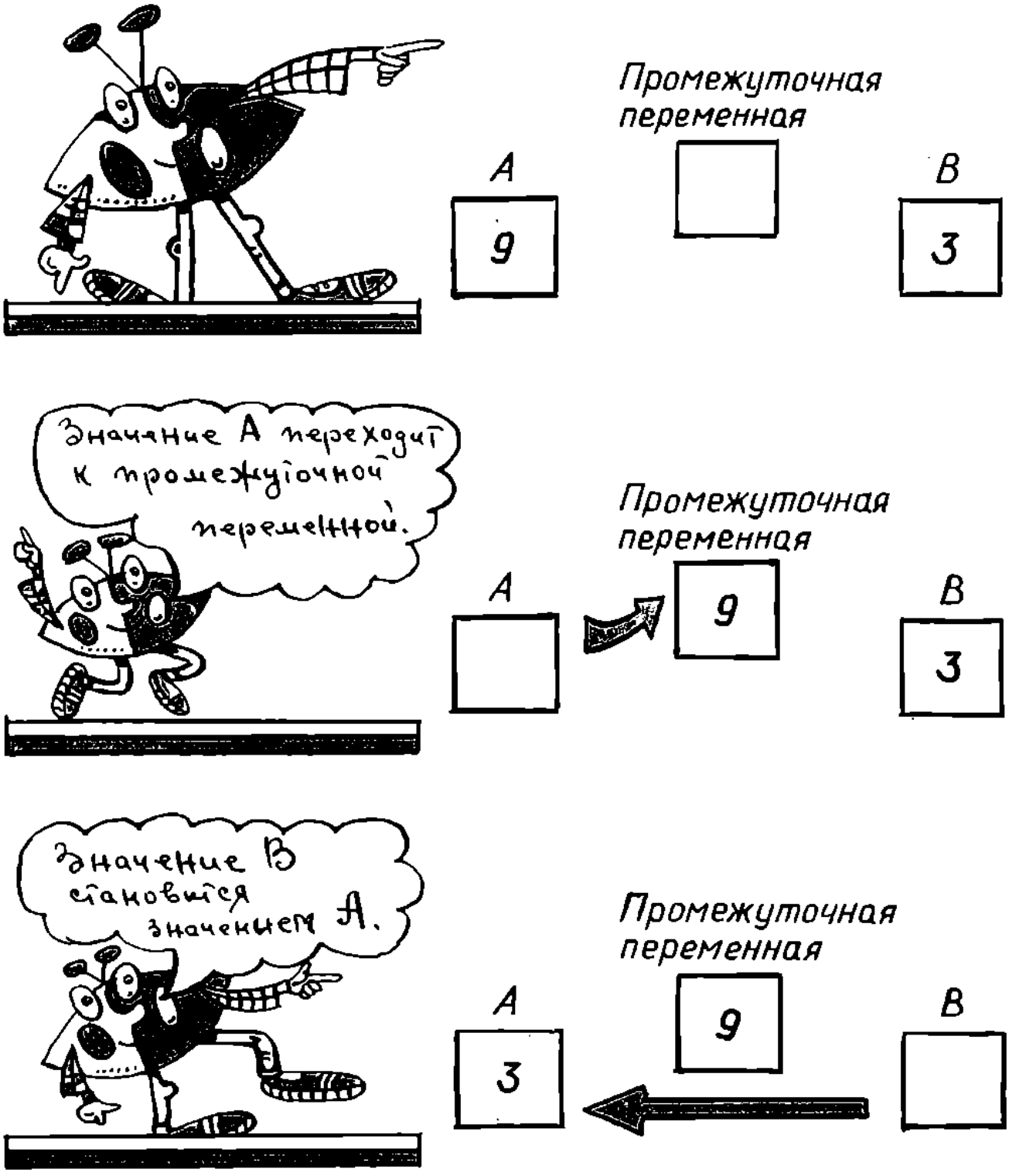

Но я вижу, что теперь A и B имеют одинаковые значения.
A
B
C
3
3
5


Подождите, подождите!!!
Тут что-то не так. Ведь мы хотели, чтобы они поменялись местами, а не стали равными.
В бейсике напишем
10 LET A=B
20 LET B=A
и это не имеет смысла, так как это то же самое, что
10 LET A=3
20 LET B=A (которое равно 3).
Если мы напишем так, то значения не поменяются.
Надо что-то придумать!

УПОРЯДОЧЕНИЕ

Артуро прав. Необходимо предупредить компьютер, что переменная А не должна терять своего значения, принимая значение В. Есть один способ, чтобы сделать это: использовать дополнительную промежуточную переменную, которая может сохранить это значение.

Это компьютер выполняет так:

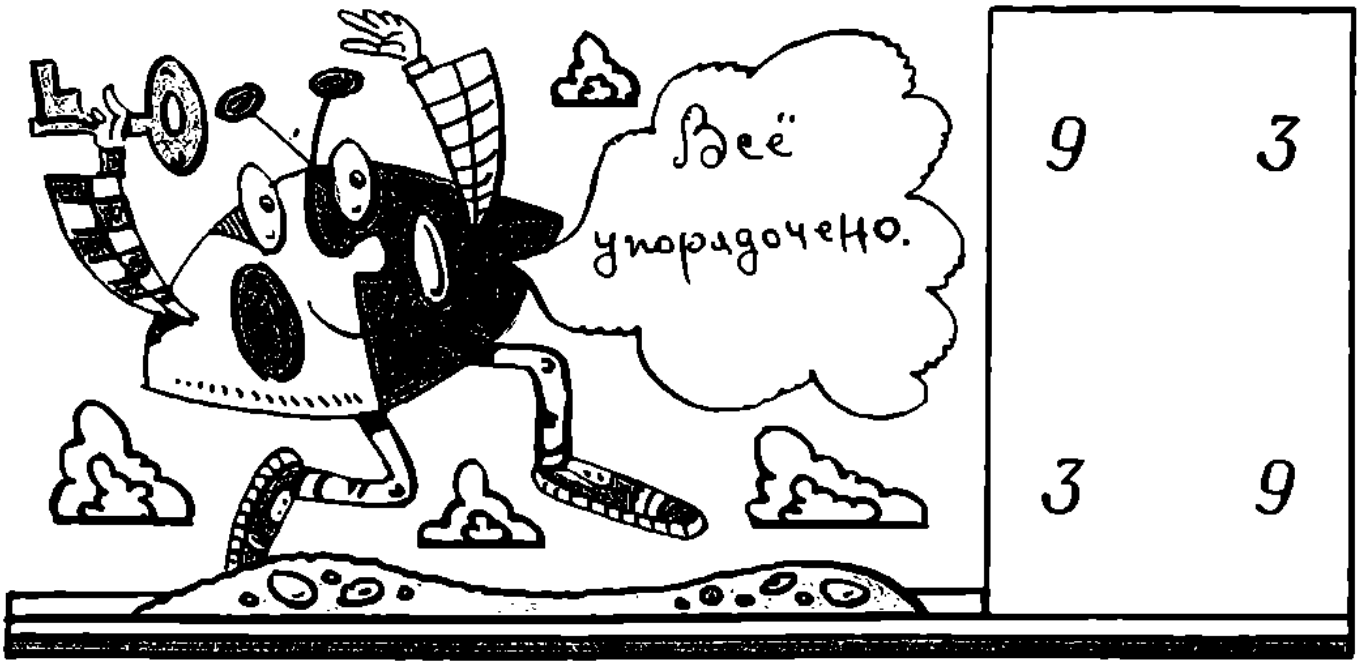


На бейсике предыдущий процесс будет выглядеть так:

```
10  LET A = 9
20  LET B = 3
30  PRINT A; B
40  LET E = A
50  LET A = B
60  LET B = E
70  PRINT A; B
```

В этом виде значения A и B меняются местами без потерь любого из них.

Это ты можешь легко проверить. Если выполнишь программу, то увидишь, что компьютер выведет:

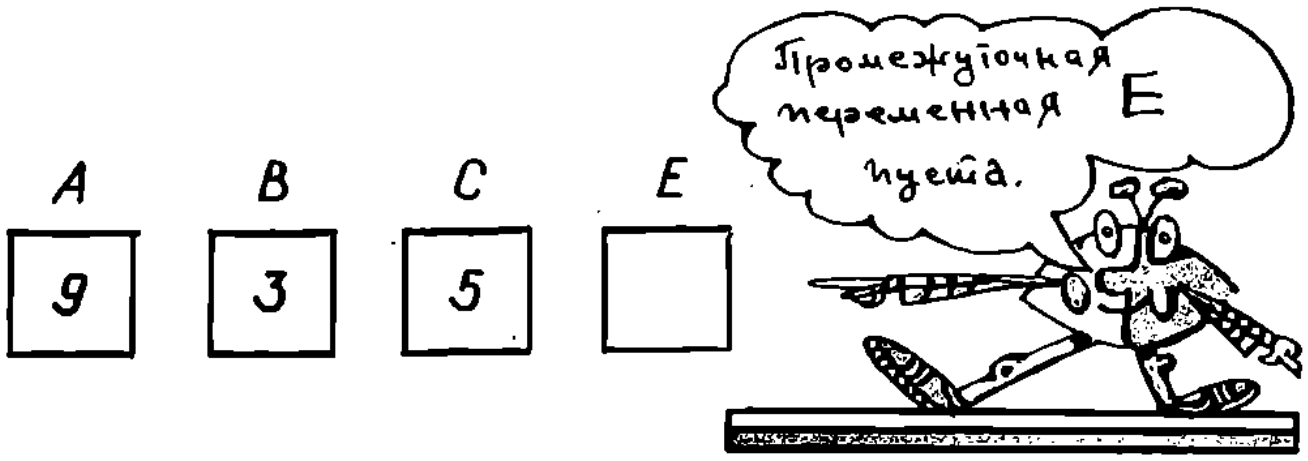


С помощью строк 40, 50, 60 меняются значениями переменные A и B. Этот прием очень важен, так как является ключом для понимания всех программ, которые будем рассматривать дальше. Все эти строки можно объединить в одну:

```
40  LET E = A : LET A = B : LET B = E
```

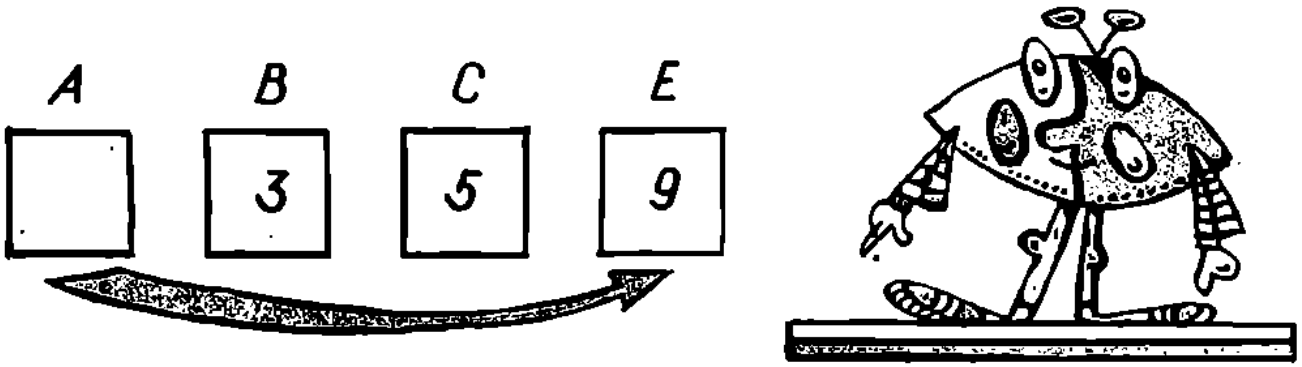
Продолжим анализировать процесс, который прервали на странице 171, когда Артуро предупредил нас об ошибке. Давайте начнем снова, используя одну промежуточную переменную (E).

Компьютер будет работать так:

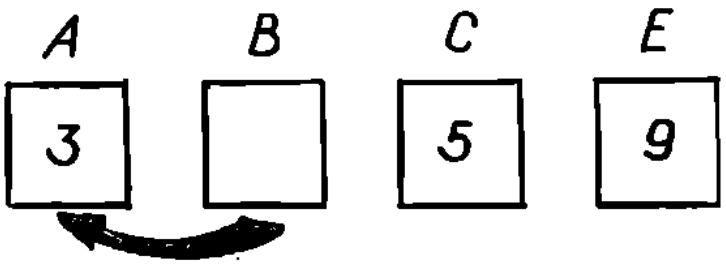


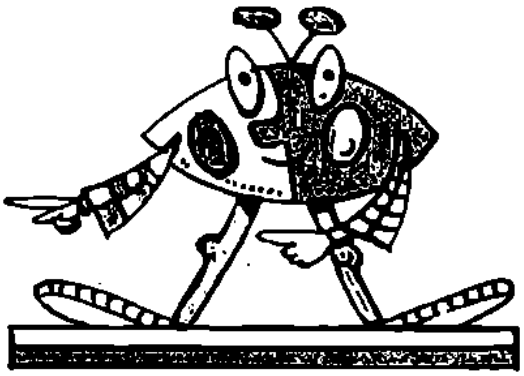
1-й шаг

Возьми A (зиачение 9) и сравни с B (значение 3). Так как B меньше, то поменяем его с A. Следуя действиям, которые мы рассмотрели раньше, перешлем значение A (равное 9) в ячейку для вспомогательной переменной E.

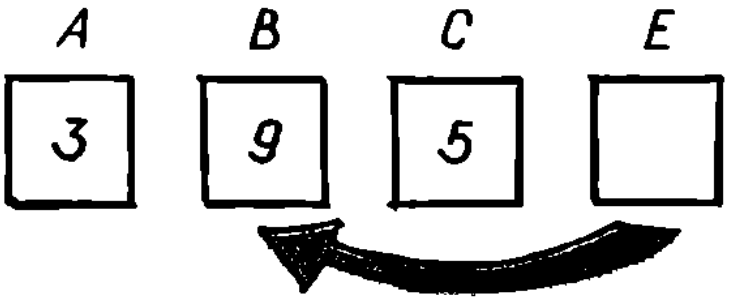


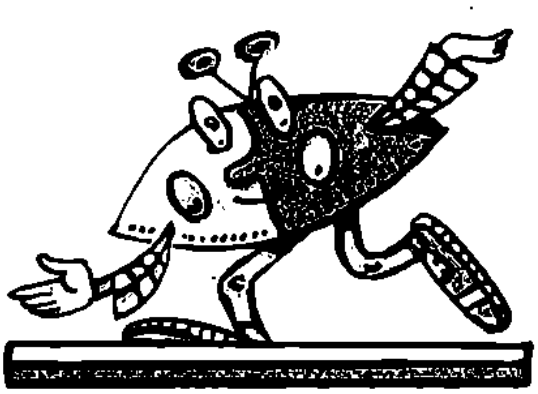
Теперь возьмем значение B (равно 3) и направим его в ячейку для A (которая сейчас пустует).





Затем значение внешней переменной (равно 9) направляем в B.

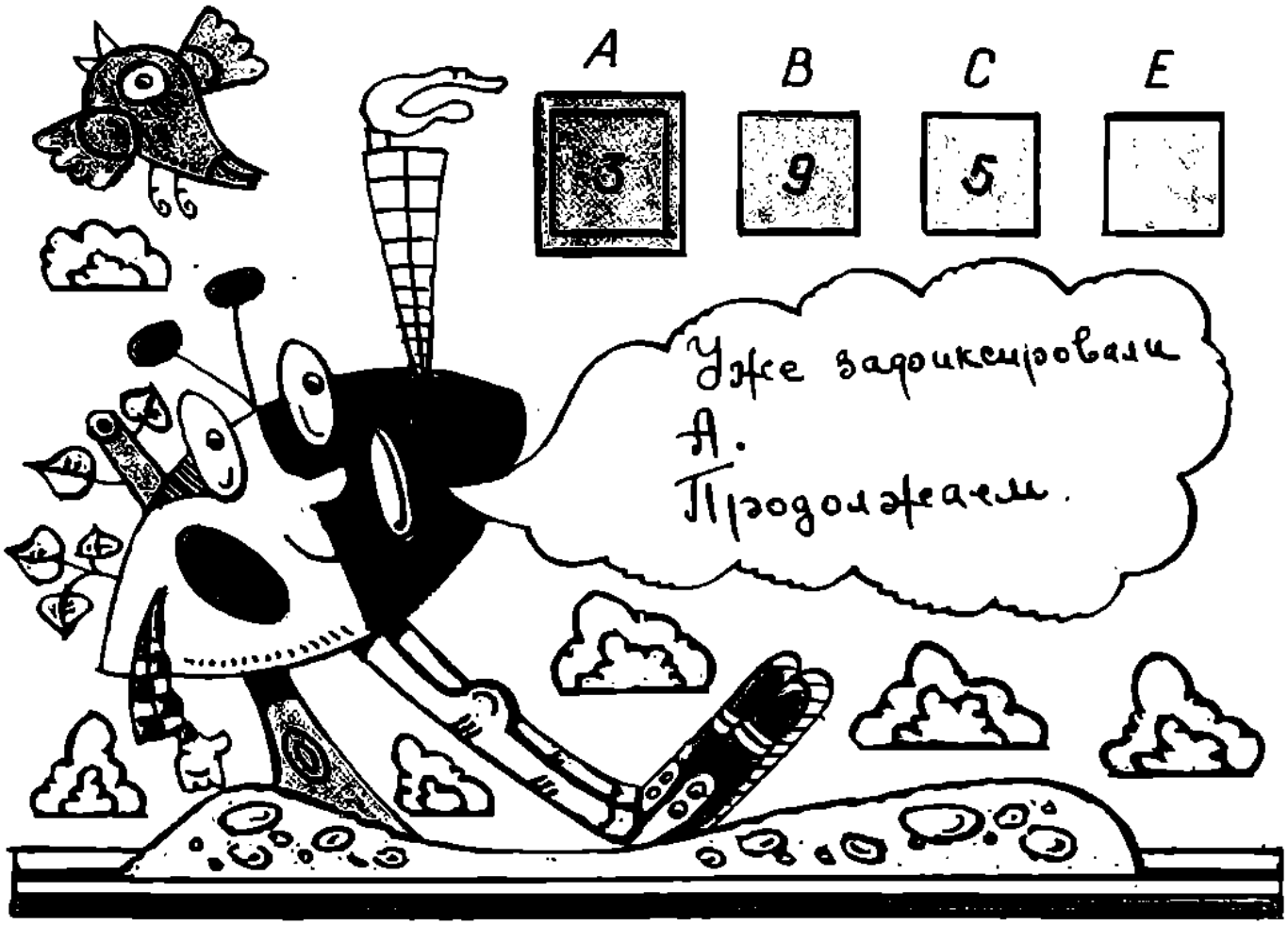




Закончив этот шаг, компьютер сравнил первый и второй элементы и поставил меньший из них на место A.

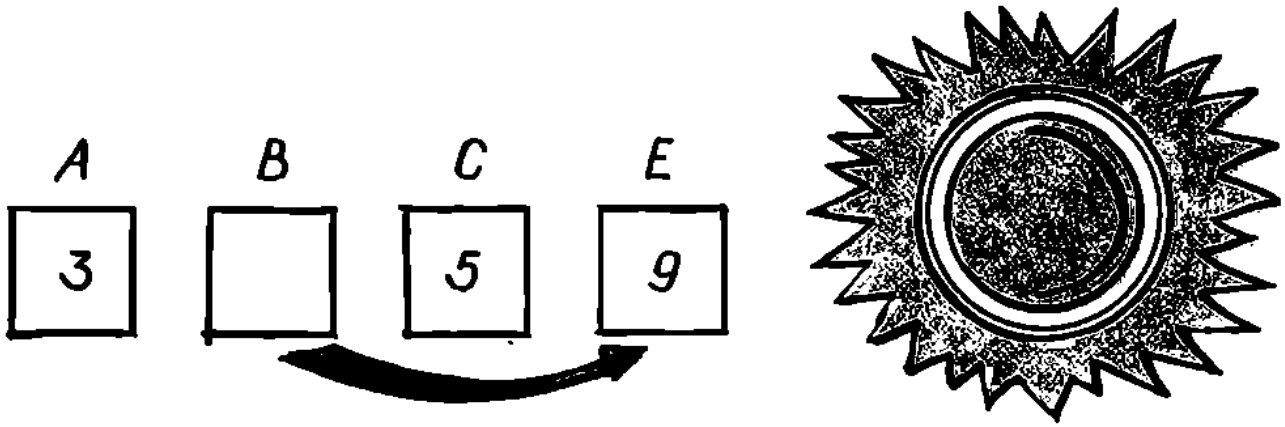
2-й шаг

Возьми A (которое теперь равно 3) и сравни с C (равно 5). Так как A меньше C, то менять их местами не следует. Получаем что 3 — наименьшее число, потому что мы сравнили его с остальными двумя. Переменная A теперь имеет наименьшее значение.

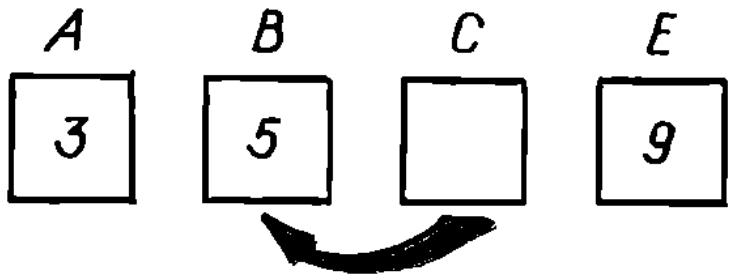


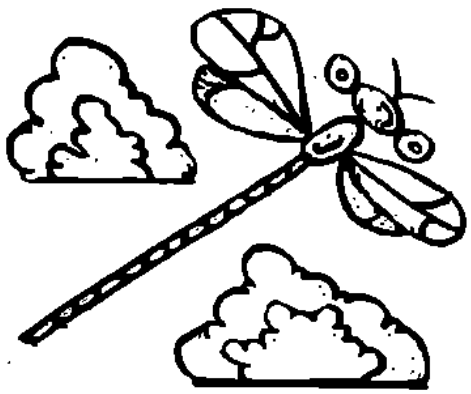
3-й шаг

Возьми B (теперь равно 9) и сравни с C (равно 5). Так как C меньше, чем B, то меняем их местами. Присваиваем значение B (равно 9) вспомогательной переменной E.

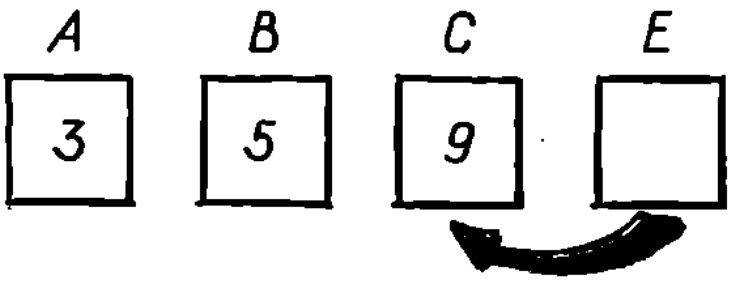


Теперь бери значение C (равно 5) и перенеси его в ячейку переменной B (которая сейчас пустует).





Затем значение вспомогательной переменной E (равно 9) передаем в C.



И теперь все три числа упорядочены по величине от меньшего к большему.

Весь этот процесс, описанный в виде программы, выглядит так:

```
10  INPUT A, B, C
20  IF A>B THEN LET E=A:LET A=B:LET B=E
30  PRINT A; " "; B; " "; C
40  IF A>C THEN LET E=A:LET A=C:LET C=E
50  PRINT A; " "; B; " "; C
60  IF B>C THEN LET E=B:LET B=C:LET C=E
70  PRINT A; " "; B; " "; C
80  END
```

УПОРЯДОЧЕНИЕ

Ты думаешь, что эта программа великовата для упорядочения всего трех чисел? Если нужно упорядочить 100 чисел, то удобнее сделать это вручную? Здесь нужно подумать...

Точно! С DIM все можно легко сделать. Ты уже знаешь, как его использовать, и потому будем понемногу писать программу.

Сначала напишем:

```
10  CLS
20  INPUT "СКОЛЬКО ЧИСЕЛ ХОЧЕШЬ УПОРЯДОЧИТЬ?"; N
30  DIM A(N)
40  FOR I = 1 TO N
50  INPUT "ВВЕДИ ЧИСЛО И НАЖМИ ENTER";A(I)
60  PRINT A(I); "  ";
70  NEXT I
75  PRINT : PRINT
```

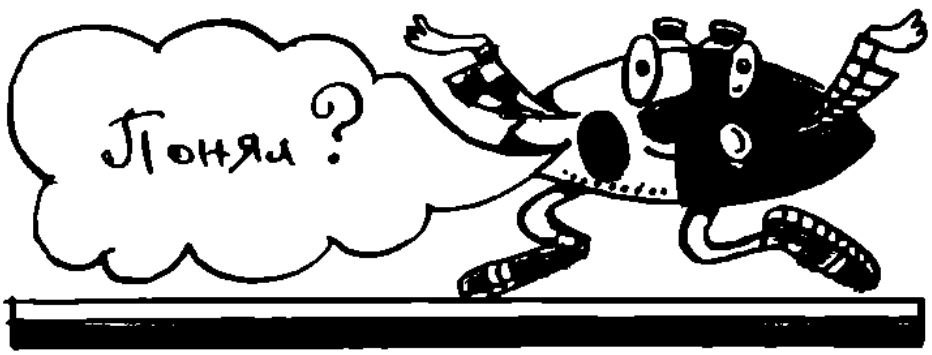
УПОРЯДОЧЕНИЕ

Оператор DIM в строке 30 подготавливает компьютер для резервирования в своей памяти N ячеек по количеству переменных, которые мы будем использовать.

Например, запусти программу на счет и дай N значение 5. Компьютер будет сравнивать 5 чисел. С помощью цикла, начинающегося в строке 40, компьютер получит 5 чисел, которые ты вводишь с помощью клавиатуры и присвоит их переменным A(I). Например, вводим следующую серию чисел: 5—9—2—3—1.

Компьютер присвоит такие значения:

A(1) = 5
A(2) = 9
A(3) = 2
A(4) = 3
A(5) = 1



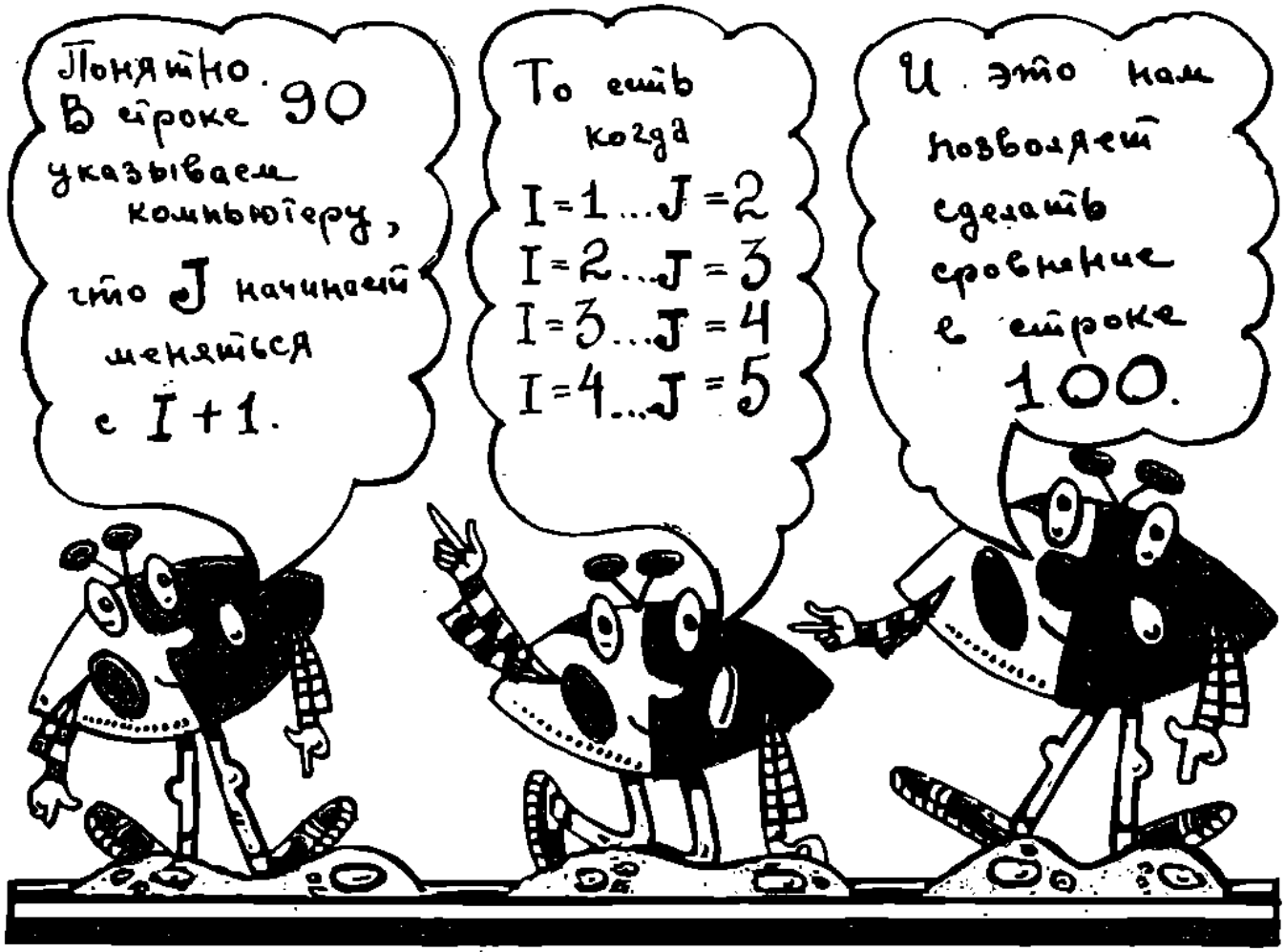
Компьютер выведет все значения на экран.

Продолжаем. Будь внимателен, сейчас пойдут очень важные и красивые строки.

```
 80  FOR I=1 TO N-1
 90  FOR J=I+1 TO N
100  IF A(I)>A(J) THEN LET E=A(I):LET A(I)=A(J):LET
     A(J)=E
110  NEXT J
120  NEXT I
```

Ты уже знаешь, как работает строка 100: здесь компьютер сравнивает и меняет местами значения, используя вспомогательную переменную.

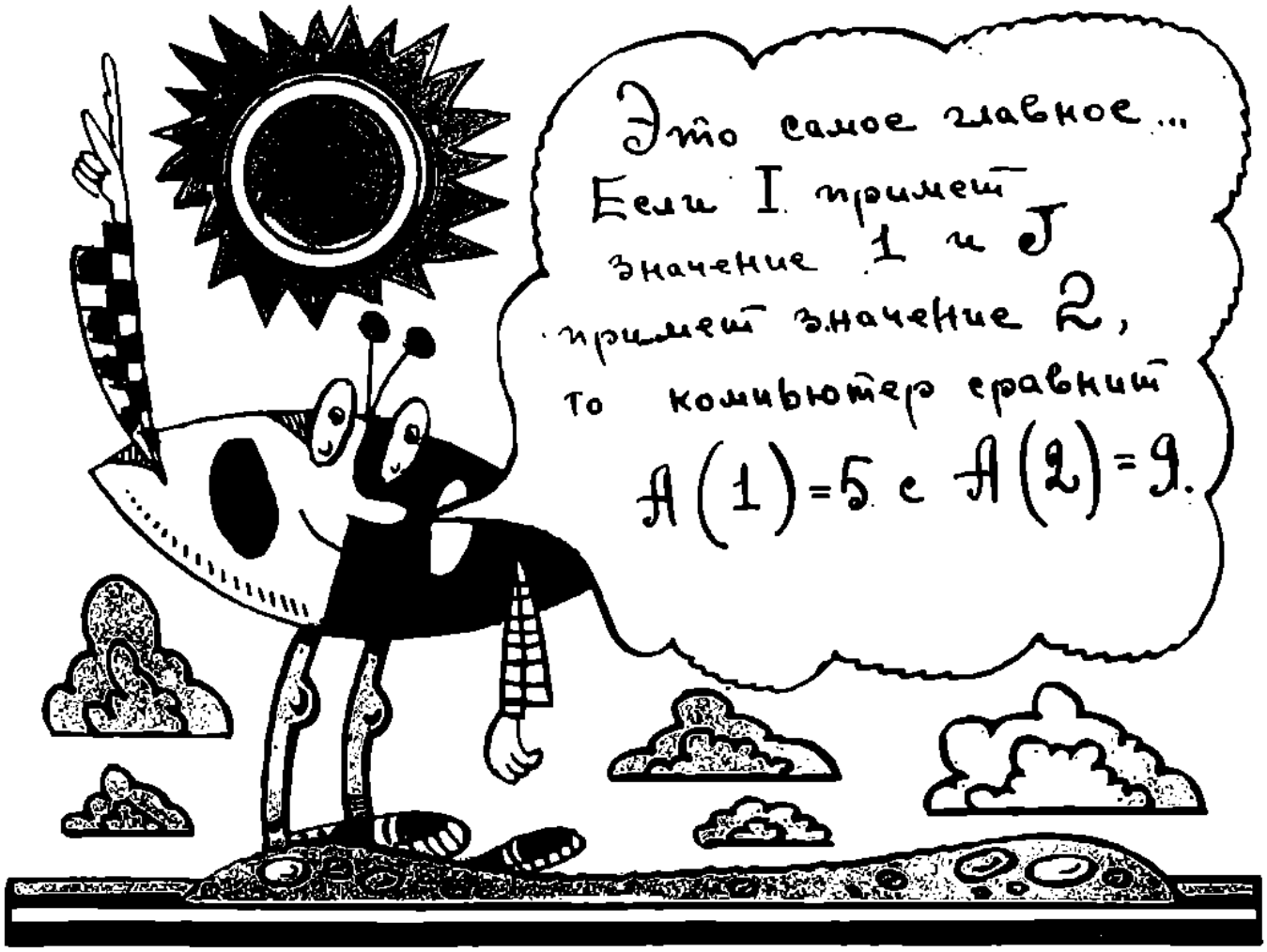
Обрати внимание, что сравниваемые значения — это A(I) и A(J). В строках 80 и 90 начинаем циклы, которые присваивают соответствующие значения A(I) и A(J). Эти циклы очень хорошо продуманы. Посмотри. Цикл, начинающийся в строке 90, находится внутри цикла, начинающегося в строке 80. Так, если I равно 1, то J принимает последовательно значение 2, 3, 4 и 5.



Еще раз посмотри на строку 100. Сначала сравииваем A(1) с A(2), то есть 5 и 9, и, если необходимо, меняем местами их значения. Переходим на строку 110 и берем следующее значение J. Теперь сравниваем A(1) с A(3) и так далее.

Когда J станет равным 5, переходим на строку 120 и берем следующее значение I (равное 2). Теперь сравниваем A(2) с A(3) и т. д.

Внимание! Когда компьютер сравнивает A(1), A(2) и т. д., то на самом деле он сравнивает значения перемеиных A(1), A(2) и т. д., которые они имеют в настоящий момент.



Мы используем I и J для того, чтобы индексировать переменные и сравнивать первый элемент ряда со вторым и всеми остальными, а затем то же самое делаем со вторым и всеми следующими и т. д.

Нам осталось только дописать несколько строк, чтобы получить результат.

```
130  FOR I=1 TO N
140  PRINT A(I); "  ";
150  NEXT I
160  END
```

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Изучаемые в этом пункте вопросы достаточно сложны. Этот пункт нельзя рассматривать, не зная достаточно полно технику использования оператора DIM, описанную в предыдущем пункте.

- Если считаете необходимым, потратьте больше времени на изучение этой темы, разъясняйте ее в несколько приемов, чтобы ребенок постепенно осваивал различные фазы процесса сравнения и упорядочивания элементов.

- Очень важно, чтобы была видна разница между ручным упорядочиванием и компьютерным, в котором требуется использование вспомогательных переменных. Добейтесь того, чтобы ребенок понял метод использования вспомогательной переменной.

- Нужно показать ребенку, что компьютер всегда сравнивает первый элемент некоторой последовательности со всеми остальными, чтобы зафиксировать место его расположения (в нашем случае это наименьший элемент). Сначала мы фиксируем наименьший элемент, затем начинаем аналогичный процесс со следующим элементом и т. д.

- «Гвоздь» программы находится в строках 80—120. Уделите необходимое время на усвоение процесса выполнения двух циклов. Повторите при необходимости операторы FOR/NEXT и DIM из предыдущих пунктов.

- Подчеркиваем, что переменные I и J этих циклов служат исключительно для определения A(1), A(2), A(3) и для сравнения их значений. Однако очень важно заметить, что A(1), A(2) и т. д. имеют каждый свое значение, которое ранее введено с помощью клавиатуры при выполнении программы.

- В нашем примере мы упорядочили числа от меньшего к большему. Для упорядочения от большего к меньшему, достаточно только изменить знак сравнения $>$ (больше) на знак сравнения $<$ (меньше) в строке 100.

INT (немного подробнее)

Вспомни, что при использовании INT получаем целую часть числа. Теперь воспользуемся программой:

```
10  FOR A = 10 TO 100 STEP 10
20  LET B = A / 2.34
30  PRINT B
40  NEXT A
```

Эта программа берет последовательно (число за числом) числа, кратные 10, делит их на 2,34 и печатает результат. При выполнении программы ты сможешь увидеть, что в результате получаем десятичные дроби.

Если используем INT, то мы можем отбросить десятичную часть числа. Добавим в программу строку:

```
25  LET C = INT(B)
```

И изменим 30-ю строку следующим образом:

```
30  PRINT B; "  "; C
```

Выполни эту программу. У тебя остались только целые части результатов. Несомненно, намного лучше округлять с избытком или недостачей результат, чем оставлять только целую часть.

■ Конечно! Представь себе, что данные, которые мы используем, имеют важное значение для вычислений. Допустим, число 29,9145299 — это почти 30, но с INT остается 29.

INT (немного подробнее)

Это очень просто. Нужно только просуммировать с 0.5 число до того, как отбросить десятичную часть.

Например:

```
   42.7350427
    0.5
-------------
   43.2350427
INT(43.2350427)
```

Так как это число близко к 43, суммируем его с 0.5, оно становится больше 43, и с INT остается только 43.

```
   34.1880342
    0.5
-------------
   34.6880342
INT (34.6880342)
```

Поскольку число близко к 34, суммируем его с 0.5, результат не превысит 35, и с INT остается 34.

Измеиим программу согласио этому правилу, добавив:

```
27  LET R = INT(B + 0.5)
 5  PRINT "ДЕСЯТИЧНОЕ ЧИСЛО"; "  "; "ЦЕЛАЯ ЧАСТЬ";
    "  "; "ОКРУГЛЕННОЕ ЧИСЛО"
```

и поменяем строку 30:

```
30  PRINT B; "  "; C; "  "; R
```

При выполнении программы на экране получим три столбца:

Десятичное число	Целая часть	Округленное число
4.27350427	4	4
8.54700855	8	9
. . . . .	. . .	. . .

Посмотри на разницу между столбцами.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Этот пункт не ограничивается описанием функции INT (уже известной вам), но содержит сведения о форме использования INT. Очень важно, чтобы ребенок овладел этим понятием, научнлся применять его и понял, как именно он может использовать ту или иную форму этой функции в соответствии с нашими интересами.

- На самом деле основное выражение LET A = INT(X + 0.5) эквивалентно некой новой функции бейсика, которая автоматически округляет любое число.

- Таким образом, мы можем комбинировать некоторые функции бейсика с простыми математическими выражениями для того, чтобы найти новое применение илн решать конкретные задачи, которые могут возннкнуть в наших программах.

- Так, если нам нужно в какой-то программе определить, четным или нечетным является целое число, то можем использовать формулу

```
10  IF INT(X/2)*2 = X THEN PRINT
    "ЧИСЛО ЧЕТНОЕ"
```

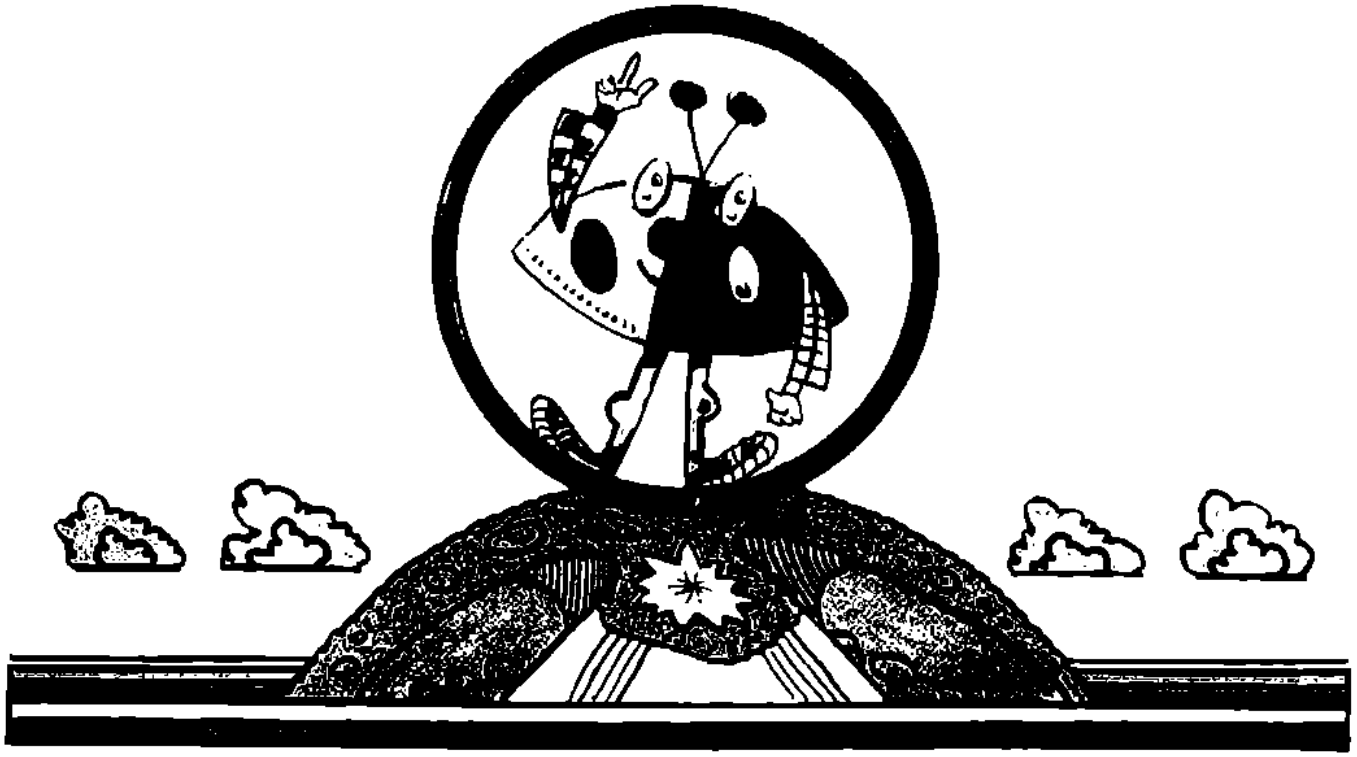
Формула очевидна и не требует поясненнй.

- Кроме INT и RND существуют другие числовые функции, которые пока не нспользуются детьми на этом уровне изучеиия бейсика. Приведем эти фуикции.

INT (немного подробнее)

SGN(N)	Показывает знак числа N	SGN(3) — на экране получим 1 SGN(—2) — на экране получим — 1 SGN(0) — на экране получим 0
ABS(X)	Вычисляет абсолютное значение числа X	ABS(—7) — на экране получим 7 ABS(7) — на экране получим 7
SQR(N)	Находит квадратный корень числа N	SQR(9) — на экране получим 3

- В бейсике используются также другие стандартные функции — логарифмические, экспоненциальные и тригонометрические, которые облегчают выполнение трудоемких вычислений. Но мы в этой книге о них говорить не будем.

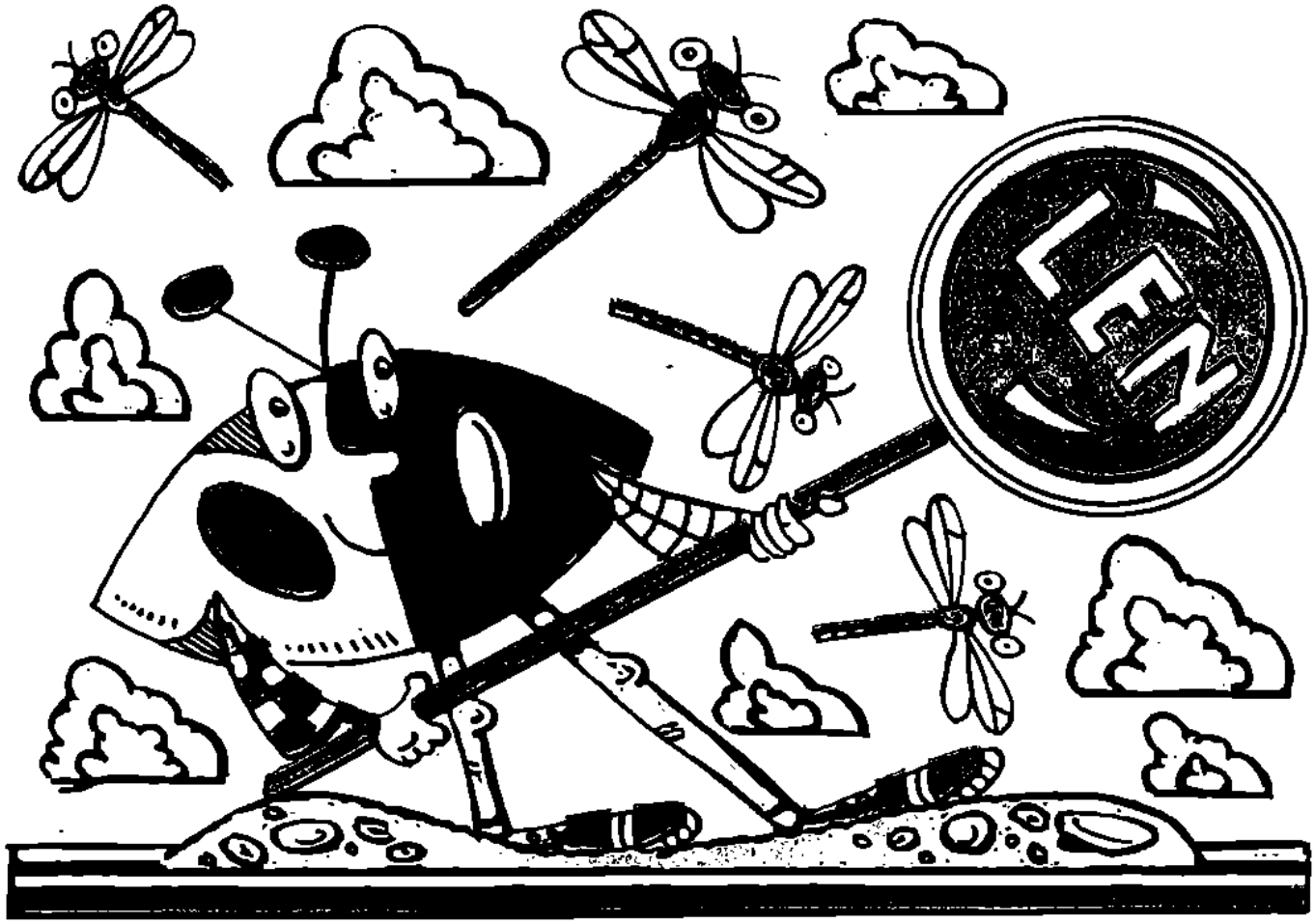


СТРОКИ СИМВОЛОВ

Кроме операторов (PRINT, IF, GOTO, LET, FOR/NEXT, READ...) мы узнали также числовые функции INT и RND.

Теперь давай изучим некоторые строковые функции, которые работают с текстами. Они очень полезны, так как с ними бейсик тебе позволит работать со словами, фразами и т. д.

Мы не будем их все объяснять и не будем углубляться в их изучение. Главное, чтобы ты знал, как они функционируют и для чего служат.



LEN сообщает иам о числе символов, которые имеет строка символов (знаков).

Например:

```
10  LET A$ = "БЕЙСИК ДЛЯ ДЕТЕЙ"
20  PRINT LEN(A$)
```

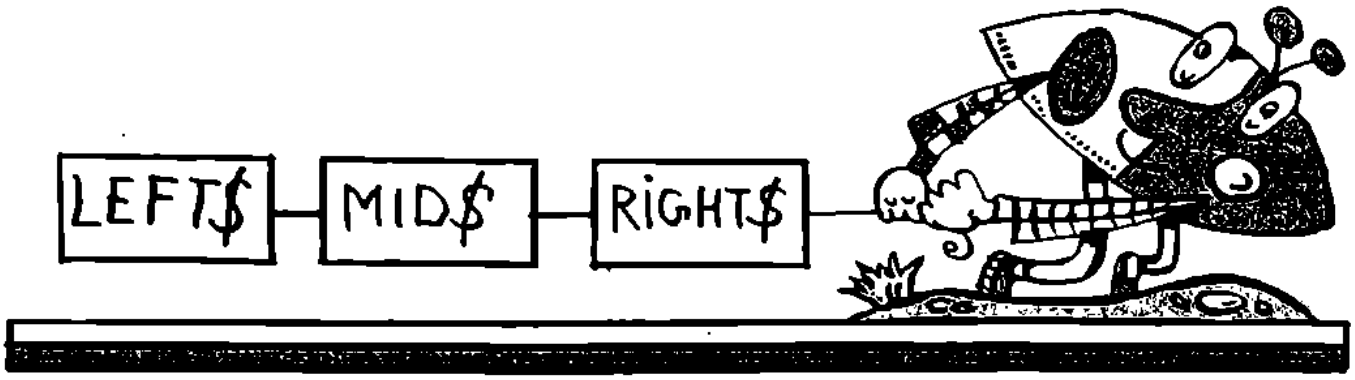
При выполнении программы на экране появится число 16. LEN подсчитает все символы в символьной (литерной) переменной A$, включая и пробелы.

LEN может быть очень полезной, если ты хочешь подчеркнуть какое-нибудь слово, длину которого ты предварительно не знаешь.

Например, введи программу:

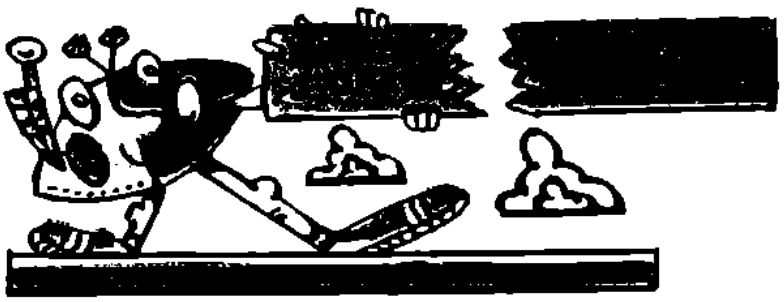
```
10  INPUT "НАПИШИ СВОЕ ИМЯ"; A$
20  PRINT A$
30  FOR N = 1 TO LEN(A$)
40  PRINT "—";
50  NEXT N
```

Все буквы будут подчеркнуты.



Эти функции нам позволят выделить нужное количество символов в символьной переменной. Например, LEFT$ отделяет нужное количество символов в символьной переменной, начиная слева. RIGHT$ делает то же самое, но начинает считать справа. Посмотрим один пример.

Введи:



```
10  LET A$ = "АРТУРО"
20  PRINT LEFT$(A$,2)
```

Выполни программу и получишь на экране АР. Добавь 30-ю строку:

```
30  PRINT RIGHT$(A$,3)
```

На экране появится УРО.

Приведем еще один пример для глаголов на испанском языке.

```
10  INPUT "НАПИШИ КАКОЙ-НИБУДЬ ГЛАГОЛ В ИНФИНИ-
    ТИВЕ, И Я СКАЖУ, К КАКОМУ СПРЯЖЕНИЮ ОН ОТНО-
    СИТСЯ"; A$
20  LET B$ = RIGHT$(A$,2)
30  IF B$ = "AR" THEN PRINT "ПЕРВОЕ"
40  IF B$ = "ER" THEN PRINT "ВТОРОЕ"
50  IF B$ = "IR" THEN PRINT "ТРЕТЬЕ"
```

Обрати внимание, что в строке 20 присваиваем переменной B$ значение двух последних символов переменной A$. В информатике мы говорим, что B$ является подстрокой A$.

MID$ — это функция более совершенная, чем предыдущая. Посмотри пример:

```
10  LET A$ = "АРТУРО"
20  PRINT MID$(A$, 2, 3)
```

Выполни программу, и твой компьютер выведет РТУ.

Ты уже знаешь, как действуют LEN, LEFT$, RIGHT$ и MID$. Теперь создадим программу, использующую две из этих функций.

Введи:

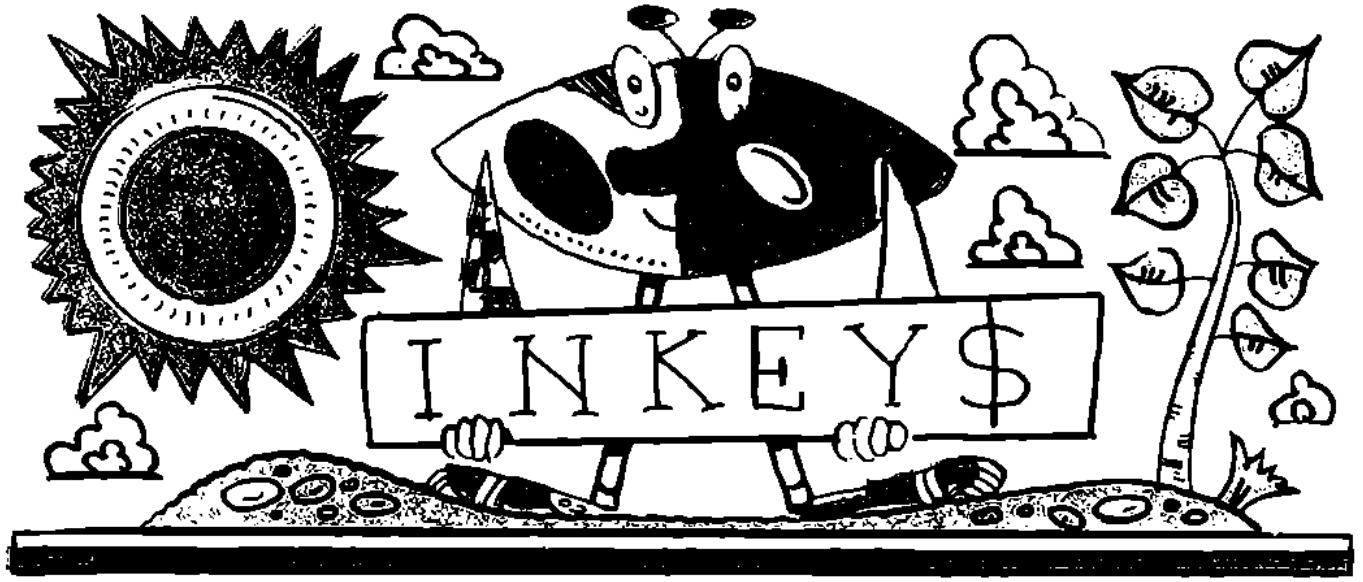
```
10  INPUT "НАПИШИ ДЛИННЫЙ ТЕКСТ";A$
20  CLS
30  LET L = LEN(A$)
40  FOR N = 1 TO L
50  PRINT MID$(A$,N,1);
60  FOR X = 1 TO 50 : NEXT X
70  NEXT N
```

В строке 30 переменная L получит числовое значение, равное количеству символов в твоем тексте, включая пробелы.

В строке 40 организуется цикл, в котором N принимает значения от 1 до числа символов в тексте.

В строке 50 компьютер берет один за другим символы из текста, начиная с первого (значение N начинается со значения 1), и печатает один за другим.

Скорость появления текста на экране регулируется в строке 60.



Эта функция позволит нам присвоить значение символьной переменной с помощью клавиатуры без нажатия клавиш ENTER или RETURN после ввода.

Например, напишем:

```
 5 PRINT "НАЖМИ ЛЮБУЮ КЛАВИШУ"
10 LET A$ = INKEY$: IF A$ = "" THEN 10
```

В этой строке компьютер ждет значение, которое ты ему дашь с помощью клавиатуры и которое затем присваивается переменной A$.

Второй оператор строки 10 сообщает компьютеру, что если A$ не имеет никакого значения (то есть ты не нажимаешь никакой клавиши), он должен ждать в этой строке. Продолжим:

```
20 PRINT "ТЫ НАЖАЛ КЛАВИШУ..."; A$
30 GOTO 5
```

Выполни программу и попробуй нажимать разные клавиши. Ты увидишь, что нет необходимости нажимать ENTER для того, чтобы ввести данные в программу.

Прием, используемый в строке 10, очень удобен для остановки программы. Она будет ждать, пока ты не нажмешь какую-нибудь клавишу.

■ Помнишь программу «Шахматы» на странице 109—110? В строках 10, 90 и 100 мы сделали так, что компьютер выводит на экран количество зерен для каждой новой клетки всегда, когда ты нажимаешь S, и потом ENTER (ВК или ↵).

С INKEY$ мы можем сделать программу более удобную. Изменим строки 10 и 80 на:

```
10 PRINT "НАЖМИ ЛЮБУЮ КЛАВИШУ ДЛЯ ДРУГОЙ
   КЛЕТКИ"
80 LET A$ = INKEY$: IF A$ = "" THEN 80
```

Уберите строки 90 и 100 из старой программы. Результат будет такой же, но теперь для работы программа более удобна.

Ты можешь сделать то же самое с программой на странице 133, которая дает нам таблицу умножения (измени только строку 90).

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Выполните все примеры вместе с ребенком. В принципе здесь не должно встретиться нникаких трудностей в понимании программ и использовании указанных функций.

- Некоторые компьютеры (SINCLAIR) используют другие выражения, соответствующие функциям LEFT$, RIGHT$ и MID$. Так, при работе с A$ = "АРТУРО":

```
PRINT  A$(2 TO 4) напишет РТУР,
PRINT  A$(3)      напишет Т,
PRINT  A$(TO 4)   напишет АРТУ,
PRINT  A$(3 TO)   напишет ТУРО.
```

В строке 20 программы на странице 191 можно написать:

```
20  LET B$ = A$(LEN(A$) — 1 TO),
```

которая присваивает точно так же символьной переменной B$ два последних знака переменной A$.
Аналогично строка 50 со страницы 192 запишется:

```
50  PRINT A$ (N);
```

Учтите также, что в зависимости от типа компьютера может варьироваться расположение скобок у функции.

- Существуют еще и другие строковые функции (их детальное рассмотрение не входит в наши цели):

VAL(A$) — переводит цифровую строку A$ в цифровое значение.

Например:

```
10  LET A$ = "234"
20  LET A = VAL(A$)
30  PRINT A
```

На экране появится 234, но не как строка символов, а как число.

STR$(A) — переводит цифровое зиачение в цифровую символьную строку. Позволяет использовать при работе с числами преимущества обращеиия с ними как с текстами посредством функций обработки строк.

Например:

```
10  LET A = 234
20  LET A$ = STR$(A)
30  PRINT A$
```

На экране появится 234, но уже не как число, а как строка знаков. Испытай ее, добавив строку 40:

```
40  PRINT RIGHT$(A$, 2)
```

Компьютер выведет:

```
234
 34
```

● В некоторых компьютерах функция INKEY$ замещается эквивалентиой функцией GET. Хотя эта функция не есть строковой, она применяется в основном для обработки строк. Так, GET позволяет ввести цифровую перемениую, а GET$ символьную строку. Очевидно, более часто используется последняя форма, так как в этом случае компьютер воспримет любую клавишу, которую нажмешь. И эта функция принимает форму:

```
10  GET A$
```

CHR$ и ASC

Все числа, буквы и другие знаки, имеющиеся на клавиатуре твоего компьютера, соответствуют некоторым числам, которые компьютер запоминает.

Числа, соответствующие символам (имеющимся на клавиатуре), называются *кодами*. Эти коды могут меняться в зависимости от типа компьютера.

Функции CHR$ и ASC позволяют нам увидеть соответствие, которое существует между этими цифровыми кодами и символами, имеющимися на клавиатуре. Например, PRINT ASC("A") нам даст значение кода, который имеет символ "A" в твоем компьютере.

PRINT CHR$(65) нам даст символ, который соответствует значению кода 65 твоего компьютера.

CHR$ и ASC

Иногда требуется знать соответствие между символами и значениями кода. Рассмотрим как пример одну игровую программу.

■ Сначала посмотрим, какие коды имеет твой компьютер.

Введи:

```
10  FOR N = 1 TO 255
20  PRINT CHR$(N), N
30  NEXT N
```

Если ты не успеваешь по времени прочитать все символы, введи один цикл ожидания:

```
25  FOR I = 1 TO 200 : NEXT I
```

Выполни программу. В столбце слева появятся символы, числа, буквы твоей клавиатуры. В столбце справа — соответствующие им коды.

Обрати внимание, например, на коды для букв. Возможно, в твоем компьютере все буквы латинского алфавита соответствуют кодам от 65 до 90.

■ Теперь давай писать программу игры. Сначала используем уже знакомые нам операторы.

Набери:

```
10  LET A = INT(RND(X)*10) + 1
20  INPUT "ЗАДУМАНО ЧИСЛО ОТ 1 ДО 10. ОТГАДАЙ И
    НАПИШИ ЕГО"; Z
30  IF A<>Z THEN 20
40  PRINT "МОЛОДЕЦ, УГАДАЛ!"
```

Это очень простая игра, и ты, конечно, ее понимаешь.

■ Но предположим, что вместо игры в отгадывание чисел мы хотим, чтобы компьютер задумал букву. В любом случае должны ему ответить.

Конечно! CHR$ преобразует номера кодов в соответствующие им символы. Как мы знаем, в нашем случае буквы латинского алфавита имеют номера кодов с 65 по 90. Напишем:

```
10  LET A = INT(RND(X)*26) + 65
15  LET A$ = CHR$(A)
```

Обрати внимание на эти строки. В строке 10 делаем так, чтобы выражение RND(X)*26 генерировало нам случайные числа в диапазоне от 0 до 25 (по числу букв латинского алфавита).

Потом к этому числу прибавляем 65, поскольку буква А соответствует коду 65. Таким образом, суммируя число 65 с числом, получаемым с помощью RND, получим случайные числа, находящиеся между 65 и 90.

С помощью CHR$ компьютер преобразует полученные номера кодов в соответствующие им буквы.

Добавь в программу строки:

```
20  INPUT "ЗАДУМАНА ОДНА ИЗ БУКВ ЛАТИНСКОГО АЛ-
    ФАВИТА. ПОПЫТАЙСЯ ОТГАДАТЬ ЕЕ. НАПИШИ СВОЙ
    ОТВЕТ"; Z$
30  IF A$ <> Z$ THEN 20
40  PRINT "МОЛОДЕЦ, ОТГАДАЛ"
```

При запуске программы компьютер случайным образом возьмет одну из букв алфавита. Как видишь, CHR$ может быть полезной.

Можешь дополнить программу счетчиками, которые тебе скажут, как долго ты угадывал, выдадут количество очков и т. п. Как это сделать, ты уже знаешь.

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

● Этими операторами ребенок не будет часто пользоваться, но мы включили их, чтобы хотя бы ознакомить его с ними. В то же время этим осуществляется введеиие в код ASCII — интернациональиого

кода, применяемого для представления символьно-цифровой информации.

- ASCII — стандартиый код, и основная символьно-цифровая информация имеет всегда одинаковые коды для всех компьютеров. Однако производители компьютеров иногда вносят небольшие изменения в систему кодов, поэтому советуем вам, прежде чем пропускать программы этого пункта, перечитать руководство к вашему персональному компьютеру и пропустить программу со страницы 197 или набрать PRINT ASC("A") и PRINT ASC("Z") для определения кодов снмволов для вашего компьютера.

- Не забывай, что не все компьютеры воспринимают команду RND в одинаковой форме. Возможно, твой компьютер требует такой записи строки 10:

```
10  LET A = RND(26) + 65
```

или такой:

```
10  LET A = INT(RND*26) + 65
```

- Некоторые компьютеры используют команду CODE вместо ASC. В любом случае не забывай, что эта команда дает нам номер кода, соответствующий первому символу строки символов.

Так:

```
ASC("АНТОНИО") = 65
ASC("A") = 65
```

- С другой стороны, для некоторых компьютеров нужно записать:

```
ASC "АНТОНИО"      вместо ASC("АНТОНИО")
    CHR$ 65        вместо CHR$(65)
```

ЧТОБЫ ХОРОШО ПРОГРАММИРОВАТЬ...

После изучения этой книги ты уже довольно много знаешь о языке бейсик. Ты можешь программировать достаточно сложиые разнообразные задачи. Тебе, конечно, понравится, что твои программы хорошие, работают и выполняются, удобны для использования, написаны логически правильно и т. д.

Когда Артуро начал программировать, он встретил некоторые трудности. Не все его программы заработали сразу. На экране появлялись сообщеиия об ошибках и т. д.

Действительно, только практика позволит тебе шаг за шагом программировать с большей уверенностью. В любом случае мы дадим тебе некоторые советы для того, чтобы твои программы были по-настоящему хорошими.

● Прежде всего тебе нужно ясно представить, что ты хочешь получить. Когда ты хорошо разберешься в задаче, тебе будет легче создать программу, соответствующую процессу ее решения.

ЧТОБЫ ХОРОШО ПРОГРАММИРОВАТЬ...

● Потом полезно составить алгоритм решения задачи в виде схемы. Она поможет тебе представить, как будет работать твоя программа.

● Удобно также написать более сложные части программы на бумаге до ввода их в компьютер.

● При вводе программы ты должен быть внимателен к номерам строк, чтобы не повторить их и не стереть (помимо твоего желания) одну из них.

● Будь очень внимателен при обозначении переменных, как цифровых, так и символьных. Используй имена переменных, напоминающие величины, которые они представляют. Например: K$ — для книг, O$ — для одежды, N — для номеров, X1, X2 — для корней квадратных уравнения, D — для дискриминанта и т. д.

ЧТОБЫ ХОРОШО ПРОГРАММИРОВАТЬ...

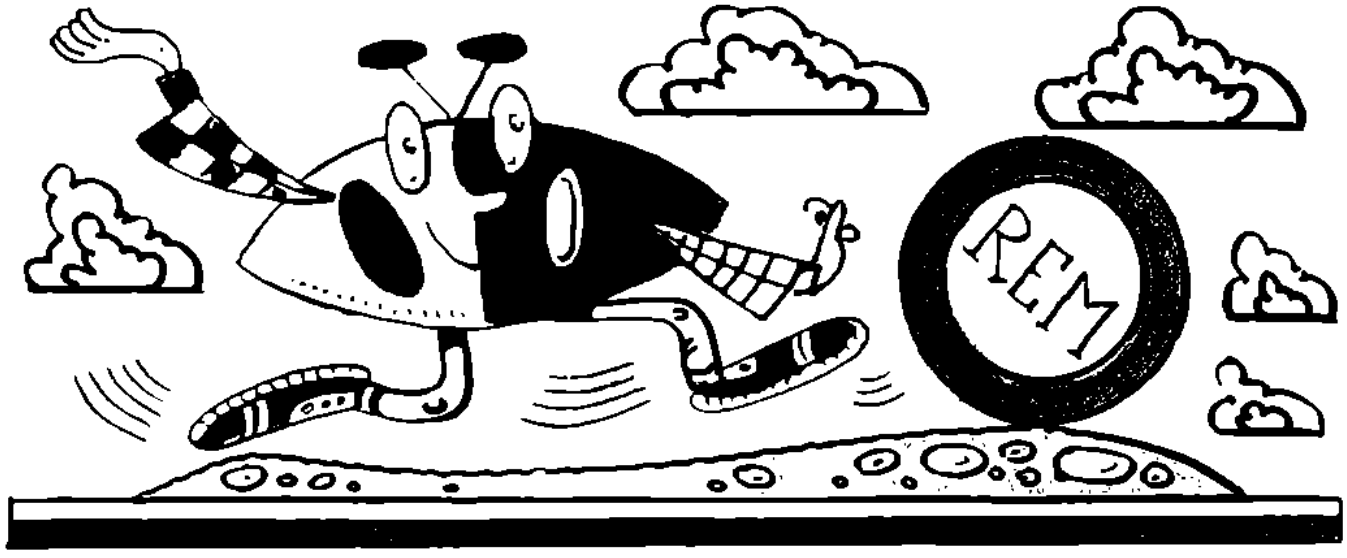
● Помни, что вся структура цикла начинается со служебного слова FOR и заканчивается NEXT.

● Внимательно проверяй операторы внутри конструкции IF... THEN, которые передают управление на соответствующие строки программы.

● Не забудь включить строки для определения возможных ошибок при вводе данных с клавиатуры. С помощью этих строк определяются допущенные при вводе ошибки.

● Когда вносишь изменения, выводи текст программы, чтобы посмотреть, сделал ли компьютер все правильно по форме и в том ли месте, где ты хотел.

● Очень важно также, чтобы программа была понятна любому пользователю, а не только тому, кто ее писал. Не жалей конструкций REM, чтобы вводить комментарии в программу. Ты будешь благодарен сам себе, когда снова будешь работать со своей программой через некоторое время.



● В конце концов ты должен сделать свою программу такой, чтобы ее легко могли использовать другие. Включи в програм-

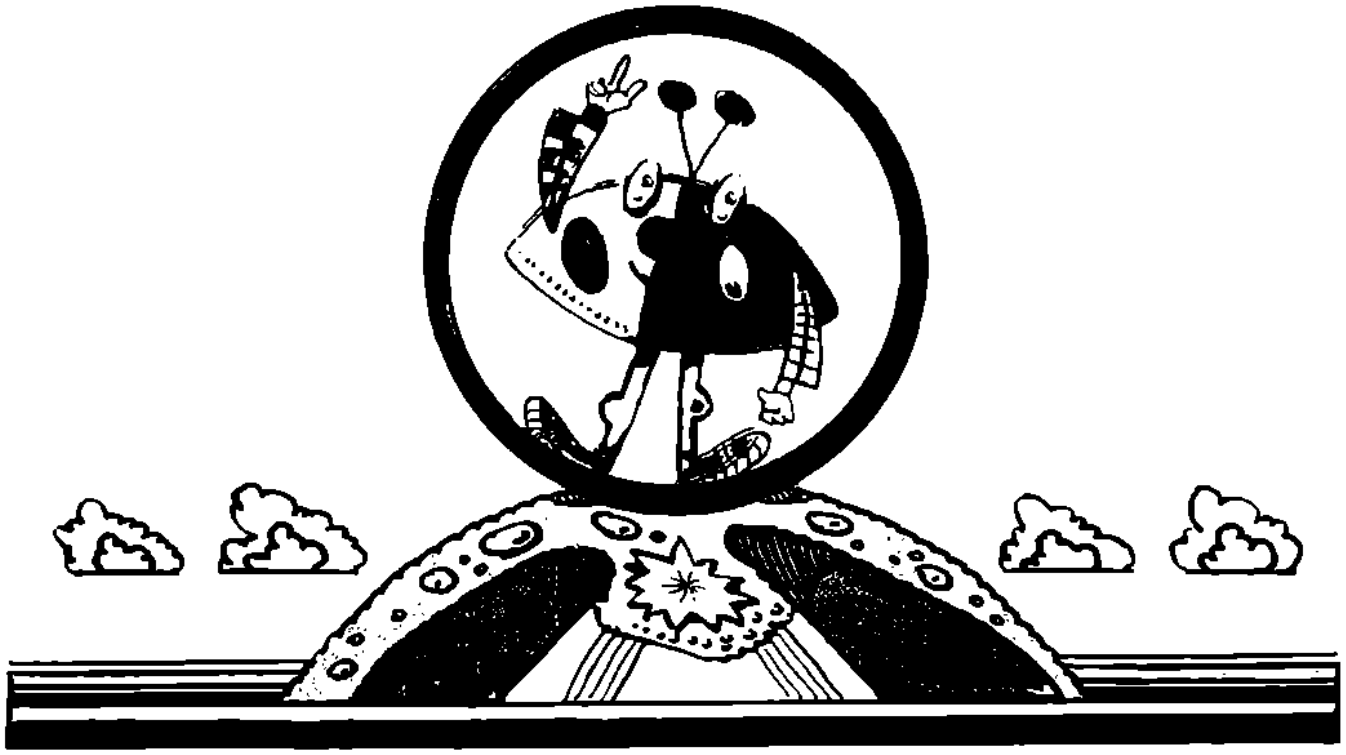
му меню, которое покажет возможности выбора и последующие действия программы, например:

МЕНЮ

1 — купить книгу
2 — купить две книги
3 — купить альбом

● Твои выражения в PRINT должны быть понятны. Например, лучше написать: "СКОЛЬКО КОНФЕТ ЕСТЬ У АРТУРО?", чем "НАПИШИ КОНФЕТЫ". Лучше написать: "РЕЗУЛЬТАТ УМНОЖЕНИЯ ЕСТЬ", чем "РЕЗУЛЬТАТ".

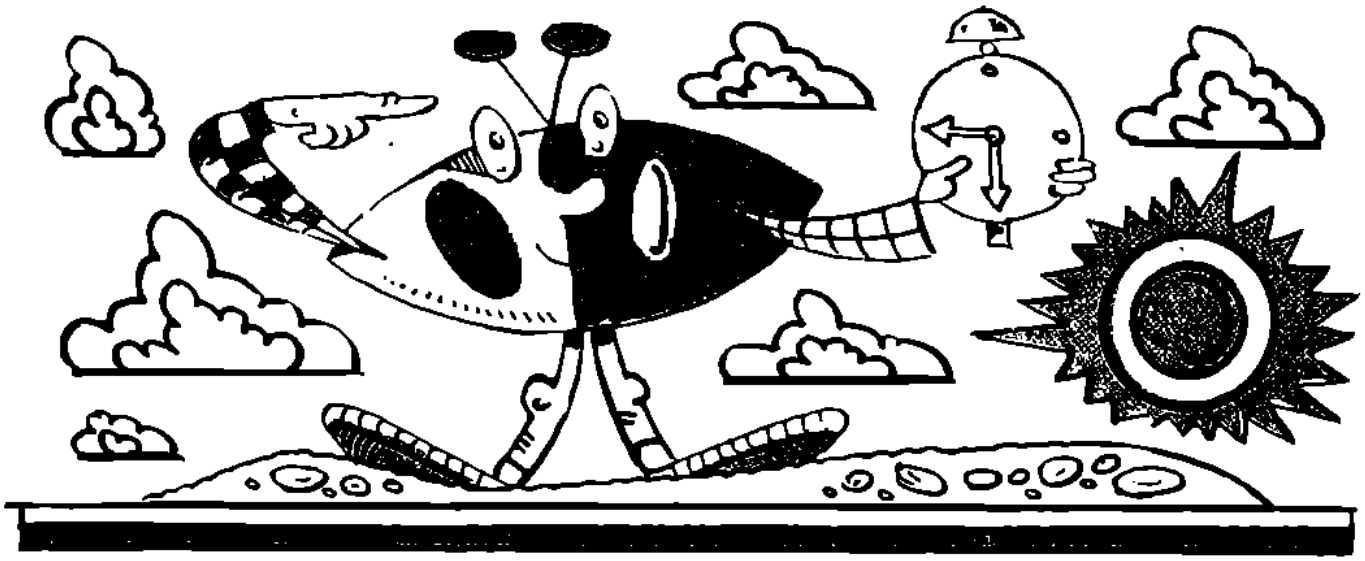
Со временем ты научишься составлять отличные программы.



ПРОВЕРЬ СВОИ СПОСОБНОСТИ

Этой игрой ты можешь выяснить свои способности. Попытайся выиграть у нашего друга Артуро (который хорошо тренирован). Только не нервничай...

```
10  CLS
20  LET Q$ = "ИСПЫТАЙ СВОИ СПОСОБНОСТИ. Я ПОКАЖУ
    НА ОЧЕНЬ КОРОТКОЕ ВРЕМЯ ОДНУ БУКВУ. НАЧИ-
    НАЕТСЯ СЧЕТ ВРЕМЕНИ И БУДЕТ ДЛИТЬСЯ ДО ТЕХ ПОР,
    ПОКА ТЫ НЕ НАЖМЕШЬ НА КЛАВИШУ С ЭТОЙ БУКВОЙ.
    ПОСОРЕВНУЙСЯ СО СВОИМИ ДРУЗЬЯМИ. КТО ПОЛУ-
    ЧИТ МЕНЬШЕ БАЛЛОВ, ТОТ ПОБЕДИТ"
30  LET L = LEN(Q$)
40  FOR N = 1 TO L
50  PRINT MID$(Q$,N,1);
60  FOR W = 1 TO 20 : NEXT W
70  NEXT N
80  FOR Z = 1 TO 1000 : NEXT Z
90  RANDOMIZE
100 LET X = 0
110 LET N = INT(RND(0)*26) + 65
120 LET A$ = CHR$(N)
130 PRINT : PRINT
140 PRINT A$
150 FOR Z = 1 TO 100 : NEXT Z
160 LET B$ = INKEY$ : LET X = X + 1 : PRINT X : CLS : IF B$ =
    "" THEN 160
170 IF B$ = A$ THEN 200
180 IF B$ <> A$ THEN PRINT "ОШИБСЯ. СДЕЛАЙ ЗАНО-
    ВО"
190 FOR Z = 1 TO 200 : NEXT Z : GOTO 90
200 PRINT : PRINT : "ТВОЕ КОЛИЧЕСТВО БАЛЛОВ"; X
210 PRINT : PRINT : PRINT : PRINT : PRINT "ДАВАЙ ЕЩЕ
    РАЗ? (D / N)"
220 LET R$ = INKEY$ : IF R$ = "" THEN 220
230 IF R$ = "D" THEN 90
240 IF R$ = "N" THEN STOP
250 IF R$ <> "D" AND R$ <> "N" THEN PRINT "ОШИБСЯ.
    ОТВЕТЬ ПРАВИЛЬНО D(ДА) ИЛИ N (НЕТ)"
260 GOTO 210
```



ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- В строках с 10 по 70 обеспечивается «красивый» вывод текста на экран. Со строки 90 по 140 генерируются случайным образом буквы алфавита и обнуляется счетчик. В строках со 160 по 180 компьютер анализирует, какая клавиша нажата, и запускает счетчик. Со строки 210 и дальше компьютер позволяет вернуться и повторить все сначала.

- Желательно добавить в эту программу комментарии с REM для пояснеиия некоторых шагов, выполияемых компьютером.

- В этой программе введены специальным образом циклы ожидания, функции MID$, INKEY$, CHR$ и LEN, а также INT и RND. В некоторых моделях компьютеров строка 50 должна быть написана так:

```
50  PRINT Q$(N);
```

Строка 110 может также быть изменена в зависимости от типа компьютера:

```
110  LET N = INT(RDN * 26) + 65
```

или

```
110  LET N = RND(26) + 65
```

ИГРА В РАЗВЕДЧИКИ

Как ты уже убедился, читая предыдущие страницы, нашему другу Артуро очень нравится быть разведчиком. Рассмотрим одну программу для шифрованной связи с секретной группой.

Система очень простая. Когда посылается сообщение, то оно передается в специальном закодированном виде. Каждая буква (ее номер) суммируется с 5 и получаемая буква печатается в сообщении N. Так, на место А пишется Е и т. д.

А	Б	В	Г	Д	Е	Е	Ж
1	2	3	4	5	6	7	8

Когда получатель прочитает сообщение Артуро, он вычтет 5 из каждой буквы для того, чтобы понять и прочитать послание.

Действительно, с помощью компьютера все это можно сделать быстрее. Ты можешь определить с твоими друзьями любую систему кодирования. Единственное, что ты должен иметь в виду,— это то, что нужно использовать одни и те же модели компьютеров для последующей шифровки и дешифровки.

Программа для автоматического кодирования посланий будет иметь вид:

```
10  INPUT "НАПИШИ ФРАЗУ"; F$
20  INPUT "ДАЙ ЦИФРОВОЙ КЛЮЧ"; C
30  FOR X = 1 TO LEN(F$)
35  LET A$ = MID$(F$, X, 1)
40  LET Z = ASC(A$) + C
50  PRINT CHR$(Z);
60  NEXT X
70  PRINT
```

Программа для дешифровки будет следующей:

```
10  INPUT "НАПИШИ КОДИРОВАННУЮ ФРАЗУ"; F$
20  INPUT "ВВЕДИ КЛЮЧ"; C
```

```
30  FOR X = 1 TO LEN (F$)
35  LET A$ = MID$(F$, X, 1)
40  LET Z = ASC (A$) — C
50  PRINT CHR$(Z);
60  NEXT X
```

Для компьютера марки «Синклер» необходимо изменить строку 35 на

```
35  LET A$ = F$(X TO)
```

Определи со своими друзьями свои собственные секретные ключи, и никто не сможет понять ваших посланий...

Конечно, нужно быть очень бдительным... Если «контрразведка» знает вашу систему кодирования и захочет определить ключ кодирования, то это можно сделать, изменив строку 20 на

```
20  FOR C = 1 TO 24
```

и добавив строки

```
70  PRINT
80  NEXT C
90  STOP
```

После таких изменений компьютер напишет на экране все возможные фразы с различными ключами. Одна из них будет иметь смысл.

СОРЕВНОВАНИЕ ПО БАСКЕТБОЛУ

Как уже упоминалось, мы имеем очень простую программу для составления календаря соревнований по баскетболу.

Артуро организует свой чемпионат, и эта программа будет ему очень полезна и придаст уверенности, что он не забудет включить какую-нибудь игру. В чемпионате Артуро участвуют только три команды.

Календарь соревнований будет следующим:

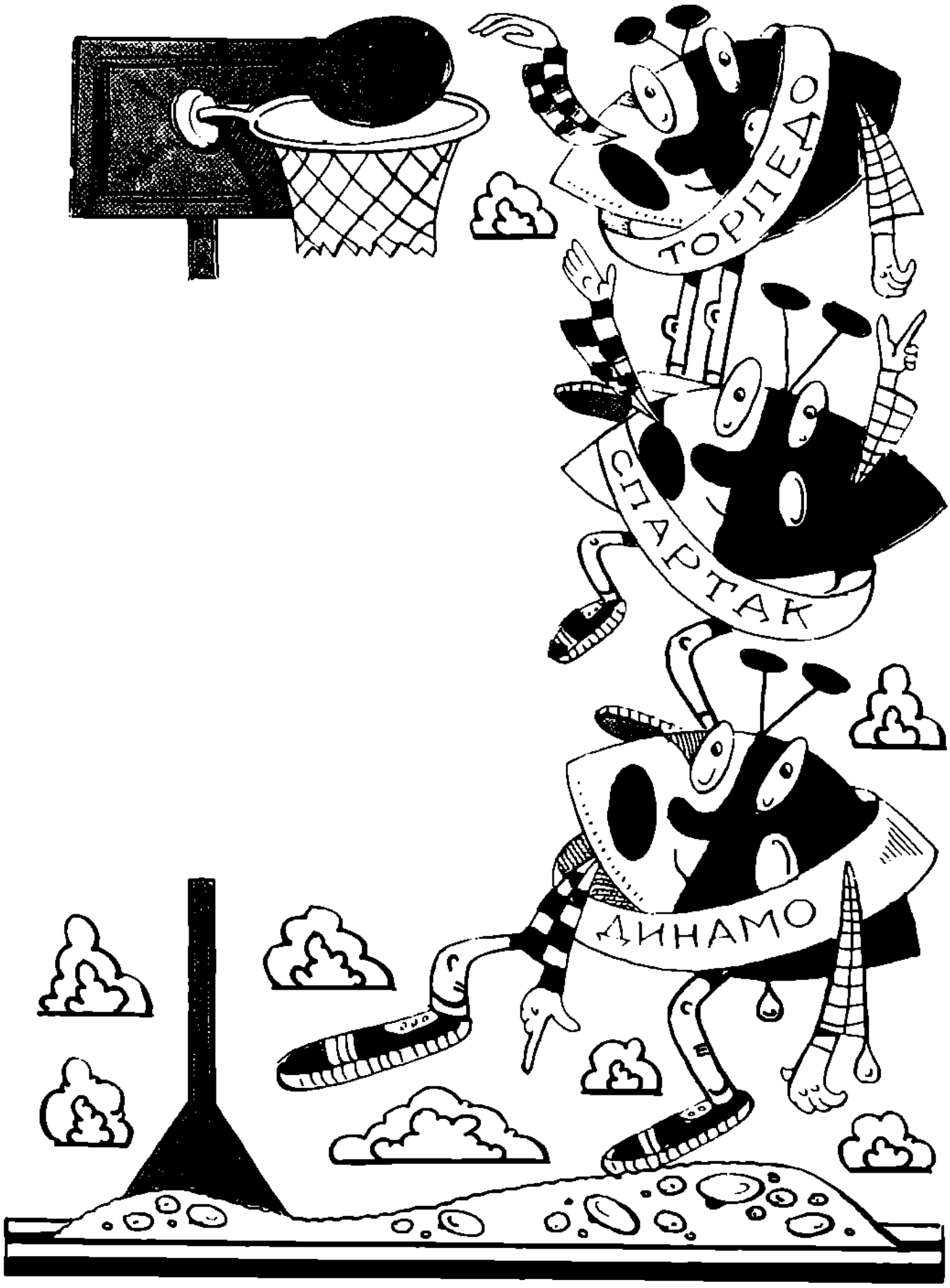
ИГРЫ ЧЕМПИОНАТА

«Динамо»	против	«Спартака»
«Динамо»	против	«Торпедо»
«Спартак»	против	«Динамо»
«Спартак»	против	«Торпедо»
«Торпедо»	против	«Динамо»
«Торпедо»	против	«Спартака»

Программа будет следующей:

```
 10  FOR A = 1 TO 3
 20  FOR B = 1 TO 3
 30  IF B = A THEN GOTO 100
 40  LET E = A
 50  GOSUB 200
 60  PRINT "ПРОТИВ";
 70  LET E = B
 80  GOSUB 200
 90  PRINT
100  NEXT B
110  NEXT A
120  STOP
200  IF E = 1 THEN PRINT "ДИНАМО";
210  IF E = 2 THEN PRINT "СПАРТАК";
220  IF E = 3 THEN PRINT "ТОРПЕДО";
230  RETURN
```

Эта программа подходит только для организации соревнований трех команд, как у Артуро. Если ты хочешь организовать свои собственные соревнования, ты должен указать число команд участников с помощью оператора DIM, который мы изучили (см. с. 157).


ТОРПЕДО
СПАРТАК
ДИНАМО

Программа будет такой:
Введи:

```
 5  INPUT "КОЛИЧЕСТВО КОМАНД"; N
10  DIM A$(N)
20  FOR J=1 TO N
30  PRINT "ВВЕДИ НАЗВАНИЕ КОМАНДЫ С НОМЕРОМ "; J
40  INPUT A$(J)
50  CLS
60  NEXT J
70  CLS
80  PRINT "ИГРЫ ЧЕМПИОНАТА"
90  PRINT
100  FOR X=1 TO N
110  FOR Y=1 TO N
120  IF  X=Y THEN GOTO 150
130  PRINT A$(X); " ПРОТИВ ";
140  PRINT A$(Y)
150  NEXT Y
160  NEXT X
```

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

- Вторая программа позволяет включать любое количество команд, названия которых должны быть введены непосредственно с клавиатуры во время выполнения программы.

- Некоторые компьютеры требуют модифнкацнй строки 10 программы для определения максимального чнсла символов, которые может нметь символьная переменная.
  Так, можно записать

```
10  DIM A$(N, 12)
```

ТЕЛЕФОННЫЙ СПРАВОЧНИК

Ты можешь сделать свой собственный телефонный справочник. Есть много различных систем составления, но мы тебе предлагаем довольно простую.

```
    5  CLS
   10  INPUT "СКОЛЬКО НОМЕРОВ ИМЕЕТ СПРАВОЧНИК?";
       M
   20  PRINT "СПРАВОЧНИК": PRINT
   30  PRINT "ПОИСК ПО:"
   40  PRINT "ИМЕНИ...1"
   50  PRINT "АДРЕСУ...2"
   60  PRINT "КОНЕЦ...3"
   70  INPUT "ВАШ ВЫБОР?"; A
   80  CLS
   90  IF A=1 THEN GOTO 2000
  100  IF A=2 THEN GOTO 3000
  110  IF A=3 THEN STOP
  120  PRINT "ВЫ НЕПРАВИЛЬНО СДЕЛАЛИ ВЫБОР":
       GOTO 30
 2000  INPUT "ВВЕДИТЕ ИМЯ"; E$
 2010  RESTORE
 2020  FOR X=1 TO M
 2030  READ N$, C$, T$
 2040  IF E$=N$ THEN GOTO 2070
 2050  NEXT X
 2060  IF E$<>N$ THEN PRINT "Я НЕ ЗНАЮ ЭТОГО
       ИМЕНИ "; E$: GOTO 2000
 2070  PRINT N$:PRINT C$:PRINT T$
 2080  INPUT "ДРУГОЕ ИМЯ? (D/N)"; Z$
 2090  IF Z$="D" THEN GOTO 2000
 2100  INPUT "ВОЗВРАТ В МЕНЮ? (D/N)"; J$
 2110  IF J$="D" THEN GOTO 30
 2120  STOP
 3000  INPUT "ВВЕДИТЕ АДРЕС"; E$
 3010  RESTORE
 3020  FOR X=1 TO M
 3030  READ N$, C$, T$
 3040  IF E$=C$ THEN GOTO 3070
 3050  NEXT X
 3060  IF E$<>C$ THEN PRINT "НЕ ЗНАЮ ТАКОГО АДРЕ-
       СА "; E$:GOTO 3000
 3070  PRINT N$:PRINT C$:PRINT T$
 3080  INPUT "ДРУГОЙ АДРЕС? (D/N)"; Z$
 3090  IF Z$="D" THEN GOTO 3000
 3100  INPUT "ВОЗВРАТ В МЕНЮ? (D/N)"; J$
```

```
3110  IF J$ = "D" THEN GOTO 30
3120  STOP
5000  DATA ЛУИС, НЕЗАВИСИМОСТИ 345, 2-56-33-24
5010  DATA ФЕРНАНДО, ЦЕНТРАЛЬНАЯ 23, 2-67-9B-99
5020  DATA АНЯ, ПЛ. КОЛУМБА 3, 2-55-54-42
5030  DATA АРТУРО, ПРОСПЕКТ ИНФОРМАТИКИ 2,
      1-01-10-11
```

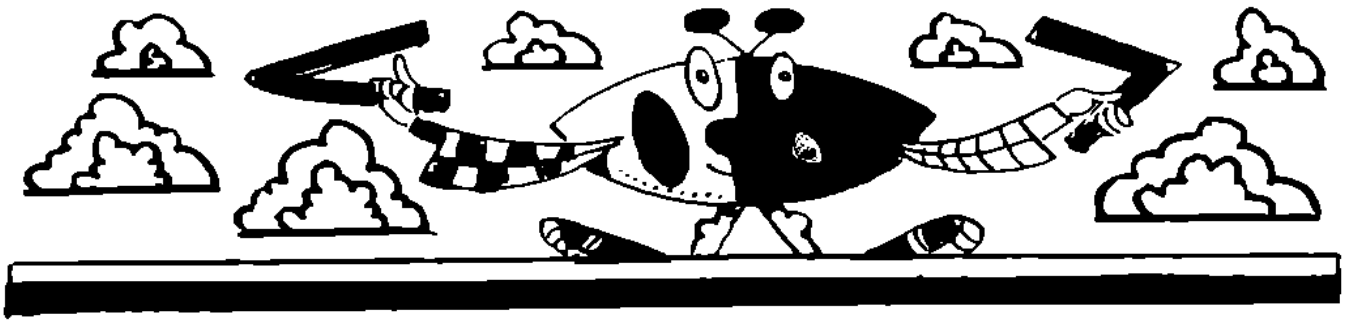
Данные о своих друзьях ты можешь ввести с помощью DATA. Вспомни наши советы, которые мы давали тебе на странице 155 об операторе DATA. Если будешь добавлять новые имена, не забудь изменить строку 10.

СОРТИРОВКА ПО АЛФАВИТУ

```
 10  CLS
 20  INPUT "СКОЛЬКО ИМЕН ХОЧЕШЬ ОТСОРТИРОВАТЬ?"; N
 30  DIM A$(N)
 40  FOR I = 1 TO N
 50  INPUT A$(I)
 60  PRINT A$(I)
 70  NEXT I
 80  PRINT : PRINT
 90  FOR I = 1 TO N - 1
100  FOR J = I + 1 TO N
110  IF A$(I) > A$(J) THEN LET E$ = A$(I) : LET A$(I) =
     A$(J) : LET A$(J) = E$
120  NEXT J
130  NEXT I
140  FOR I = 1 TO N
150  PRINT A$(I)
160  NEXT I
170  END
```

Эта программа позволит тебе записать по алфавиту имена твоих друзей. Как видишь, процесс такой сортировки точно такой же, как и при цифровой сортировке, только в этом случае используем символьные переменные.

Как ты уже заметил в предыдущих пунктах, компьютер также анализирует и классифицирует символы и может опре-

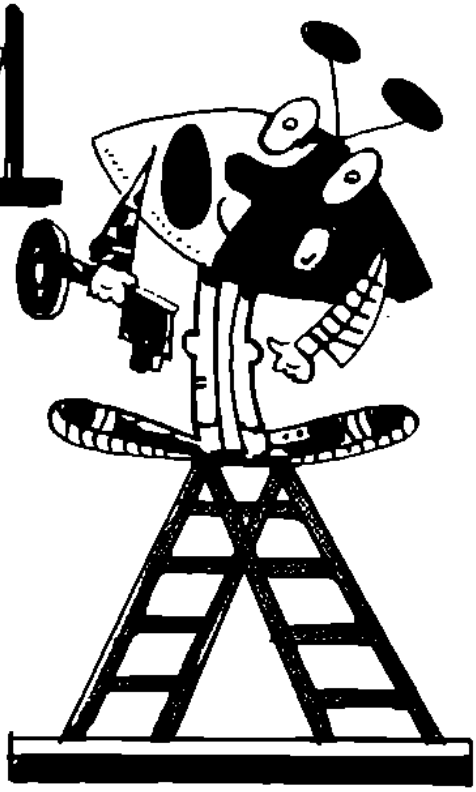


делить, какой из них больше. Со словами он делает то же самое. Сначала сравнивает первые буквы и, если они равны, сравнивает вторые и т. д.

Например:

АНТОНИО < КАРЛОС
АРТУРО > АНТОНИО

УПРАВЛЕНИЕ БИБЛИОТЕКОЙ
(УПОРЯДОЧИМ БИБЛИОТЕКУ)



Очень удобно, чтобы твои книги были отсортированы и классифицированы в такой форме, чтобы ты мог легко найти любую книгу, когда нужно.

Компьютер тебе может помочь в этом. Программа, которую мы тебе предлагаем, очень удобная и простая, только немного длинная.

Ты можешь использовать до двух цифр для определения раздела.

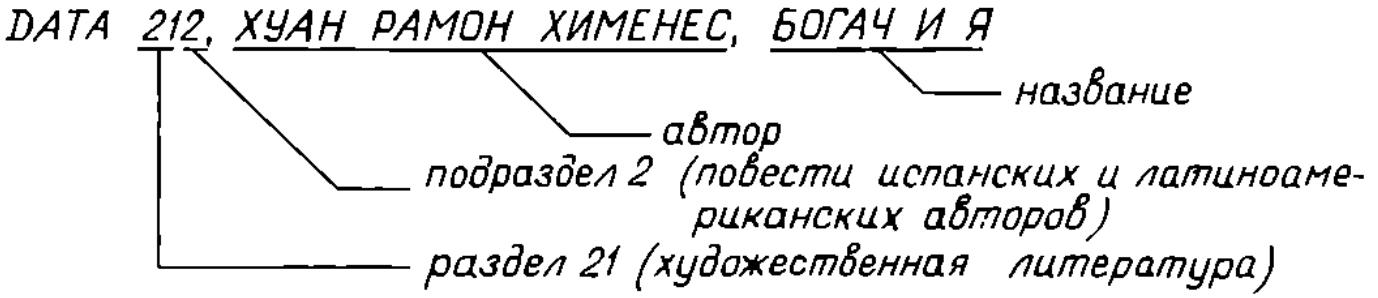
Например:

МЕДИЦИНА — 14
ХУДОЖЕСТВЕННАЯ ЛИТЕРАТУРА — 21

И один разряд для подраздела. Например:

ПОВЕСТИ — 2

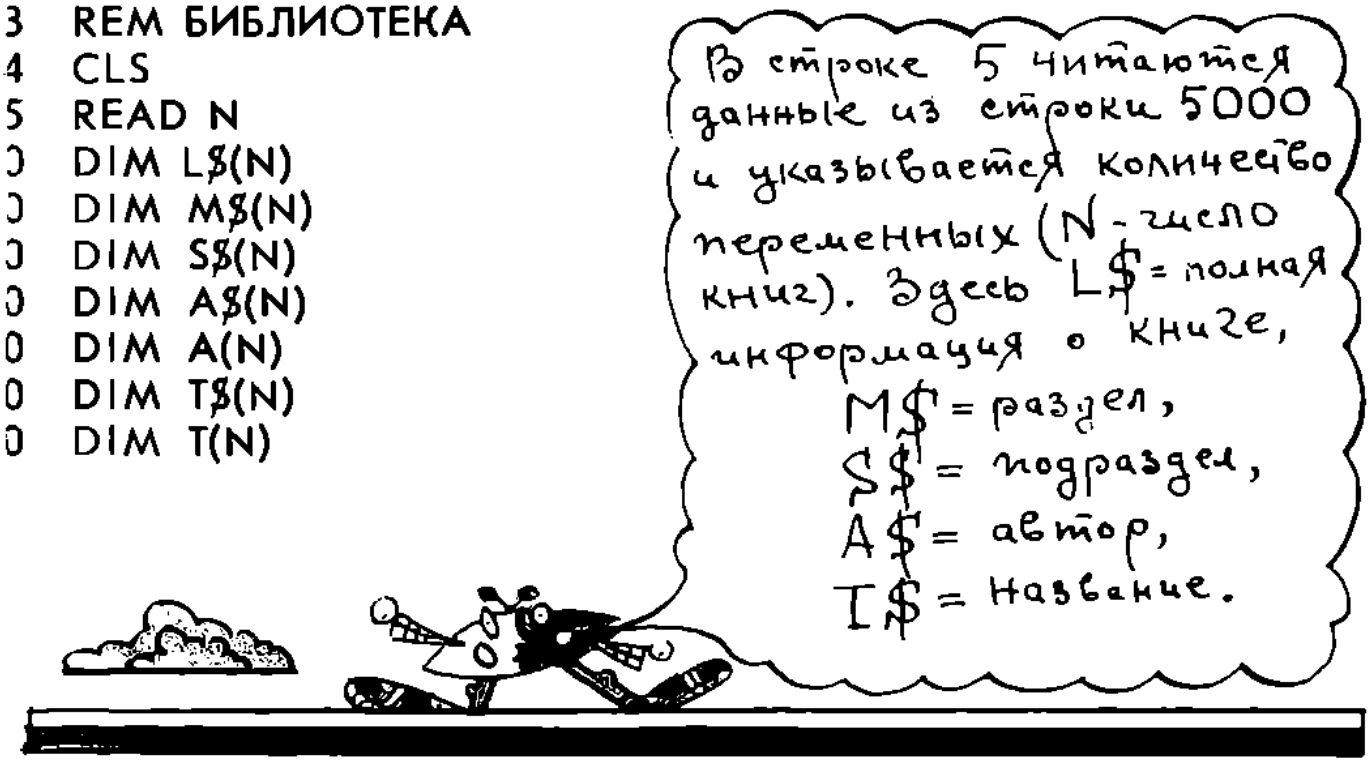
Ты можешь придумать свою собственную классификацию или воспользоваться библиотечной классификацией. Посоветуйся со своим учителем. Используя ключи (коды) для разделов и подразделов, мы можем отсортировать книги логическим способом. Для кодирования какой-нибудь повести испанского или латиноамериканского автора мы можем использовать код 212. Кроме этого, можем добавить и фамилию автора, и название. Например:



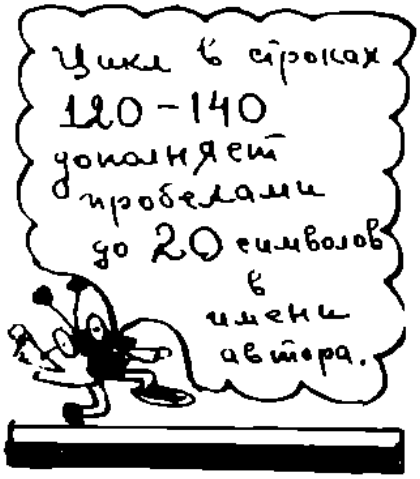
Расположи все данные в конце программы и ты сможешь запросить любую информацию у компьютера. Если ты введешь номер раздела, то компьютер тебе даст фамилии авторов и названия произведений, которые есть в твоей библиотеке по этому разделу (и притом в алфавитном порядке). Если ты введешь фамилию автора, компьютер сообщит названия произведений этого автора и номера соответствующих разделов. Если введешь название произведения, то компьютер выдаст фамилию его автора и номер соответствующего раздела.

Эта программа резервирует 20 символов для фамилий авторов и 20 символов для названий произведений, поэтому при вводе данных необходимо не выйти за эти ограничения.

```
  3  REM БИБЛИОТЕКА
  4  CLS
  5  READ N
 10  DIM L$(N)
 20  DIM M$(N)
 30  DIM S$(N)
 40  DIM A$(N)
 50  DIM A(N)
 60  DIM T$(N)
 70  DIM T(N)
```

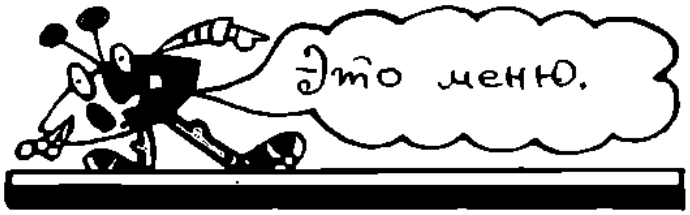



```
 80  FOR I = 1 TO N
 90  READ M$(I), A$(I), T$(I)
100  LET A(I) = LEN(A$(I))
110  LET T(I) = LEN(T$(I))
120  FOR J = A(I) + 1 TO 20
130  LET A$(I) = A$(I) + "  "
140  NEXT J
```




```
150  FOR K = T(I) + 1 TO 20
160  LET T$(I) = T$(I) + "  "
170  NEXT K
180  LET L$(I) = M$(I) + S$(I) + A$(I) + T$(I)
```

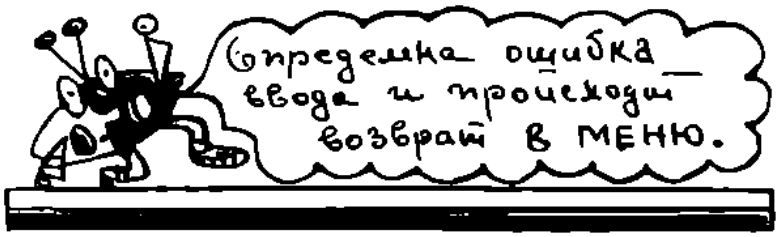

```
 190  NEXT I
1000  FOR J = 1 TO N - 1
1010  FOR K = J + 1 TO N
1020  IF L$(J) < L$(K) THEN GOTO 1060
1030  LET E$ = L$(J)
1040  LET L$(J) = L$(K)
1050  LET L$(K) = E$
1060  NEXT K
1070  NEXT J
```



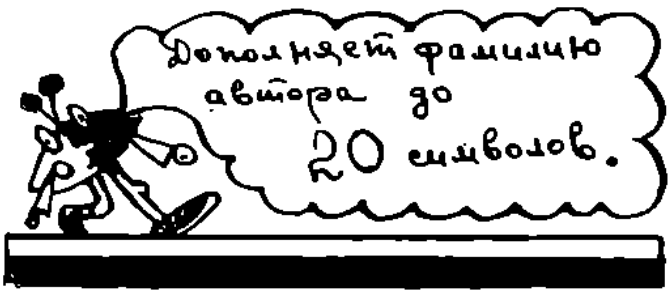

```
1080  PRINT : PRINT : PRINT
1090  PRINT "ДЛЯ ПОИСКА ПО РАЗДЕЛУ НАЖМИ 1"
1100  PRINT "ДЛЯ ПОИСКА ПО ИМЕНИ АВТОРА НАЖМИ 2"
1110  PRINT "ДЛЯ ПОИСКА ПО НАЗВАНИЮ НАЖМИ 3"
1120  INPUT "ЧТО ВЫ ХОТИТЕ СДЕЛАТЬ?"; O

1130  IF O = 1 THEN GOTO 1200
1140  IF O = 2 THEN GOTO 1400
1150  IF O = 3 THEN GOTO 1600
1160  IF O <> 1 AND O <> 2 AND O <> 3 THEN
      GOTO 1120
1170  GOTO 1090
1200  CLS
```

```
1210  INPUT "ВВЕДИ ШИФР РАЗДЕЛА, КОТОРЫЙ ХОЧЕШЬ
      ПОСМОТРЕТЬ"; X$
1220  FOR I = 1 TO N
1230  IF X$ = LEFT$(L$(I),2) THEN GOTO 1250
1240  GOTO 1260
1250  PRINT L$(I)
1260  NEXT I
```

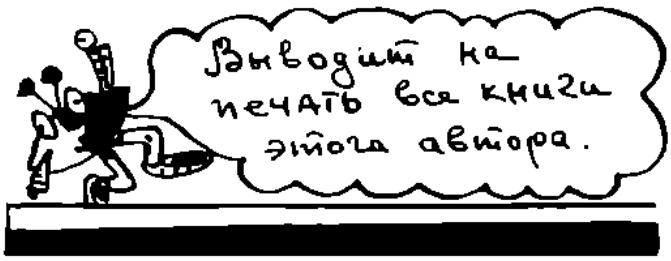

```
1270  INPUT "ХОЧЕШЬ ВЕРНУТЬСЯ В МЕНЮ? (D/N)"; Y$
1280  IF Y$ = "N" THEN STOP
1290  IF Y$ <> "D" AND Y$ <> "N" THEN GOTO 1270
1300  CLS

1310  GOTO 1080
```




```
1400  CLS
1410  INPUT "КНИГИ КАКОГО АВТОРА ХОЧЕШЬ ПОСМОТ-
      РЕТЬ?"; N$

1420  LET Q = LEN(N$)
1430  FOR W = Q + 1 TO 20

1440  LET N$ = N$ + "  "
1450  NEXT W
```

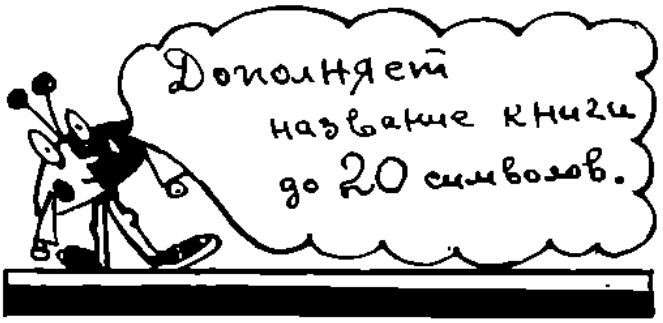


```
1460  FOR I=1 TO N
1470  IF MID$(L$(I),4,20)=
      N$ THEN GOTO 1490
1480  GOTO 1500
1490  PRINT L$(I)
1500  NEXT I
```

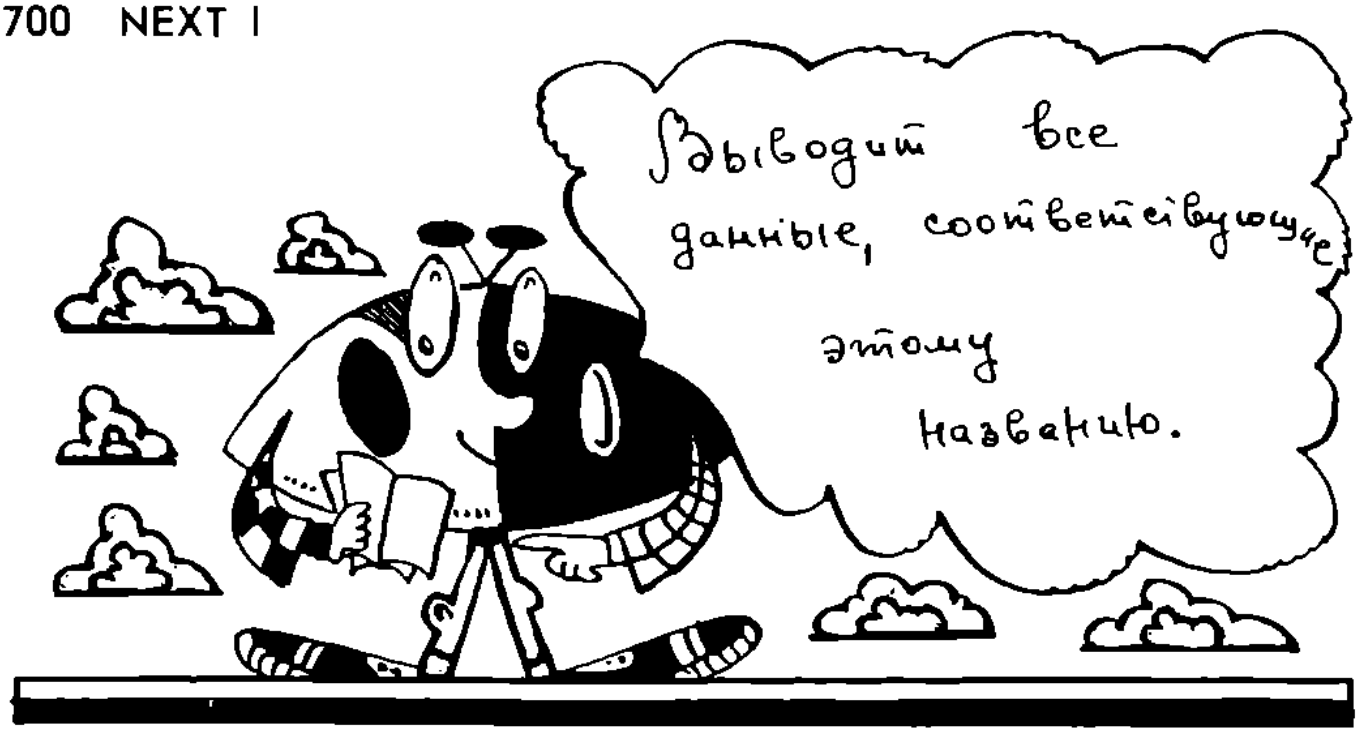



```
1510  INPUT "ХОЧЕШЬ ВЕРНУТЬСЯ В МЕНЮ? (D/N)"; Y$
1520  IF Y$="N" THEN STOP
1530  IF Y$<>"D" AND Y$<>"N" THEN GOTO 1510
1540  CLS
1550  GOTO 1090
1600  CLS
1610  INPUT "ВВЕДИ НАИМЕНОВАНИЕ КНИГИ"; Z$
```

```
1620  LET P=LEN(Z$)
1630  FOR  R=P+1 TO 20
1640  LET Z$=Z$+"  "
1650  NEXT R
```




```
1660  FOR I=1 TO N
1670  IF Z$=RIGHT$(L$(I),20) THEN GOTO 1690
1680  GOTO 1700
1690  PRINT L$(I)
1700  NEXT I
```



```
1710  INPUT "ХОЧЕШЬ ВЕРНУТЬСЯ В МЕНЮ? (D/N)"; Y$
1720  IF Y$="N" THEN STOP
1730  IF Y$<>"D" AND Y$<>"N" THEN GOTO 1710
1740  CLS
1750  GOTO 1090
5000  DATA 3
5010  DATA 21,2,ХУАН РАМОН ХИМЕНЕС,БОГАЧ И Я
5020  DATA 21,2,Г. ГАРСИЯ МАРКЕС,ПУСТОСЛОВИЕ
5030  DATA 08,5,МАРТИНЕС, МАТЕМАТИКА
```

ЗАМЕЧАНИЯ ДЛЯ ВЗРОСЛЫХ

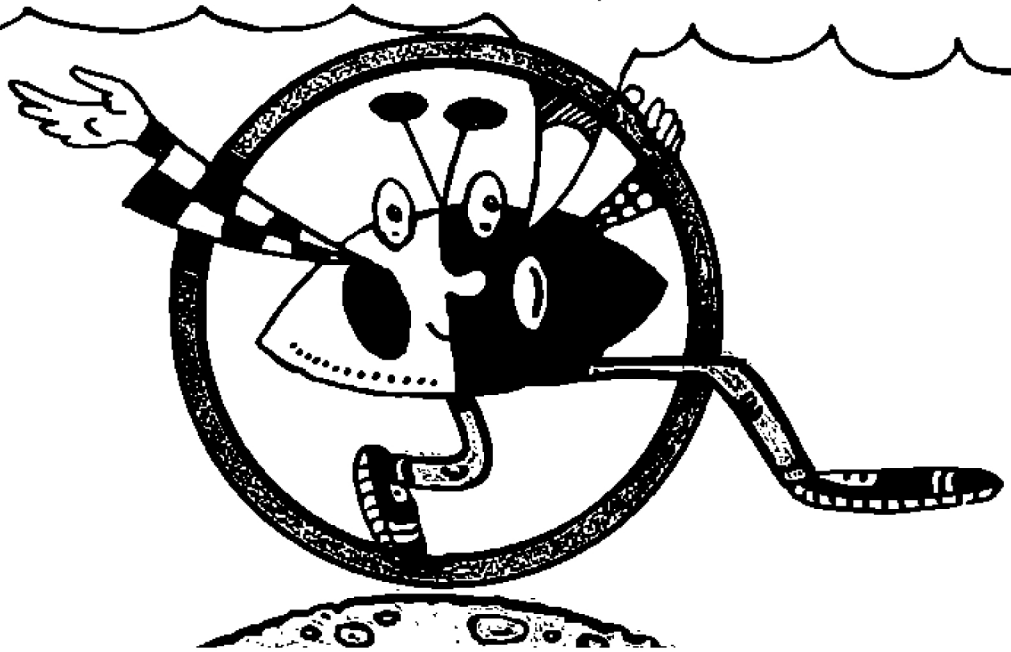
- Это типичная программа, написанная на обобщением бейсике. Как ты уже знаешь, необходимо ввести минимальные изменеиия для того, чтобы она выполнялась иа определенном компьютере. Ты должен иметь в виду все уточнения, сделанные нами в предыдущих пунктах по этому поводу.

- Напоминаем о важности правильного написания оператора DATA, описания размерности массивов и модификаций (где необходимо) таких операторов, как LEFT$, MID$ и RIGHT$ в зависимости от типа компьютера.

А теперь я ухожу. Мы вместе много поняли в Бейсике. Если тебе это понравилось, ты можешь продолжить изучение. Конечно, совершенствованию нет предела, однако ты знаешь уже довольно много для составления достаточно сложных программ.

Главное, что ты можешь составлять собственные программы, давая работу своему воображению, и понимаешь, что важнее, чем программировать, — знать, как использовать компьютер в качестве инструмента повышения производительности

ТРУДА.



СОДЕРЖАНИЕ

Учебное издание

Ватт София, Мангада Мигель

БЕЙСИК ДЛЯ ДЕТЕЙ

Перевод с испанского

Переводчики *Олейник Анатолий Григорьевич, Симоненко Валерий Павлович*
Художник *Игнатов Вадим Николаевич*

Киев, «Радянська школа»

Заведующий редакцией математики и физики *Н. Е. Зубченко.*
Литературный редактор *Г. В. Брезницкая.*
Художественный редактор *В. А. Пузанкевич.*
Технические редакторы *Л. Б. Ланцман, В. Н. Зайцева.*
Корректор *Л. А. Волынская.*

ИБ № 6449

Сдано в набор 23.03.89. Подписано к печати 20.03.90. Формат 70×100/$_{16}$. Бумага офсетн. № 1. Гарнитура школьная. Усл. печ. л. 18,2+0,325 форзац. Усл. кр-отт. 74,74. Уч.-изд. л. 13,02+0,54 форзац. Тираж 50 000 экз. Изд. № 32801. Заказ 9-116. Цена 1 руб.

Издательство «Радянська школа», 252053, Киев, Ю. Коцюбинского, 5

Книжная фабрика «Коммунист», 310012, Харьков, Энгельса, 11.

Ватт С., Мангада М.

В21 Бейсик для детей: Пер. с исп.— К.: Рад. шк., 1990.— 222 с.

ISBN 5-330-00135-8.

В книге в игровой и наглядной форме, доступной младшим школьникам, излагаются основы программирования и объясняется азбука одного из наиболее распространенных и удобных в использовании языков программирования — языка бейсик. Текст снабжен необходимыми комментариями для взрослых — родителей и учителей, которые даже не имея специальных знаний в области программирования, смогут проконсультировать юных читателей.

Для учащихся младшего и среднего школьного возраста.

В $\frac{4802020000—246}{M210(04)—90}$ 381—90

ВБК 32.97

1 р.