BEGINNERS' BASIC

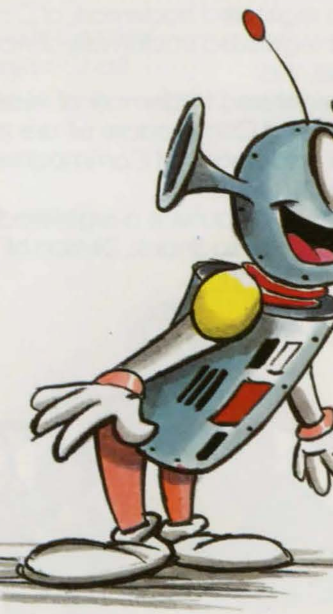
MICRO WORLD

HAYES

BEGINNERS' BASIC

by Peter Lear

Editor	Carol Batchelor, B.A., M.A. Graham Richards
Illustrator	Ron Migliore
Designers	Karen McArdle Margaret Horvath Pat Strachan



Hayes Publishing Ltd., 3312 Mainway, Burlington, Ontario L7M 1A7

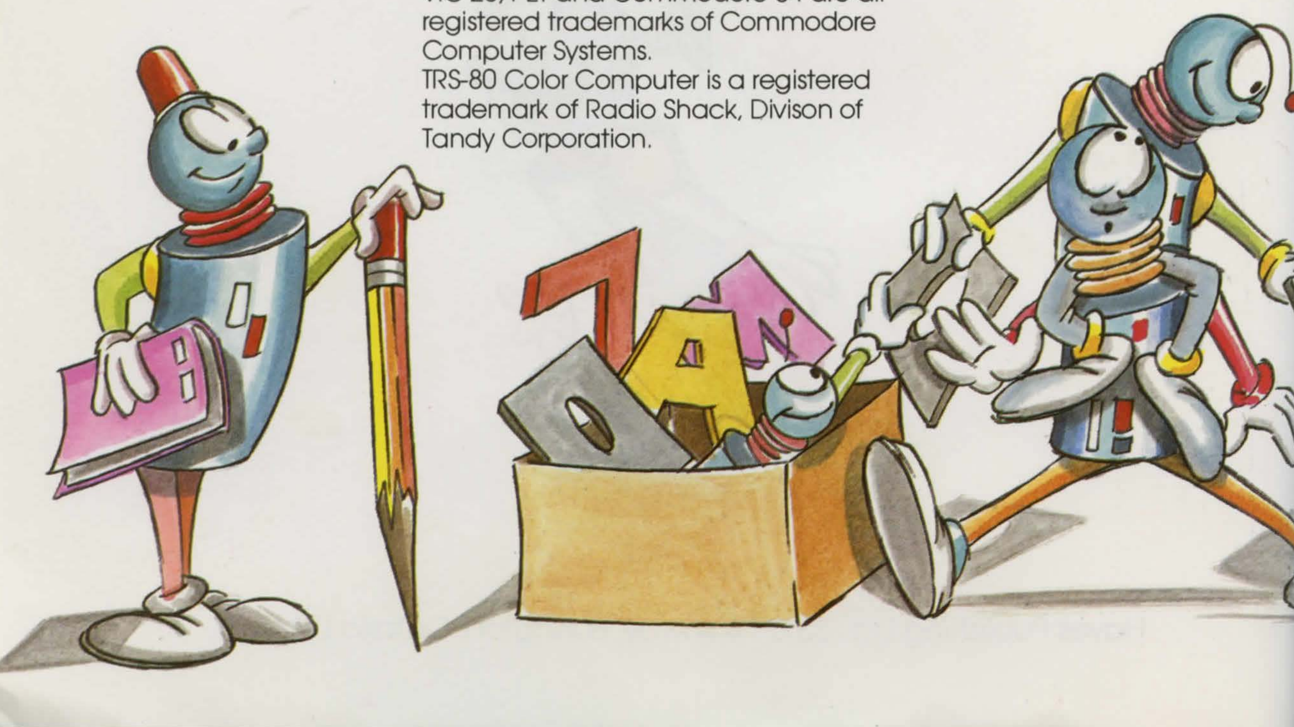
NOTE

The publisher and author have made every effort to assure that the computer programs are accurate and complete. However, this publication is prepared for general readership, and neither the publisher nor the author have any knowledge about or ability to control any third party's use of the programs and programming information. There is no warranty or representation by either the publisher or the author that the programs or programming information in this book will enable the reader or user to achieve any particular result.

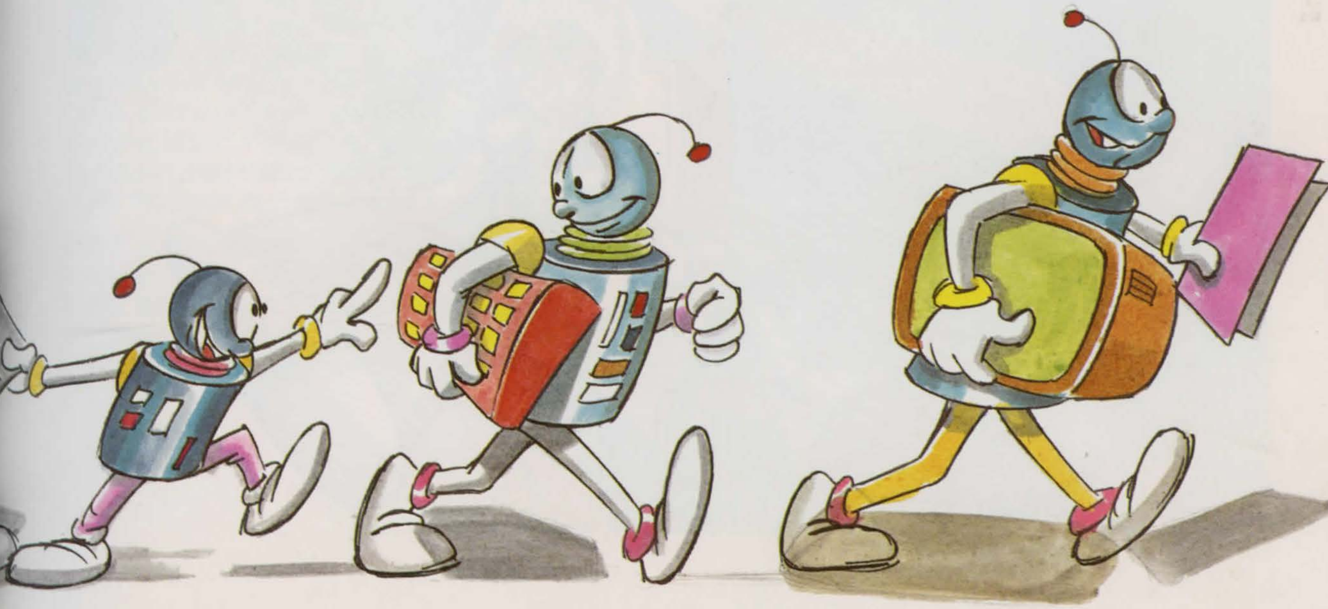
Copyright© 1984 by Hayes Publishing Ltd.
Burlington, Ontario

ISBN 0-88625-075-7

Printed in Hong Kong.



CONTENTS



INTRODUCTION

This book will help you learn most of the common BASIC language words you'll need to write your own computer programs. To learn the fastest and to have the most fun, sit at your computer and type in the examples that are given.

BASIC varies slightly from one brand of computer to another. Because of this, the book uses a "generic" form of BASIC throughout, so you'll have to consult the BASIC CONVERSION CHART on page 60 to find the variations used by Apple, ADAM, Commodore, Atari and TRS-80 Color Computers. Make sure you type in the proper words and format for your computer! (There are reminders of this throughout the book.)

Now let's have some fun and learn BEGINNERS' BASIC!

BASIC

This book is about "BASIC." BASIC is a language just like English is a language, French is a language and Spanish is a language.

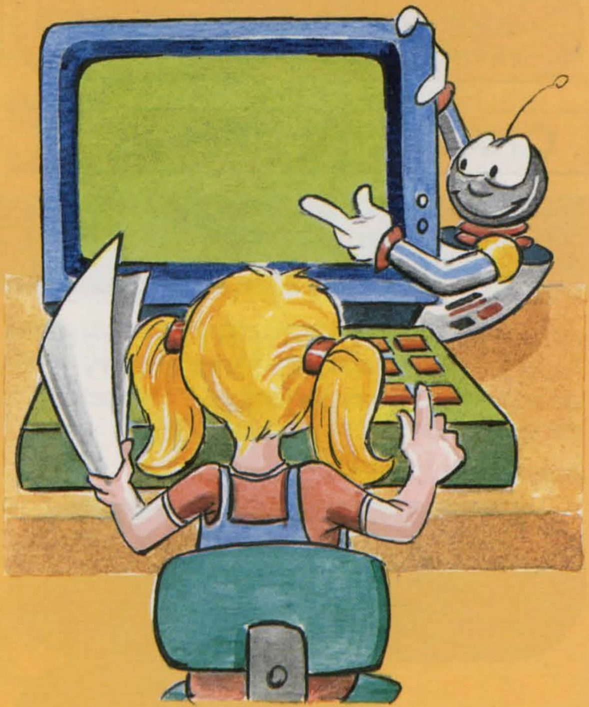
If I want to talk to someone in Spain, I must speak to that person in Spanish so he will understand. It is the same with computers. If I want to communicate with a computer, I must use a language the computer will understand. One of the languages a computer understands is BASIC.

By the way, BASIC stands for "Beginner's All-purpose Symbolic Instruction Code." If you want to confound your enemies and amaze your friends, you can tell them you're studying Beginner's All-purpose Symbolic Instruction Code. It sounds more profound than, "I'm studying BASIC." You'll find many people like to confound their friends with computer jargon, so study this book and join the club.

BASIC as a language is quite similar to English. There are words like "PRINT," "SAVE" and "LIST." BASIC tends to use English words so you can understand them easily. It also takes other English words and compresses or shortens them. For example, "GOTO" is a BASIC word which is the same as "go to" in English. BASIC takes out the space between the English words because the computer wants short, concise statements as commands. In English I could say, "Please go to the store." But when I am communicating with the computer I must use the exact form of the BASIC command so that it will understand.

When you are communicating with the computer and you use a word it does not understand or you misspell a word, the computer usually responds with the expression "SYNTAX ERROR." This is the computer's way of saying "I do not understand" or "The spelling of that word is not correct" or "That word is not in my vocabulary." The computer, you see, has a very limited vocabulary. This is sometimes good and sometimes bad. It's good because you don't have to learn a lot to become a good programmer, but bad because it has a limited number of phrases to tell you what's wrong or why something isn't working.

The important thing for now is to learn the commands precisely so that when you use BASIC the computer will understand what you want it to do and not respond with SYNTAX ERROR every time you try to do something.



YOUR FIRST COMMAND

"PRINT" is a very powerful BASIC command. PRINT tells the computer to write something, either on the screen or on paper through a printer. Telling the computer to PRINT in BASIC is like saying, "Speak to me," or "Write for me," or "Respond to me," in English. Someday when all home computers can talk, we will probably have a command in BASIC called SPEAK, but for now we must use PRINT. So let's see what PRINT can do.

Turn on your computer and type the command word PRINT. Now you must tell the computer what you want it to PRINT. It will PRINT whatever you type next, providing you put your request inside double quotation marks (" "). So enter this:

```
PRINT "HELLO"
```

The word PRINT plus whatever you want PRINTed is called a "PRINT STATEMENT."

Now press "ENTER" or "RETURN." What did the computer do? You can put anything you want in between the quotes: spaces, words, numbers, symbols and punctuation. Try your name. Try:

```
PRINT "HELLO, MR. SMITH"
or
PRINT "HELLO, MOM"
```

CHARACTER STRINGS

In computer language any series of words or numbers in quotes is called a "STRING." Also in computer language letters, numbers, symbols and even spaces are called "CHARACTERS."

Here are some examples of STRINGS or, to be more precise, CHARACTER STRINGS. That's a good term in computer jargon – "character string." Remember it!

Examples of CHARACTER STRINGS:

```
"BASIC IS FUN"
"TESTING 1 2 3"
"CHARACTER STRING"
"GOOD BYE"
```

A CHARACTER is defined as a letter, number, symbol or space.

PRINT them on your screen. Remember to type PRINT first, then your string in quotes. This method of instructing the computer is known as command structure.

```
PRINT "            "
```

Press ENTER or RETURN.

As you've seen from the examples, the computer can PRINT numbers on the screen if told to with the PRINT command.

Type in this PRINT statement:

```
PRINT "1 2 3 4"
```

Remember the quotes. What is the character string? 1 2 3 4, that's correct. Numbers may also be PRINTed out of quotes. Try these:

```
PRINT 1; 2; 3; 4
PRINT 1, 2, 3, 4
```

Notice how the first example only left one space between the numbers, while the second example put 10 spaces between the numbers.

The reason for this is that the semicolon (;) in the first example instructs the computer just to separate the numbers, and the comma (,) in the second instructs the computer to space over 10 spaces between the numbers. Try:

```
PRINT 27, 34, 4763, 1
```

Make up some examples of your own and use commas and semicolons to mix up the numbers.

PRACTICAL STUFF

When numbers are not in quotes, we can do math with them.

Addition uses the + sign.
Subtraction uses the – sign.
Multiplication uses the * sign.
Division uses the / sign.

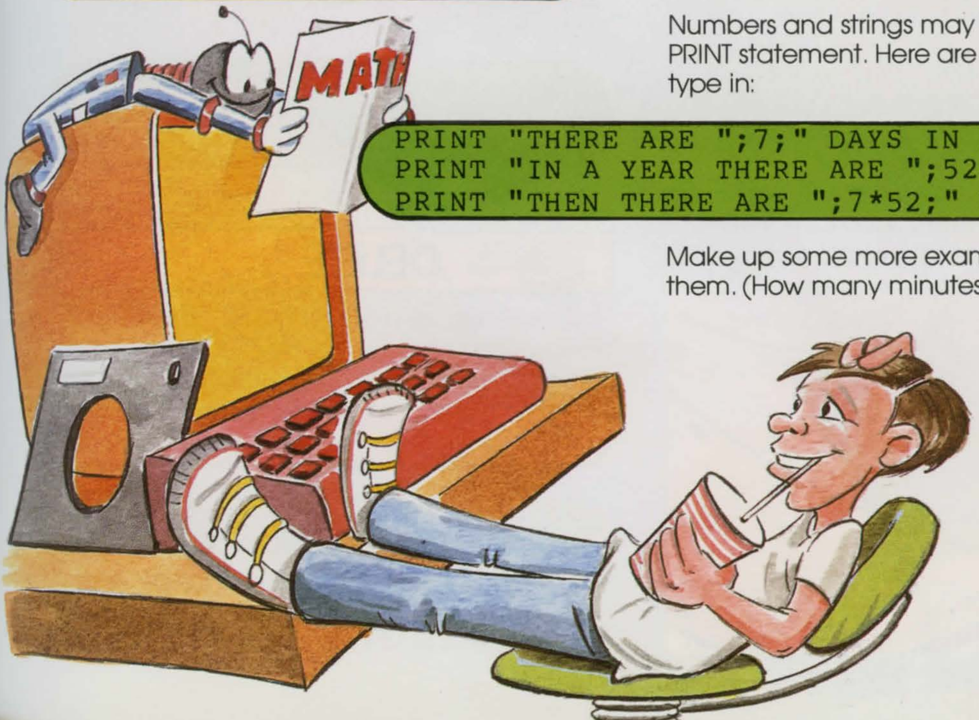
Type in the following examples. Remember to press ENTER or RETURN after each line.

```
PRINT 14+8
PRINT 12.5-3.75
PRINT 6*4
PRINT 376*238
PRINT 12/3
PRINT 4+7-1Ø
PRINT 6*3-11
PRINT 4*7-1,16/4;7*8
```

The computer can become a good homework helper. Try some questions from your math book.

Brackets may be used to have the operations performed in the order you want. The computer will do the calculations in brackets first, then use the answers it got to do the other calculations. Try these examples:

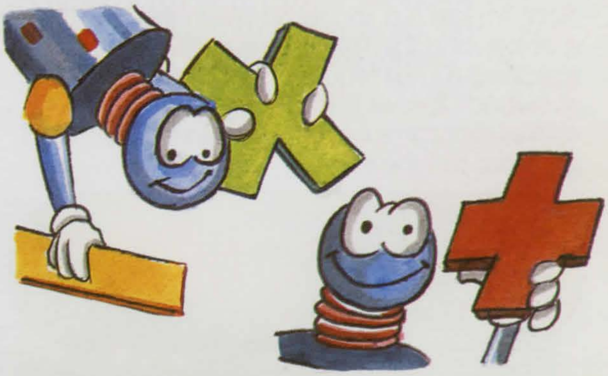
```
PRINT 27-(6*7)
PRINT ((3*17)+(18*19))/5
```



PROBLEM:
You went to the store and bought four candy bars that cost 42 cents each. What change did you get from $5.00? The computer will give you the answer if you write the problem in BASIC. Here is how it looks:

```
PRINT 5-(4*.42)
```

Try this one:
You got $100 dollars for your birthday and bought 3 pets for $27 each. How much have you got left?



```
Answer:  PRINT 1ØØ-(3*27)
```

Think of some more examples for yourself and try them. Use your math book or help your mom with her shopping list.

COMBINING STRINGS

Numbers and strings may be COMBINED in a PRINT statement. Here are some examples to type in:

```
PRINT "THERE ARE ";7;" DAYS IN A WEEK"
PRINT "IN A YEAR THERE ARE ";52;" WEEKS"
PRINT "THEN THERE ARE ";7*52;" DAYS IN A YEAR"
```

Make up some more examples like this and try them. (How many minutes are there in a year?)

PROGRAMS

When you have several PRINT statements, they can be joined together by putting a number before each PRINT statement line. These numbers are called "LINE NUMBERS." By putting statements together in this way you create what is known as a "PROGRAM." A computer program is a series of instructions for the computer to carry out in the sequence given by the line numbers.

Notice there is a space between the line number and the PRINT statement. Strictly speaking it's not required, but almost everybody follows this convention because it makes programs easier to read. Here is a PRINT program to type in:

```
1Ø PRINT "HELLO"
2Ø PRINT "THERE ARE ";
   24*6Ø;" MINUTES IN A DAY"
3Ø PRINT "GOOD BYE"
```

Whoops! The computer did not write anything on the screen this time. There is a reason. Because you used line numbers, the computer accepted and stored each line (so it wouldn't forget what you'd told it) and waited for the next line you typed in. In computer terms, the computer put the small program in its "MEMORY." Now in order to get it back so we can use it, we must instruct the computer to recall it with a BASIC command word. "RUN" is the BASIC command that tells the computer to go get a program from its memory and then do what the program tells it to do. (Of course, you must still press ENTER or RETURN.) When you do so, the computer will "call up" the program and perform each instruction in the numerical order in which you have entered it. It will do what's on line 10 first, line 20 second, line 30 third, and so on. RUN the above program.

LIST

You can look at the program lines you have entered by typing the BASIC command "LIST." Do so now.

The computer LISTs out the whole program. You can ask the computer to LIST only one line, like this:

```
LIST 1Ø
```

Or a number of lines, like this:

```
LIST 2Ø-3Ø
```

Atari users check the conversion chart.

Make sure you use the right SYNTAX (the right commands and the proper spelling) or you'll get a SYNTAX ERROR.

Find the SYNTAX ERROR in this line:

```
1Ø RPINT "HELLO"
```

You can also add to your program. Enter these lines:

```
4Ø PRINT
5Ø PRINT "HELLO AGAIN"
```

Are you pressing ENTER or RETURN after each line?

Now LIST your program again. RUN the program. What did line 40 do? Why do you think it did that?

Program lines are generally numbered 10, 20, 30 and so on, so that if you have to, you can insert new lines between the existing lines without retyping all the line numbers. Type these lines:

```
15 PRINT "DID YOU KNOW THAT"
25 PRINT "WHICH IS ";24*6Ø*
   6Ø;" SECONDS"
```

LIST and RUN your program again.

DELETE

You may also take out or "DELETE" lines. To do so, you simply type in the line number without any statement at all. To delete line 30, type:

```
3Ø
```

Remember to press ENTER or RETURN.

LIST the program now. Line 30 no longer exists. Put it back in by retyping it. (Again, remember to press ENTER or RETURN.)

SAVING

Programs only need to be typed in once. Then you may use a program as many times as you want with the BASIC command RUN. There is a catch though; when you turn the computer off, the program is lost forever. A computer's memory is not like your brain. You can remember things from way back, but a computer only remembers as long as it has power going to its memory. (If you want to learn more about a computer's memory and how it works, read "Micro World," another book in this series.) But you can save all your hard work by saving your program some place other than the computer's own memory.

"SAVE" is the BASIC command word that instructs the computer to store information somewhere other than its own memory. You can SAVE a program:

1) on a "CASSETTE TAPE" by means of a "CASSETTE TAPE RECORDER," or
2) on a "FLOPPY DISK" by means of a "DISK DRIVE."

The instructions to connect your cassette recorder or disk drive to your computer will be in your owner's manual. If you are going to use a tape recorder, skip the yellow circle.

To use a disk to store programs, you must first prepare the floppy disk to receive information. This process is called "INITIALIZING" the disk (some computers use the term "FORMATTING"). Each computer has its own way of initializing a disk. Check your disk drive or computer manual for details.

SAVING

When you are ready to store a program on cassette tape or disk, you must instruct the computer to do so by using the BASIC command SAVE. This is normally followed by a name for the program, in quotation marks. You should make sure that the name you give the program will help you remember what the program is about. We will now store the "minutes in a day" program we have just done. Type:

```
SAVE "                    "
```

Put the name you choose inside the quotes and press ENTER or RETURN.

Check your computer manual's section on SAVEing programs in case the instructions are slightly different for your computer.

VERIFY YOUR PROGRAM

After storing a program you should check that it was stored correctly. This is called "VERIFYing." When the computer VERIFYes a program it compares the program it has in its memory to the one you have just stored on tape or floppy disk.

To VERIFY a program on tape, you must first rewind the tape to where the program starts. This step is not needed when using a floppy disk. You then use the BASIC command VERIFY followed by the name you gave the program. Type:

```
VERIFY "                    "
```

Use the name you gave your program.

For some computers this may vary. Refer to your owner's manual.

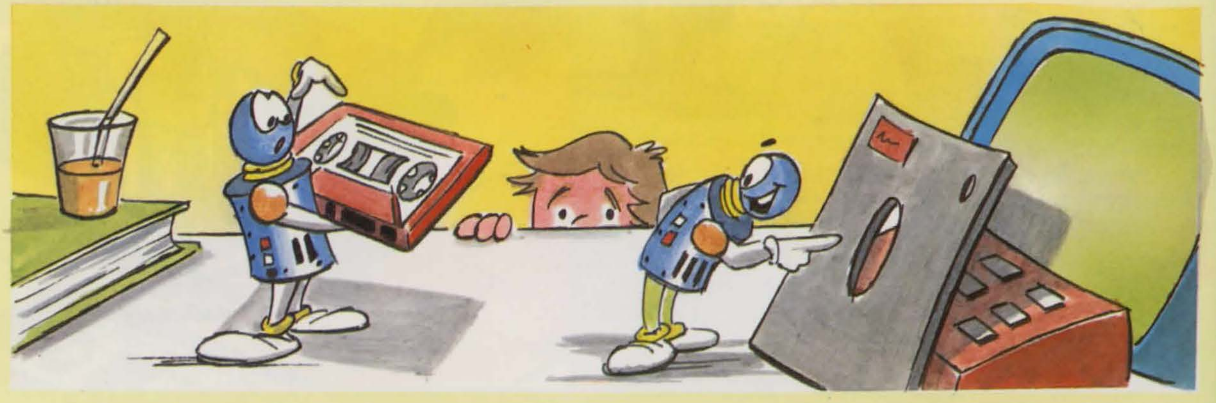
If your equipment is working and the computer gives you an error message, there could be one of several reasons why.

TAPE 1) There is no tape in the tape recorder.
2) The tape is worn out.
DISK 3) You did not initialize your floppy disk.
4) There is no disk in the disk drive.
5) The disk is worn out.

When the program is VERIFYed properly you can turn the computer off. When you turn the computer on again you can retrieve that program by either rewinding the tape to where the program starts, or rewinding the tape to the beginning.

TIP – This can be a very slow process, so if your cassette recorder has a counter, make a note of where each program you have saved starts and ends.

If you have a disk drive, you don't have to do anything yet.



LOADing

Taking a program from tape or disk and putting it back into the computer is called "LOADing." The BASIC command for most computers to LOAD a program is simply LOAD followed by the program's name in quotes.

```
LOAD "NAME"
```

ENTER or RETURN

Once the program is LOADed into the computer's memory you can RUN it by typing RUN or you can change it by typing LIST and making whatever changes you want. Remember, if you change the listings (lines) you should SAVE and VERIFY the program again.

NEW

If you want to get rid of a program in the computer's memory, you don't have to turn the computer off. If you're sure you won't be using a program again and you'd like to start a new one, all you have to do is type the BASIC word "NEW" and press ENTER or RETURN. This clears the computer's memory and lets you start fresh.

We're moving right along. We now know these BASIC words and how to use them:

PRINT

RUN

LIST

SAVE

VERIFY

LOAD

NEW

And we also know these computer terms:

CHARACTER

CHARACTER STRING

PROGRAM

DELETE

INITIALIZING

FORMATTING

Practice doing small programs. It will not only be fun, it will help you remember your new language BASIC and its vocabulary.

For Disk Users:
A feature the disk drive has that the cassette tape does not is a table of the contents on your floppy disk. This table of contents is referred to as a "CATALOG" or a "DIRECTORY." Refer to your disk drive manual for information on the particular command to see the catalog or directory.

FUN TIME!

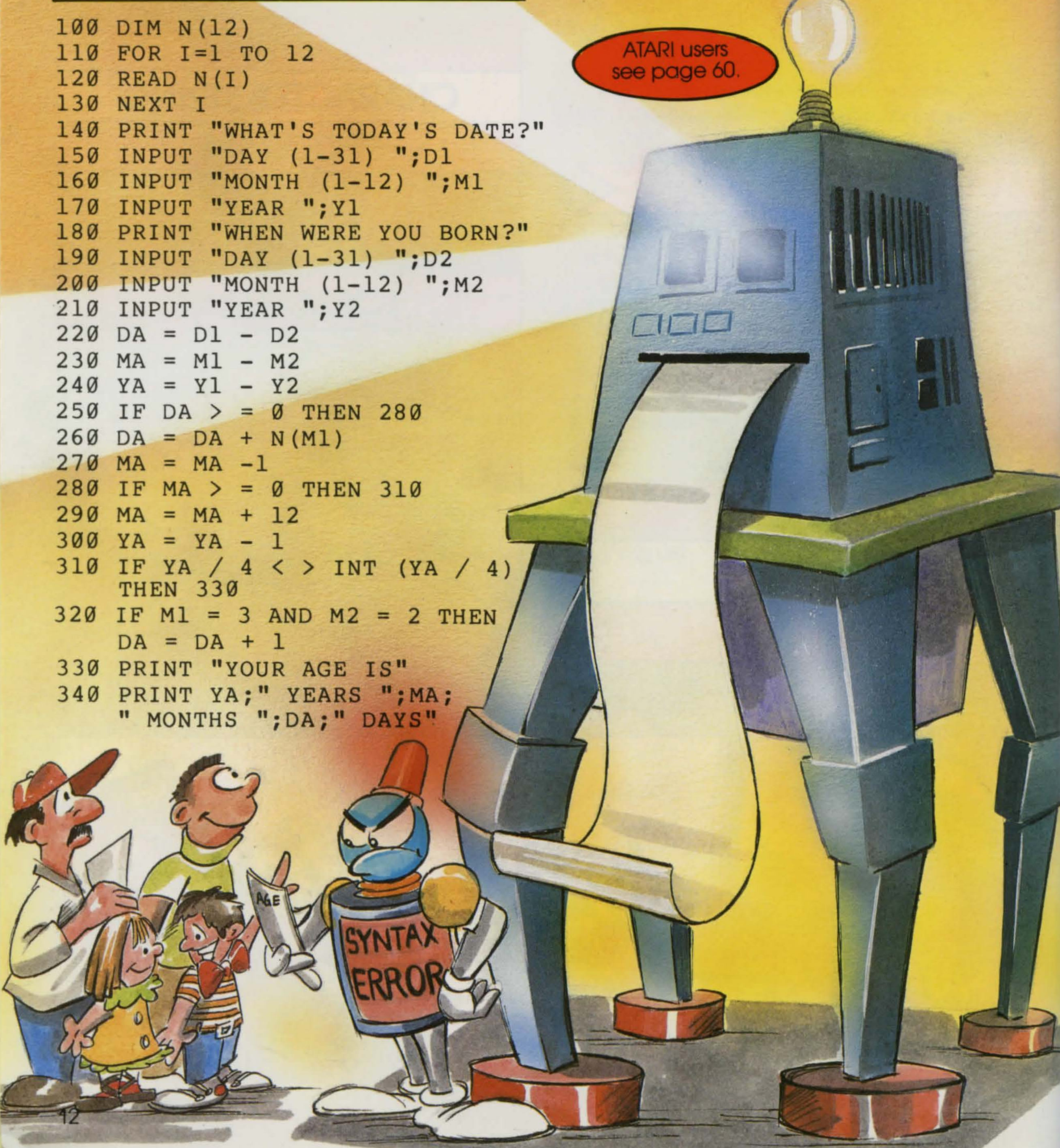
You've learned a lot, so now we'll break for some fun. While you will not be familiar with all the BASIC words in this next program, it will be good practice for you to enter a program. Make sure that you enter all the words exactly or it's SYNTAX ERROR time!

The program is called "AGE FINDER" and it asks for today's date and your birth date. With this information it calculates your exact age in years, months and days. Here is the program:

AGE FINDER

```
100 DIM N(12)
110 FOR I=1 TO 12
120 READ N(I)
130 NEXT I
140 PRINT "WHAT'S TODAY'S DATE?"
150 INPUT "DAY (1-31) ";D1
160 INPUT "MONTH (1-12) ";M1
170 INPUT "YEAR ";Y1
180 PRINT "WHEN WERE YOU BORN?"
190 INPUT "DAY (1-31) ";D2
200 INPUT "MONTH (1-12) ";M2
210 INPUT "YEAR ";Y2
220 DA = D1 - D2
230 MA = M1 - M2
240 YA = Y1 - Y2
250 IF DA > = Ø THEN 28Ø
260 DA = DA + N(M1)
270 MA = MA -1
280 IF MA > = Ø THEN 31Ø
290 MA = MA + 12
300 YA = YA - 1
310 IF YA / 4 < > INT (YA / 4)
    THEN 33Ø
320 IF M1 = 3 AND M2 = 2 THEN
    DA = DA + 1
330 PRINT "YOUR AGE IS"
340 PRINT YA;" YEARS ";MA;
    " MONTHS ";DA;" DAYS"
350 DATA 31,31,28,31,3Ø,31
360 DATA 3Ø,31,31,3Ø,31,3Ø
```

ATARI users see page 60.



INPUT and OUTPUT

INPUT

"INPUT" is a computer term that refers to any information or "RAW DATA" a computer is given to work with. In the last section we gave the computer words, numbers, strings and equations to work with. All these things are INPUT.

The computer received this information through the keyboard; we typed it in. Since the keyboard is used to INPUT information into the computer, it is called an "INPUT DEVICE."

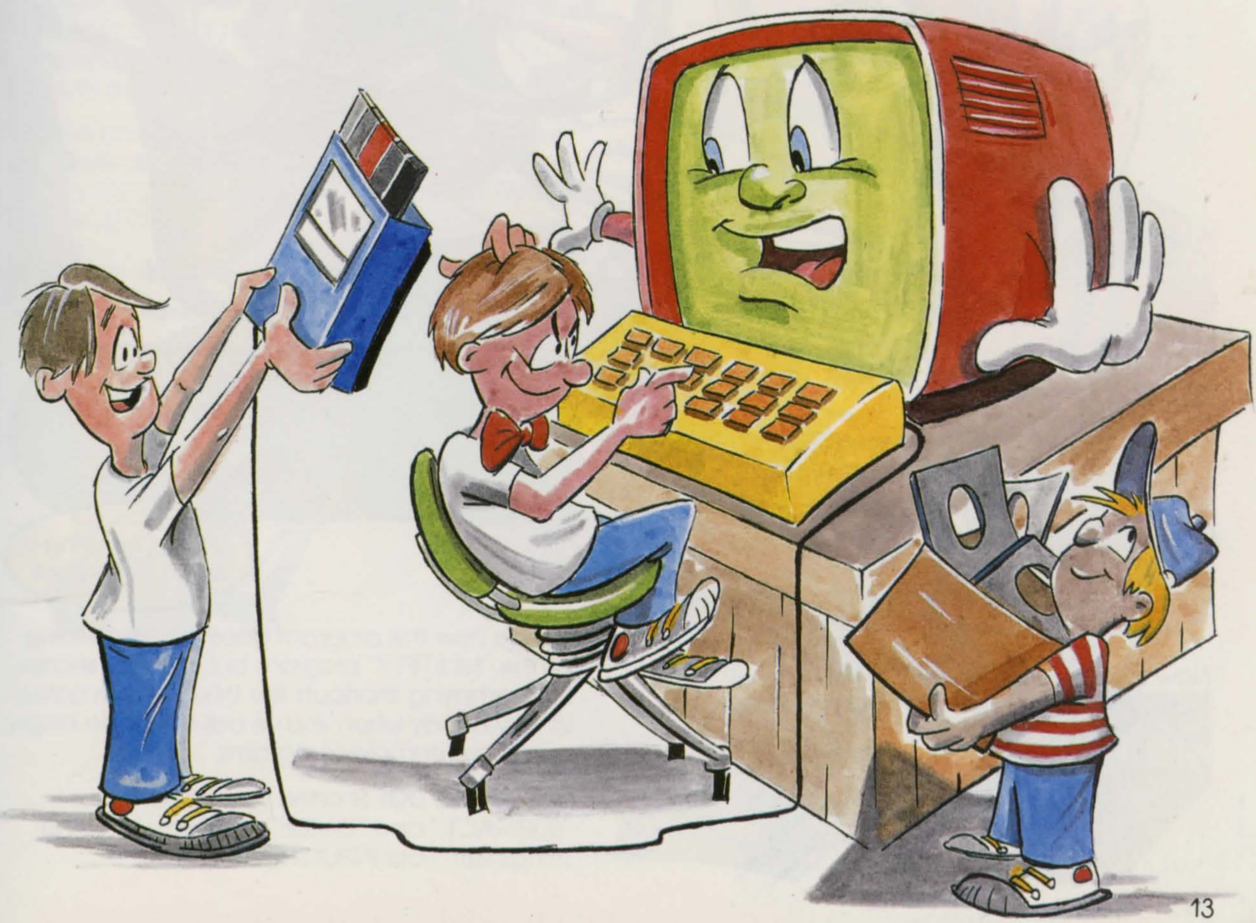
"INPUT" is also a BASIC language word. It can be used in a program to tell the computer to expect some incoming data. Here is how INPUT is used in a program:

Remember this is a PRINT statement.

```
1Ø PRINT"ENTER FIRST NUMBER"
2Ø INPUT A
3Ø PRINT"ENTER SECOND NUMBER"
4Ø INPUT B
5Ø C=A*B
6Ø PRINT A;" TIMES ";B;"=";C
```

This is called an INPUT statement.

SAVE this program using "MULTIPLY" as its name. RUN the program and enter a number for A and another number for B. The program instructs the computer to figure out the product of the two numbers.



OUTPUT

"OUTPUT" is the answer the computer gives you after it has worked on your information. If A and B in the MULTIPLY program were the INPUT, what was the OUTPUT? If you said "C" – you're right!

Just as there are INPUT devices, there are also "OUTPUT DEVICES." The screen is an OUTPUT device; it displays the answer information. If you wanted a paper copy of your OUTPUT, you could direct the information to another OUTPUT device, the "PRINTER," and it would print out your answer. Can you think of another OUTPUT device? How about your cassette recorder or disk drive?

You can OUTPUT information to your tape or disk by SAVEing it; but you can INPUT information into your computer by LOADing a program from your tape or disk. Cassette recorders and disk drives are both INPUT and OUTPUT devices.

FOR YOU TO DO:

Go back to the little program "MULTIPLY" and create a new program called "ADD" by retyping lines 50 and 60 to read:

```
5Ø C=A+B
6Ø PRINT A;"+";B;"=";C
```

Now create similar programs called "SUBTRACT" and "DIVIDE." Are you SAVEing your programs as you go along?

CHARACTER STRINGS

A "CHARACTER STRING" is a series of words or numbers inside quotation marks, like this:

```
"WORDS AND OR NUMBERS LIKE 2 3 4"
```

Most computers allow you to use a CHARACTER STRING with an INPUT statement, like this:

```
1Ø INPUT "FIRST NUMBER ";A
2Ø INPUT "SECOND NUMBER ";B
3Ø C=A*B
4Ø PRINT A;"*";B;"=";C
```

character string in INPUT statement

Notice how this program does the same thing as the "MULTIPLY" program but is 2 lines shorter. Programming shortcuts like this one can come in very handy when you've advanced to longer and more complex programs.

FOR YOU TO DO: Shorten your "ADD," "SUBTRACT," and "DIVIDE" programs by using strings with your INPUT statements.

VARIABLES

A "VARIABLE" is something which represents or stands for something else. For example, B=12. The variable "B" has the value of or represents the number 12.

For the moment we're going to look at variables that represent only numbers. These are called "NUMBER VARIABLES."

But we've already used variables, haven't we? Take another look at our "MULTIPLY" program.

The variables we've seen so far have all represented the values of positive whole numbers. Variables can also have the value of negative numbers or decimal numbers. Here are some examples:

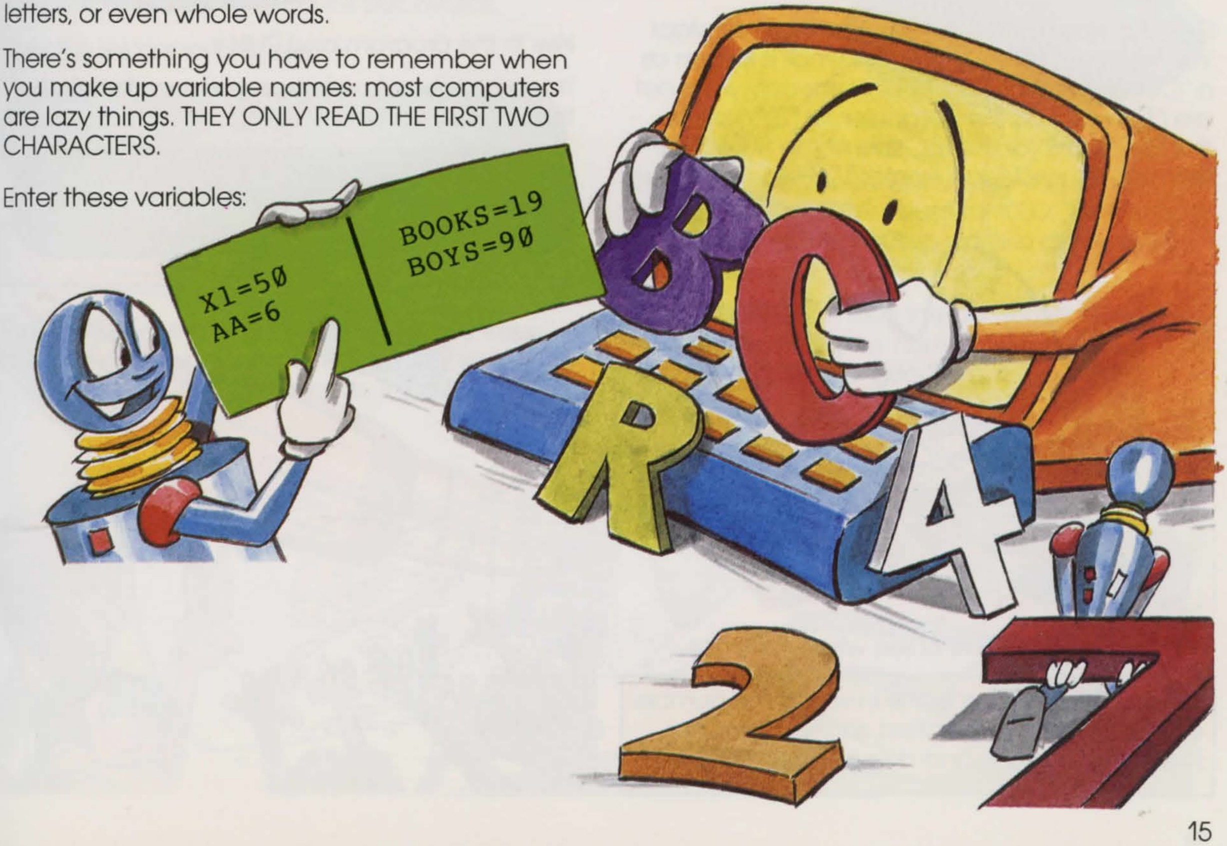
```
A=-47
G=.75
```

Variables can be PRINTed too. Wait a minute, we've already done that too! That "MULTIPLY" program taught you more than you thought!

Variable names, like the "B" in the example above, can be longer than a single character. (Remember what a character is? Right, a letter, number or symbol.) Variable names can be letters and numbers together, two or more letters, or even whole words.

There's something you have to remember when you make up variable names: most computers are lazy things. THEY ONLY READ THE FIRST TWO CHARACTERS.

Enter these variables:



```
X1=5Ø
AA=6
BOOKS=19
BOYS=9Ø
```

Now PRINT the variables, like this:

```
PRINT X1,AA,BOOKS,BOYS
```

What happened to BOOKS? Books and boys both start with "BO," so when you entered the second BO variable, the computer thought you were changing the value of the first one and not creating a second variable at all! Remember this when you are making up variable names, the computer only reads the first two characters; make them different.

There's one other thing to remember. You CANNOT use BASIC language words as variable names. The computer would read them as commands and not variables and become terribly confused. So no BASIC words as variable names.

LOOPS Repeat Things

A "LOOP" is a section of a program that is designed to perform the same set of instructions (or "ROUTINE") over and over again. The number of times the routine is repeated can be specified or unspecified.

A computer will go on doing the same thing forever if you let it. That's why computers are given lots of boring, repetitive jobs, because they just don't get bored.

GOTO

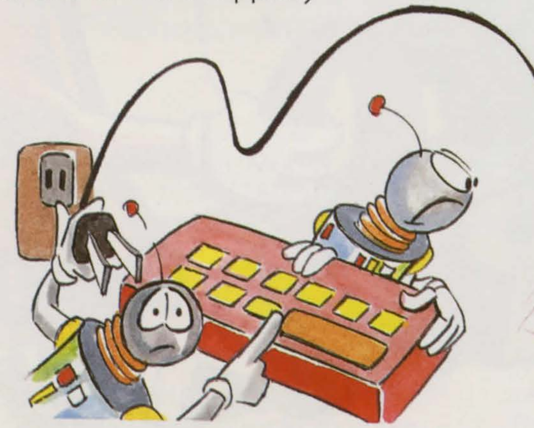
Like everything else with a computer, you must instruct it to go into a loop with a BASIC command word. There are several words that will do this; we'll look at the BASIC command "GOTO" first. GOTO in BASIC translates as "go to" in English.

What do you think the computer will do with the following program?

```
1Ø PRINT"HELLO"
2Ø GOTO 1Ø
```

Key it in (that's another way of saying type it in) and RUN it to see if you were right.

Good grief, what do you do now? It won't stop! You have put the computer in what is known as a "CONTINUOUS LOOP"** and the only ways out are (1) to turn off the computer, or (2) to press the BREAK key (or the ESCape or the RUN/STOP key, or CTRL and C on Apples.)



Press the BREAK key and see what happens.

**A continuous loop is one in which the number of repetitions is unspecified, so the computer just keeps doing it, and doing it, and doing it...

FOR, TO, NEXT

Let's look at another way of putting the computer into a loop – but this time we'll specify the number of repetitions! We'll use the BASIC words "FOR," "TO" and "NEXT." We'll need a variable to count the number of repetitions we specify. This type of variable is known as a "COUNTER VARIABLE," it counts. When the counter stops, the loop stops.

Here is an example:

command word

counter variable

command word

```
1Ø FOR I=1 TO 1Ø
2Ø PRINT"I'M IN A LOOP"
```

This line tells the computer to count from 1 to 10.

This is what you want repeated 10 times.

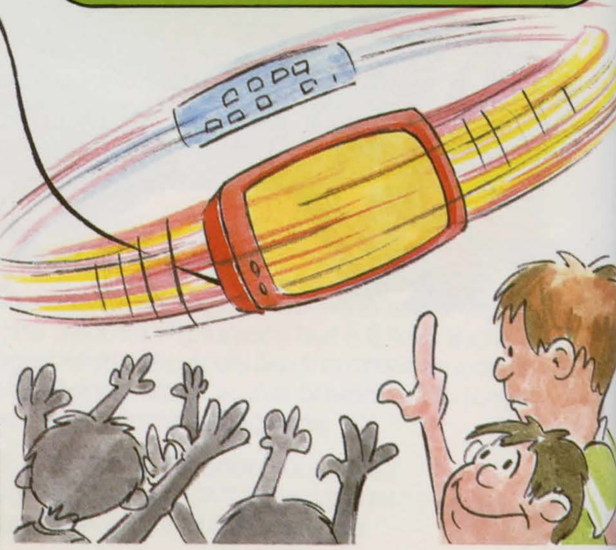
```
3Ø NEXT I
```

This tells the computer to keep going till it counts to 10.

Key in the program and RUN it.

This next example will show you exactly what the counter variable is doing. Try it.

```
1Ø FOR I=1 TO 1Ø
2Ø PRINT I
3Ø NEXT I
```



STEP

The computer counted to 10 by ones. If you wanted it to, it could count by 2's or 5's or by anything. We tell it to do this by using the BASIC word "STEP." For example:

```
FOR I=5 TO 50 STEP 5
```

This means, for a count from 5 to 50, STEP over every 5 – in other words, count by 5's.

Key in and RUN this program to see.

```
10 FOR I=5 TO 50 STEP 5
20 PRINT I
30 NEXT I
```

FOR YOU TO DO: Make the computer count from 10 to 100 by 10's.

STEP Backwards

OK, let's make the computer really do its stuff! Let's make it count backwards! We'll have to start at a big number and count to a smaller one. We'll still use STEP, but how will we show the computer we want it to go the other way? We'll use a negative number after STEP.

Remember this: With STEP, positives count forwards and negatives count backwards.

RUN this example and see:

```
10 FOR I=10 TO 1 STEP -1
20 PRINT I
30 NEXT I
```

DELAYS

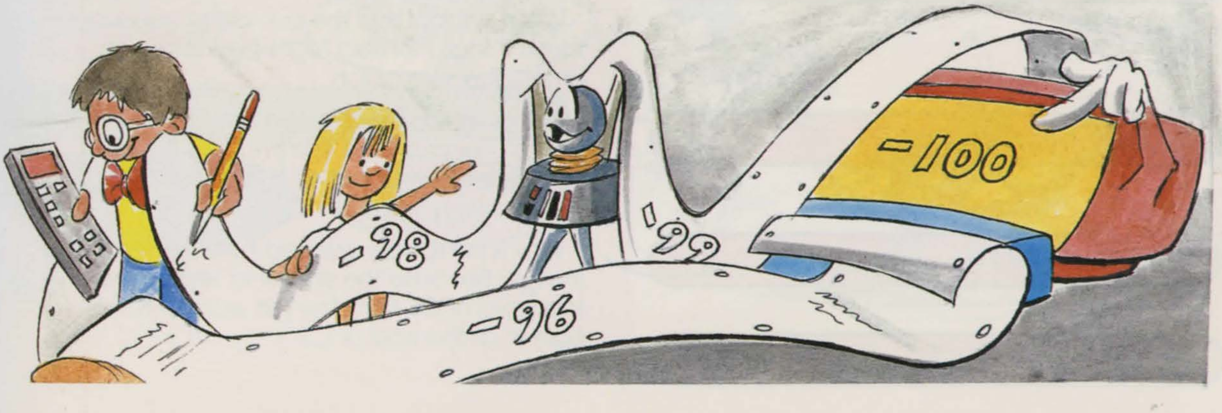
The computer is a very obedient thing; if there are no instructions in the FOR, TO, NEXT loop, nothing for the computer to do, it will do just that – nothing! This results in a "DELAY" while the computer sits there and just counts. Here's an example:

```
10 PRINT"WAIT 5 SECONDS"
20 FOR  I=1 TO 5000
30 NEXT I
40 PRINT"THANKS
   FOR THE REST"
```

No instructions to do anything, so the computer will only count.

NOTE: For every second of delay you want, make the computer count to 1000. This method works on most computers up to a delay of 15 seconds.

FOR YOU TO DO: Have the computer PRINT the word 'hello' and your name, then wait 10 seconds and PRINT 'goodbye' to you.



STRING VARIABLES

Way back in "YOUR FIRST COMMAND" we told you what a character string was. Do you remember? "A series of words or numbers in quotes" – right! In the "VARIABLES" section we looked at number variables. We had letters and numbers representing the values of other numbers. Now we're going to look at "STRING VARIABLES."

A string variable represents one or more words or numbers (in quotes) the same way that a number variable represents the value of a number. The computer needs to know which type of variable is which, so string variable names are followed by the dollar sign symbol ($). This symbol is read as "string." Here are some examples of string variables:

variable name

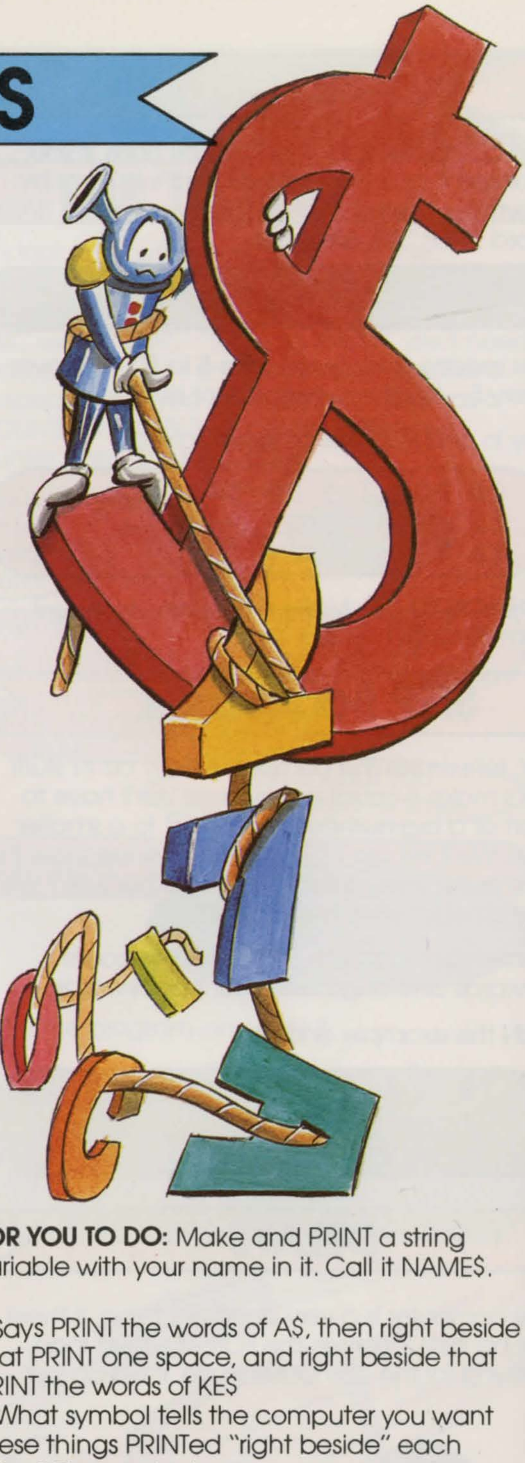
```
C$="CATS AND DOGS "
```

string symbol, so variable won't be taken for a number variable

words in quotes, the string

```
A$="HELLO"
K1$="ARE COMMON PETS"
KE$="HOW ARE YOU?"
```

Variable names can still be one or more letters, a combination of letters and numbers or whole words plus the string symbol.



FOR YOU TO DO: Make and PRINT a string variable with your name in it. Call it NAME$.

Like number variables, string variables can be PRINTed. Try these:

```
PRINT A$;" ";KE$
```

–Says PRINT the words of A$, then right beside that PRINT one space, and right beside that PRINT the words of KE$
– What symbol tells the computer you want these things PRINTed "right beside" each other? The semicolon.

```
PRINT A$,KE$
```

– Why is no space required here between the strings? Because the comma tells the computer to space over 10 spaces.

```
PRINT C$;K1$
```

– Why is no space required here between the strings? Because the space to separate the last word of C$ from the first word of K1$ is built in at the end of C$.

```
PRINT C$
```

Combined Strings Are Not Knots!

String variables can be combined. We'll use the same string variables.

```
Z$=C$+K1$
```

new string variable formed by combining two existing string variables

FOR YOU TO DO: Combine A$ with your NAME$ under the name N$.

String variables can be used in programs the same way number variables are. Enter and RUN this program:

```
1Ø INPUT"WHAT IS YOUR
   NAME ";NAME$
2Ø PRINT"HELLO ";NAME$
```

Remember that a string variable is a series of words or numbers in quotes. Write a program to INPUT today's date (just the month and date) as a string variable, and PRINT it.

LEFT$ and RIGHT$

There are BASIC command words that allow you to use only part of an existing string variable.

"LEFT$" lets you take as many characters as you want from the left side of the string. "RIGHT$" lets you do the same but starts from the right side of the string. To see how these commands work, enter these strings:

```
A$="SPRING FEVER"
N$="I HAVE A DOG"
D$="IT IS TIME TO START"
```

To PRINT only "SPRING" from A$, you would tell the computer to PRINT only the first six characters starting from the left. This is the command you would use.

command to PRINT something

tells the computer to start from the left side

```
PRINT LEFT$(A$,6)
```

tells the computer this is the string to use

use only the first 6 characters

Try it.

If you wanted to put the three right-most characters of N$ into a new variable called PET$, this is how you would do it:

start from the right side of a string

this is the string to use

```
PET$=RIGHT$(N$,3)
```

new string variable

take the first 3 characters

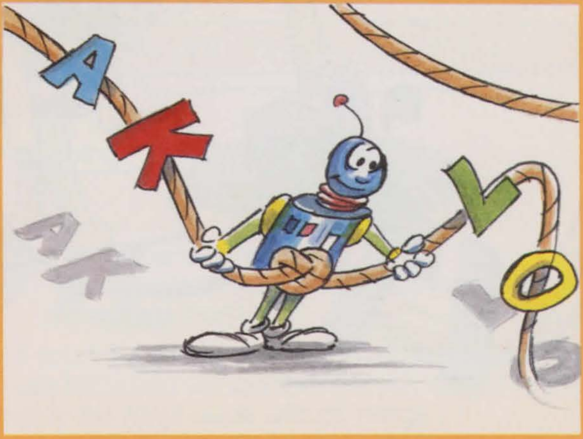
PRINT the new variable PET$.

Here are a few more examples to try that use the string variables given above. Use PRINT to see the results.

```
B$=LEFT$(N$,7)+A$
G$=LEFT$(D$,6)+RIGHT$(N$,5)
C$=LEFT$(N$,6)+RIGHT$
   (D$,9)+" FEEDING MY"+
   RIGHT$(N$,4)
```

FOR YOU TO DO: Using the proper commands and combinations of the above strings, PRINT a new variable called V$ that contains the string: "IT IS SPRING" (The answer is below.)

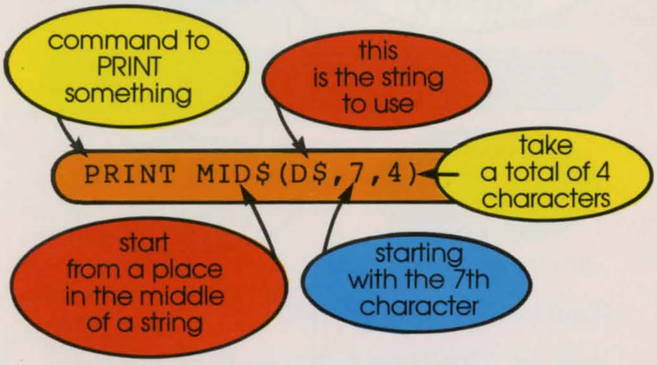
```
V$=LEFT$(D$,6)+LEFT$(A$,6)
PRINT V$
```



MID$

"MID$" is another BASIC command that lets you use only part of an existing string variable. With MID$ you specify what character you want to start taking characters from and how many you want to take.

Using our original D$="IT IS TIME TO START", try:


command to PRINT something
this is the string to use
PRINT MID$(D$,7,4)
take a total of 4 characters
start from a place in the middle of a string
starting with the 7th character

What command would PRINT "IS TIME TO" on the screen? (See answer below.)

```
PRINT MID$(D$,4,10)
```

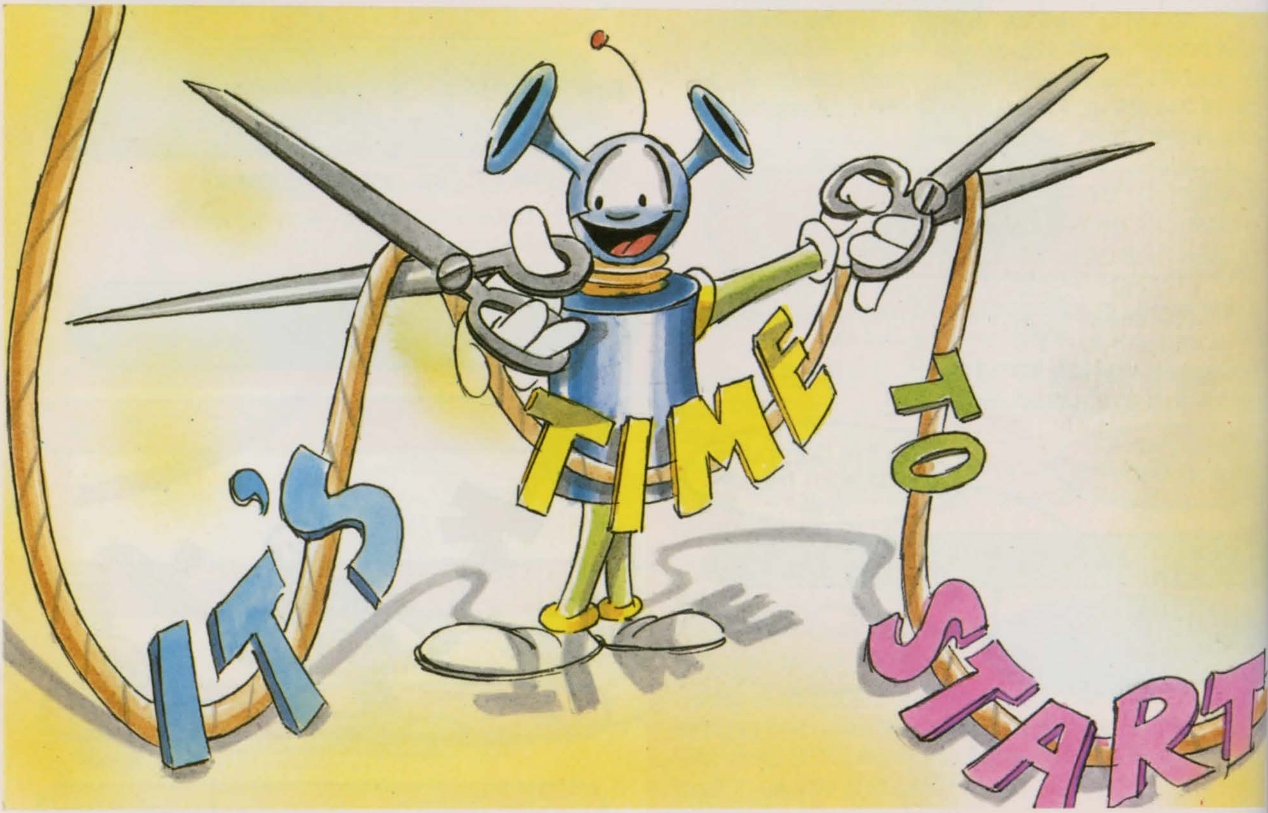
Using our original 3 string variables, A$, N$, and D$, enter the following:

```
PRINT MID$(N$,3,4)

PO$=MID$(A$,8,5)
PRINT PO$

Q$=MID$(D$,4,3)+LEFT$(D$,3)
   +LEFT$(A$,7)+MID$(D$,7,4)
PRINT Q$
```

Using everything you know so far about manipulating string variables, create one that represents this sentence:
SPRING IS THE TIME TO START TRAINING A DOG
(See answer below.)

Answer : There are many correct answers.
Here is one of them:

```
D1$=LEFT$(A$,7)+MID$(D$,4,2)
 +" THE "+RIGHT$(D$,13)+
" TRAINING "+RIGHT$(N$,5)
```



ATARI STRINGS

To use string variables with an ATARI computer, a dollar sign must follow the variable name and you must specify the maximum number of characters that the variable will hold. Specifying the number of characters is done with the BASIC word "DIM" which is short for the word DIMension. For example, to have a variable called C$ that will hold 30 characters you enter:

```
DIM C$(3Ø)
```

Now C$ can have a string assigned to it:

```
C$="CATS AND DOGS "
```

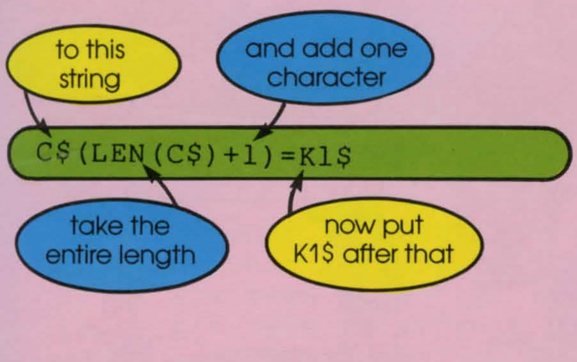
You can PRINT C$ like this:

```
PRINT C$
```

Here are some more examples of string variables and PRINT statements:

```
DIM K1$(2Ø),KE$(15),A$(25)

A$="HELLO"
K1$="ARE COMMON PETS"
KE$="HOW ARE YOU?"

PRINT A$
PRINT A$,KE$
```

Strings can be added together. K1$ is added to the end of C$ this way:



```
C$(LEN(C$)+1)=K1$
```

Another example is:

```
A$(LEN(A$)+1)=" "
A$(LEN(A$)+1)=KE$
PRINT A$
```

The result is "HELLO HOW ARE YOU".

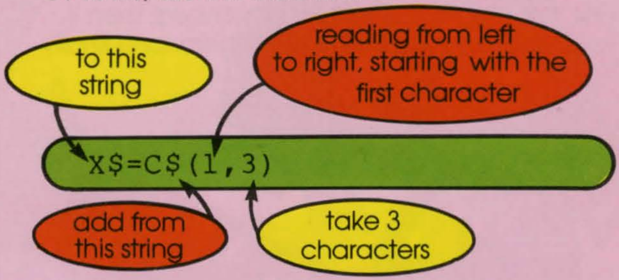
The ATARI can also use parts of a string, but it doesn't use LEFT$, RIGHT$ or MID$. When PRINTing the seventh to ninth characters of A$ you enter:

```
PRINT A$(7,9)
```

If you want to take "ARE" out of A$ and put them into X$, you need the eleventh to thirteenth characters, and you must use DIM to give X$ enough space to hold these characters:

```
DIM X$(3)
X$=A$(11,13)
```

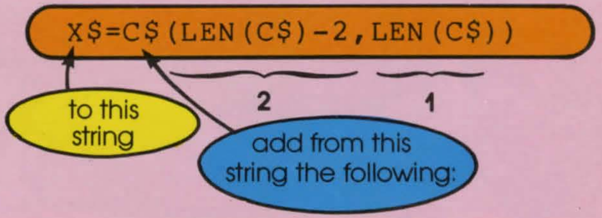
LEFT$ and RIGHT$ can be simulated on the ATARI. To assign the three left-most characters of C$ to X$, use this structure:



```
X$=C$(1,3)
```

to this string
reading from left to right, starting with the first character
add from this string
take 3 characters

If you want the three right-most characters of C$ in X$, use this:



```
X$=C$(LEN(C$)-2,LEN(C$))
```

1 – find the end of the string
2 – count back 2 more characters for a total of 3 and put them in X$

If you take a look at how the other computers use MID$ you will see that it is very similar to the way the ATARI deals with all strings.

IF THEN—Making Decisions

"IF" you want a program to make a decision "THEN" you must tell it what choices to expect and what it should then do. To do this use the BASIC command words IF and THEN.

Key in this example:

```
10 INPUT"IS TODAY YOUR
     BIRTHDAY (YES OR NO)";A$
20 IF A$="YES" THEN PRINT
   "HAPPY BIRTHDAY"
```

Key in "YES" and see what happens. What happens if you key in "NO"? Why? Nothing happens because the program doesn't tell the computer what to do if you choose "NO." Add this line:

```
30 IF A$="NO" THEN PRINT
   "TOO BAD"
```

In IF THEN situations the computer must recognize your response and decide what it should do next based on the instructions it was given.

Notice how you can jump to different lines of the program depending on your answers.

In this next program you choose between adding two numbers and subtracting them. First, the program asks for the numbers, then it asks what you want to do with them, and finally it gives you the proper answer. Here is the program:

```
10 INPUT"X";X
20 INPUT"Y";Y
30 PRINT"CHOOSE 1 OR 2:"
40 PRINT"1 - ADDITION (X+Y)"
50 PRINT"2 - SUBTRACTION
     (X-Y)"
60 INPUT A$
70 IF A$="1" THEN GOTO 100
80 IF A$="2" THEN GOTO 120
90 GOTO 60
100 Z=X+Y
110 GOTO 130
120 Z=X-Y
130 PRINT Z
```

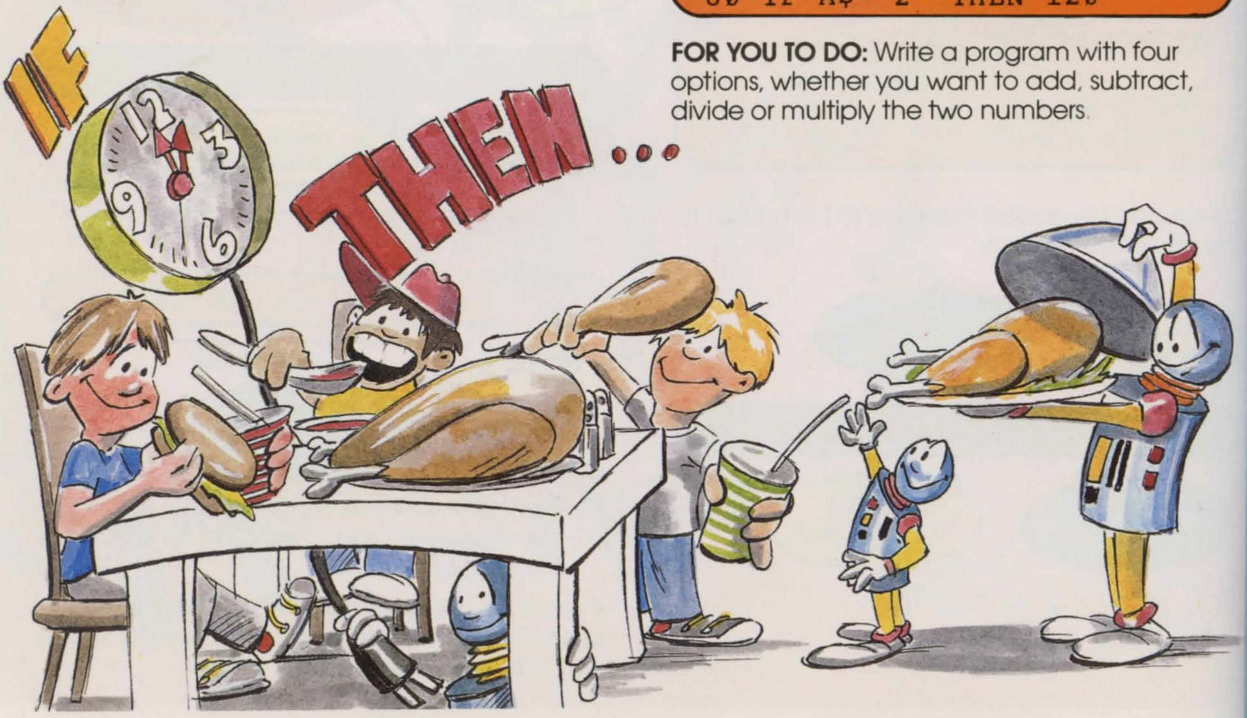
If addition is chosen, skip lines 80 and 90. This re-direction is called a "BRANCH."

To see what this line does, try INPUTing 3.

Most computers don't require GOTO after THEN when a branch is made. Lines 70 and 80 could be retyped to look like this:

```
70 IF A$="1" THEN 100
80 IF A$="2" THEN 120
```

FOR YOU TO DO: Write a program with four options, whether you want to add, subtract, divide or multiply the two numbers.



RELATIONAL OPERATOR

In the last example we had alternatives for the conditions A$="1" and A$="2". But variables don't always have to exactly equal something else.

What happens in the following example?

```
10 INPUT"ENTER YOUR AGE";AGE
20 IF AGE<16 THEN PRINT"YOU
    ARE NOT OLD ENOUGH TO
    DRIVE."
30 IF AGE>16 THEN PRINT"YOU
    ARE OLD ENOUGH TO
    DRIVE."
```

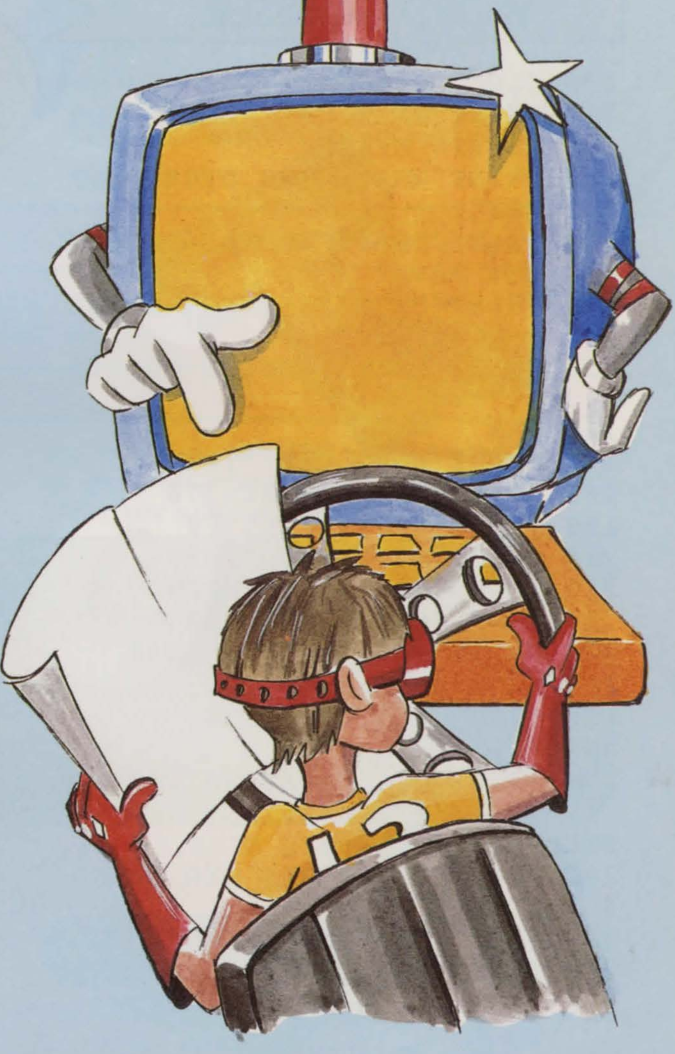
RUN the program. Experiment with the numbers you key in. What happens if you enter your age as 16? Why?

Add this line:

```
40 IF AGE=16 THEN PRINT"HAVE
   YOU GOT YOUR LICENSE YET?"
```

The equals sign, the less than sign and the greater than sign are known in the computer world as "RELATIONAL OPERATORS." Here is a list of relational operators and their meanings.

RELATIONAL OPERATOR	MEANING
=	equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
<>	not equal to



FOR YOU TO DO: Write a program that asks for options which include 3 or 4 of these relational operators:

Here's an example to get you started:

```
10 INPUT"WHAT IS YOUR AGE ";A
20 INPUT"WHAT IS YOUR BROTHER'S AGE ";B
30 IF A>B THEN PRINT"YOU ARE OLDER THAN YOUR BROTHER"
40 IF A<B THEN PRINT"YOUR BROTHER IS OLDER THAN YOU"
50 IF A=B THEN PRINT"YOU MUST BE TWINS!"
```

FUN TIME!

What a lot we've covered. It's time for a break. We have a word game for you and a friend. (The computer is the referee.) One of you enters three words into the computer while the other has his or her eyes closed. The words may be between 4 and 10 letters long. With your words entered, you're ready to play.

The computer will display the first three letters of the first word and the person who did not enter them must guess the words. Clues may be given by the enterer. The computer will allow you ten guesses per word. Good luck.

WORD GUESSER

This BASIC command word instructs the computer to CLear the Screen.

Check the Conversion Chart as this varies with some computers.

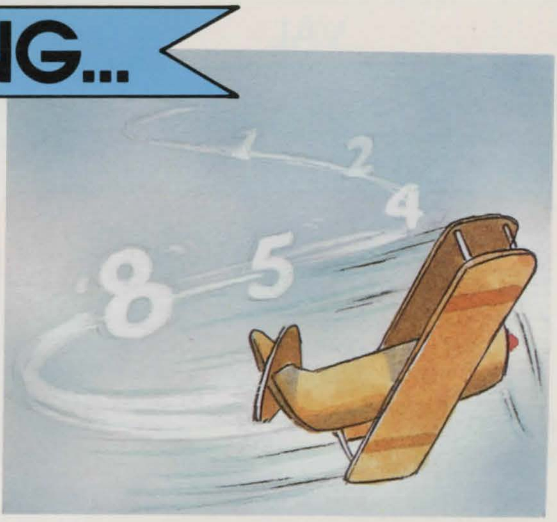
```
100 CLS
110 PRINT"HAVE ONE PLAYER CLO
    SE HIS EYES"
120 PRINT"WHILE THE OTHER ENT
    ERS"
130 PRINT"THREE WORDS. EACH M
    UST HAVE"
140 PRINT"BETWEEN FOUR AND TE
    N LETTERS."
150 INPUT"PRESS RETURN TO STA
    RT";Z$
160 CLS
170 INPUT"WORD ONE ";W1$
180 IFLEN(W1$)>1ØTHEN17Ø
190 IFLEN(W1$)<4THEN17Ø
200 INPUT"WORD TWO ";W2$
210 IFLEN(W2$)>1ØTHEN2ØØ
220 IFLEN(W2$)<4THEN2ØØ
230 INPUT"WORD THREE ";W3$
240 IFLEN(W3$)>1ØTHEN23Ø
250 IFLEN(W3$)<4THEN23Ø
260 CLS
270 PRINT"OK, HAVE THE OTHER
     PLAYER"
280 PRINT"OPEN HIS EYES."
290 PRINT"THERE ARE TEN GUESS
    ES"
300 PRINT"FOR EACH WORD."
310 PRINT
320 PRINT"WORD ONE STARTS WI
    TH "
330 PRINTLEFT$(W1$,3)
340 FORI=1TO1Ø
350 PRINT"GUESS #";I
360 INPUTG$
370 N=I
380 IFG$<>W1$THEN41Ø
390 PRINT"VERY GOOD! YOU GOT
     IT!"
400 I=1Ø
410 NEXTI
420 G=G+N
430 PRINT"WORD TWO STARTS WI
    TH "
440 PRINTLEFT$(W2$,3)
450 FORI=1TO1Ø
460 PRINT"GUESS #";I
470 INPUTG$
480 N=I
490 IFG$<>W2$THEN52Ø
500 PRINT"VERY GOOD! YOU GOT
     IT!"
510 I=1Ø
520 NEXTI
530 G=G+N
540 PRINT"WORD THREE STARTS WI
    TH "
550 PRINTLEFT$(W3$,3)
560 FORI=1TO1Ø
570 PRINT"GUESS #";I
580 INPUTG$
590 N=I
600 IFG$<>W3$THEN63Ø
610 PRINT"VERY GOOD! YOU GOT
     IT!"
620 I=1Ø
630 NEXTI
640 G=G+N
650 CLS
660 PRINT"THE WORDS WERE..."
670 PRINT
680 PRINT1,W1$
690 PRINT2,W2$
700 PRINT3,W3$
710 PRINT
720 PRINT"YOU TOOK ";G;" GUESS
    ES"
```

STRINGING ALONG...

You already know that a string is a collection of characters inside quotation marks. You also know how to combine strings by adding them together; and you can use the BASIC words, LEFT$, RIGHT$ AND MID$ to take strings apart. There are other BASIC words that manipulate strings, they are known as "STRING FUNCTIONS." Let's look at a few.

INKEY$

The BASIC word "INKEY$" allows you to go get a single character for use in a string. This may be a response to a "YES" or "NO" question using "Y" for yes and "N" for no, like this:



```
1Ø PRINT "ARE YOU OLD ENOUGH TO DRIVE? Y OR N?"
2Ø A$=INKEY$
3Ø IF A$="Y" THEN PRINT"YOU MUST BE 16 OR OLDER"
4Ø IF A$="N" THEN PRINT"YOU MUST BE UNDER 16"
5Ø GOTO 2Ø
```

Go get a one-character answer.

It has to be either Y or N.

If this program doesn't RUN properly, check the BASIC conversion chart on page 60 for commands specific to your computer.

You may also use INKEY$ to build a delay into a program. For example, a program will delay until a key is pressed. Here's one:

```
1Ø CLS
2Ø PRINT "THIS PROGRAM DEMONSTRATES HOW TO"
3Ø PRINT "MAKE THE COMPUTER WAIT UNTIL"
4Ø PRINT "A KEY HAS BEEN PRESSED"
5Ø PRINT
6Ø PRINT "PRESS A KEY TO CONTINUE"
7Ø A$=INKEY$
8Ø CLS
9Ø PRINT "THIS PROGRAM IS DONE"
```

Go get a one-character answer; it can be anything.

LEN

The BASIC word "LEN," short for LENgth, is used to count the number of characters in a string. It is used like this:

```
A$="BASIC IS A LANGUAGE"
PRINT LEN(A$)
```

the string to use

command to give the length of a string

Your string doesn't even need a variable name; but if it doesn't have a name, it must be in brackets, like this:

```
K=LEN("THIS IS A STRING")
PRINT K
```

The next little program will count the number of characters in any string:

```
1Ø INPUT"WHAT IS YOUR
     STRING ";S$
2Ø L=LEN(S$)
3Ø PRINT"THERE ARE ";L;"
     CHARACTERS IN ";S$
```

Try "TESTING 1 2 3". Now try strings of your own.

Write yourself a little program to add or subtract the number of characters in 2 strings.

VAL

The BASIC word "VAL," short for VALue, is a string function that assigns a numerical value to a number string. Here's how it works:

```
SC$="8Ø1Ø"
PRINT VAL(SC$)
```

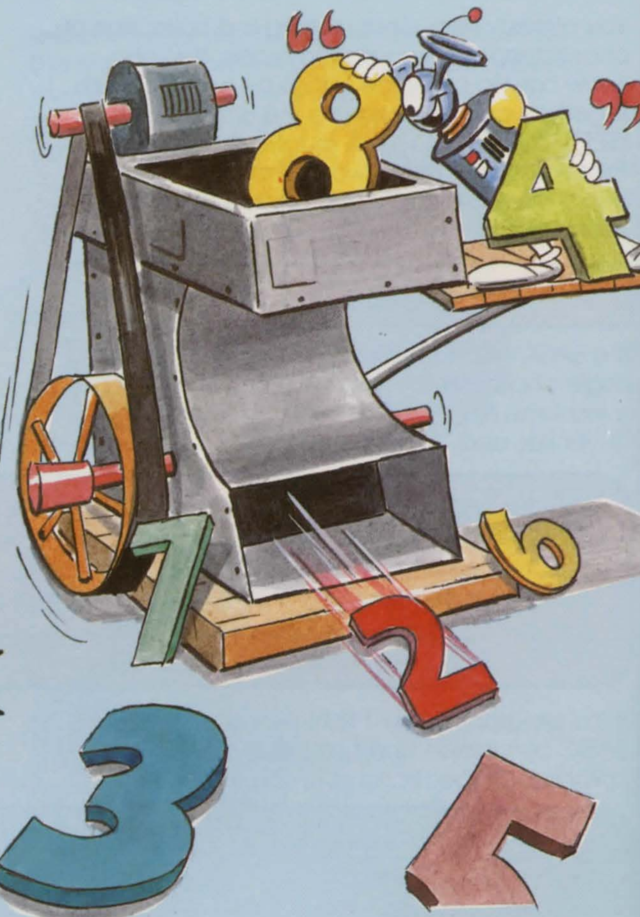
the number string to use

Give the numerical value of a number string.

```
S=VAL(SC$)*1Ø
PRINT S
```

Here's a little example to show you how VAL is used:

```
1Ø INPUT"WHAT'S YOUR AGE
   IN YEARS ";A$
2Ø PRINT"YOU ARE AT LEAST "
   ;12*VAL(A$);" MONTHS OLD"
```



STR$

The BASIC word "STR$," short for STRing plus the string symbol, functions the opposite to VAL. STR$ changes a number into a number string, like this:

Turn a number variable into a string variable.

```
C=23
B$=STR$(C)
```

new number string variable created

This is the name of the number variable to use.

Now PRINT B$.

```
X=-6
A$="THE ANSWER IS "
D$=A$+STR$(X)
PRINT D$
```

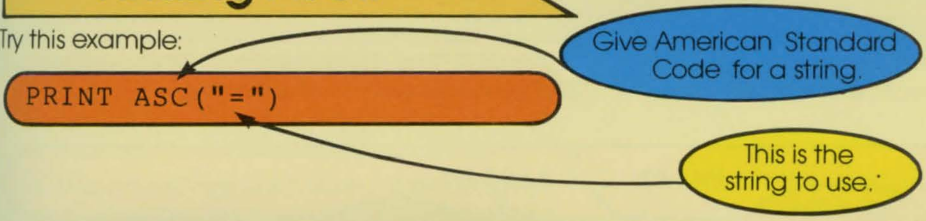
ASCII American Standard Code for Information Interchange

To the computer, every character has a numerical value. When you key in the letter A in a string, the computer reads it as 65. It reads B as 66, C as 67 and so on. The computer also reads numbers in a string, but again it reads the values it has already assigned to the numbers. Zero (Ø) is 48, 1 is 43, 2 is 50 and on it goes.

If you would like to know the value the computer has assigned to any character in a string, you can find out by using the BASIC string function "ASC." ASC is the BASIC short form for ASCII (pronounced ASH-KEY or ASK-KEY), which stands for the "American Standard Code for Information Interchange." These values are standard to most computers.

Finding ASCII

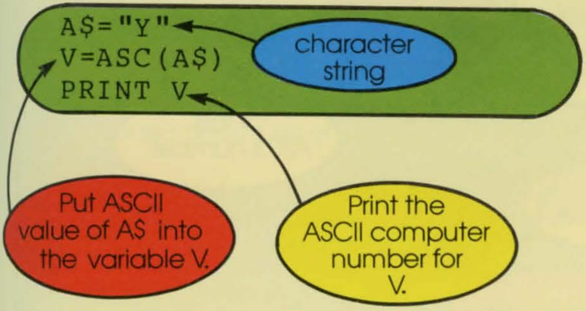
Try this example:



```
PRINT ASC("=")
```

Give American Standard Code for a string.

This is the string to use.



```
A$="Y"
V=ASC(A$)
PRINT V
```

character string

Put ASCII value of A$ into the variable V.

Print the ASCII computer number for V.

The numbers that the ASC command returns will normally be in the range of 0 to 255, although some computers only range from 0 to 127.

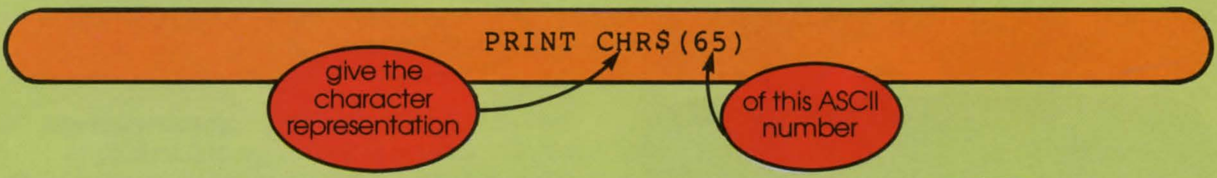
If you don't want to go through the above procedure every time you need an ASC number, you could keep this chart close to your computer and simply look them up as you need them. This is a chart of ASCII computer character numbers.

CHARACTER	ASCII NUMBER	CHARACTER	ASCII NUMBER	CHARACTER	ASCII NUMBER
space	32	4	52	H	72
!	33	5	53	I	73
"	34	6	54	J	74
#	35	7	55	K	75
$	36	8	56	L	76
%	37	9	57	M	77
&	38	:	58	N	78
'	39	;	59	O	79
(	4Ø	<	6Ø	P	8Ø
)	41	=	61	Q	81
*	42	>	62	R	82
+	43	?	63	S	83
,	44	@	64	T	84
–	45	A	65	U	85
.	46	B	66	V	86
/	47	C	67	W	87
Ø	48	D	68	X	88
1	49	E	69	Y	89
2	5Ø	F	7Ø	Z	9Ø
3	51	G	71		

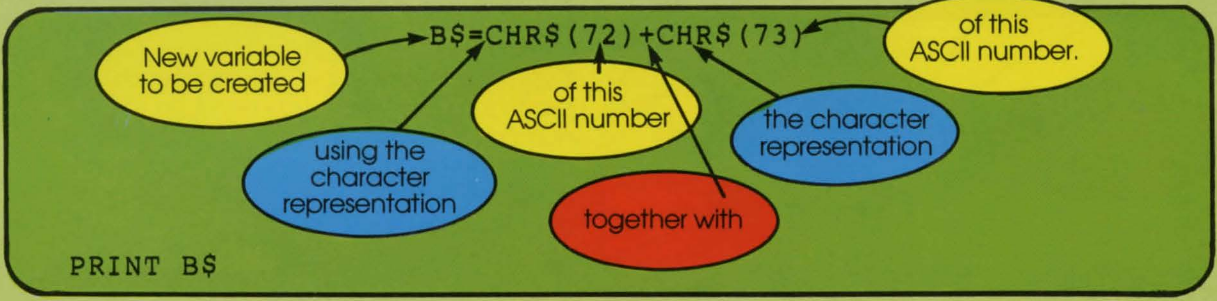
We said that the ASC command will return numbers from 0 to 255. "Where are all the rest?" you ask. The unlisted numbers have various assignments in different computers. Check your owner's manual to see if the rest of the numbers are listed for your computer.

CHR$

The BASIC word that does the opposite of ASC is "CHR$." CHR$ lets you work with characters (one at a time) by giving the computer their ASCII numbers. Enter these lines:



```
PRINT CHR$(65)
```



```
B$=CHR$(72)+CHR$(73)
PRINT B$
```

New variable to be created
using the character representation
of this ASCII number
together with
the character representation
of this ASCII number.

You can use CHR$ to see the characters not listed in the ASCII reference chart. For example:

```
PRINT CHR$(93)
```

String functions allow the easy manipulation of your character strings. Practice using them; they come in very handy in more advanced programming.

STRINGING THE ATARI ALONG

ATARI computers don't use the command INKEY$ to get a one-character response; instead they use the BASIC word "GET." Before GET can be used though, this line must appear in your program:

```
1Ø OPEN#1,4,Ø,"K:"
```

The key that is pressed will be assigned to a variable. As with all variables on the ATARI, this variable must first be DIMensioned:

```
2Ø DIM X$(1)
```

An actual GET statement looks like this:

```
3Ø GET#1,A
```

The variable used with the GET line is the numeric variable A, not X$, because the ATARI computer assigns to the variable the ASCII value of the response key. Now to put this ASCII value into your prepared string (see line 20) use this line:

```
4Ø X$=CHR$(A)
```

DATA and READ Statements

More Ways To Store Information.

DATA Statements

Variables and strings both store information for later use. Another method of storing information is to use the BASIC word "DATA." A DATA statement can store lots and lots of information until it is needed. Quite often the information is not needed all at once, or it is information that is only needed once. Lists of information are often stored this way because each piece of data can be taken and used separately, as needed. Computer quizzes, for example, make extensive use of DATA statements.

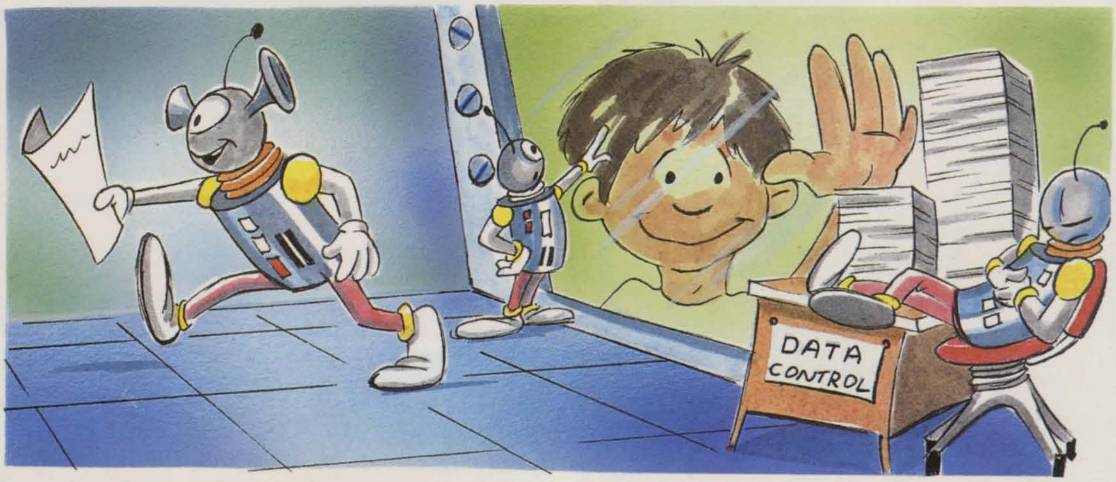
DATA statements are set up with commas between the various pieces of information, like this:

```
1Ø DATA FROG,2Ø,5
```

Use LIST to see the DATA statement. This one holds 3 pieces of information.

Notice in line 10 above that DATA statements hold strings as well as numbers. These strings don't even need quotes unless there are spaces or punctuation in them. Here are some more examples of DATA statements with strings:

```
5Ø DATA MICE,CAT,DOG
6Ø DATA HOT,"SUNNY AND DRY",COLD,"WET AND MILD"
7Ø DATA CARS,1Ø,TRUCKS,5
8Ø DATA "BUMPER STICKERS",1ØØ,WIPERS,2
```



READ Statements

You've used your DATA statements correctly and you've got all your information stored neatly away. Now how do you get it out to use it? Simple; you use the BASIC word "READ." Here is an example of a READ statement:

```
1ØØ READ Z$,Y,X
```

READ is followed by variables separated by commas the same way DATA was followed by pieces of information between commas. If you've figured out that their structures are related, you're right!

READ takes its first variable, in this case Z$, and goes looking in the first available DATA statement for something to put in that variable. Here READ would have found DATA line 10 first and would have assigned to Z$, the string of "FROG." READ would then take its second variable and assign it the value of the second piece of information in the DATA statement, and so on until it had used up or filled all of its variables.

Following the example, what would X be? If you said "5" from line 10, you're right.

READ Statements

When the READ statement is assigning values to variables and runs out of usable DATA in one line, it automatically jumps to the next DATA statement and keeps going.

READ uses both number variables and string variables, but the variables and information must match up and be of the same type or you'll come up with a TYPE MISMATCH ERROR.

To see how READ works with DATA, enter and run this short program:

```
100 DATA CATS,1Ø
12Ø READ A$
13Ø PRINT A$
14Ø READ X
15Ø PRINT X
```

Here's a slightly longer one:

```
1Ø DATA CATS,1Ø,DOGS,9
2Ø DATA MICE,3,SQUIRRELS,2
3Ø READ A$,X,B$,Y,C$,Z,D$,W
4Ø PRINT "ANIMAL BASEBALL
   LEAGUE"
5Ø PRINT
6Ø PRINT "SCORES:"
7Ø PRINT
8Ø PRINT A$;" - ";X
9Ø PRINT B$;" - ";Y
1ØØ PRINT
11Ø PRINT C$;" - ";Z
12Ø PRINT D$;" - ";W
```

Remember to READ numbers into number variables and strings into string variables!

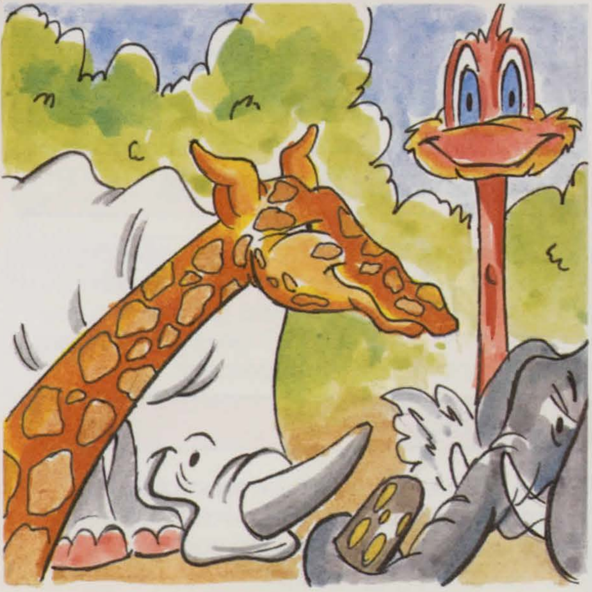
ANIMAL GUESSER

Now it's time for a quiz program using READ and DATA. You must guess the right animal. The computer will give you clues. Here's the program:

```
1Ø FOR I=1TO9
2Ø READ CLUE$,ANIMAL$
3Ø PRINT CLUE$
4Ø INPUT "WHAT IS THE ANIMAL ";REPLY$
5Ø IF REPLY$=ANIMAL$ THEN PRINT "VERY GOOD, YOU'RE RIGHT"
6Ø IF REPLY$<>ANIMAL$ THEN PRINT "WRONG - THE ANIMAL IS ";
   ANIMAL$
7Ø NEXT I
8Ø DATA "A HORSE IN PYJAMAS",ZEBRA,"IT LOVES HONEY",BEAR,"IT
    CHASES MICE",CAT
9Ø DATA "IT CHASES CATS",DOG,
   "IT EATS CHEESE",MOUSE,"IT
    EATS CARROTS",RABBIT
1ØØ DATA "IT EATS WORMS",BIRD,
   "IT BUILDS DAMS",BEAVER
11Ø DATA "IT LIVES IN WATER",
    FISH
```

In this program the DATA statements were at the end, but DATA statements can be placed anywhere in a program; the computer will always find them.

If you need all your information repeated, you can use the same DATA statements over again.



The BASIC word RESTORE sets this up with the computer. Add these lines to ANIMAL GUESSER and it will start over again using all the same DATA.

```
12Ø RESTORE
13Ø GOTO 1Ø
```

RANDOM NUMBERS

A random number is a number that is chosen for no reason. If someone said to you, "Pick a number," you would quickly say the first number that came into your head. There would have been no reason for its selection, you just needed a number.

Computer games and many other types of programs often use random numbers. The computer does the choosing. "RND," a contraction of the English word RaNDom, is the BASIC command word to produce random numbers.

To have the computer select and PRINT a random number between one and four, you would type:

```
PRINT RND(4)
```

Some computers use a slightly different command structure with RND. CHECK WITH THE CONVERSION CHART to see if your computer does it differently.

Say you want to flip a coin but you don't have a coin, you only have a computer. Well, get the computer to simulate coin flipping using the RND function. Let's call the number 1, heads and 2, tails. This line will flip your "coin":

```
PRINT RND(2)
```

FOR YOU TO DO: Write a line to simulate the rolling of a six-sided die.

Answer:

```
PRINT RND(6)
```

Aren't random numbers simple? You can use them when you write programs for games of chance, like the many card and dice games you've seen on computers. These games use the RND function. And many computer video games use RND for the unpredictable movement of aliens and other creatures.

FUN TIME

CAPITALS TESTER

Can you believe how much you've learned so far! INPUT allows almost any information to be entered during a program. Variables are used to manipulate information. Loops force the computer to do the same thing as often as required. String variables let you work with words. String functions let you work with strings. IF and THEN give the computer a way to make decisions. READ and DATA make use of all kinds of information. The computer will even pretend to roll a die for you with RND. Isn't that great?

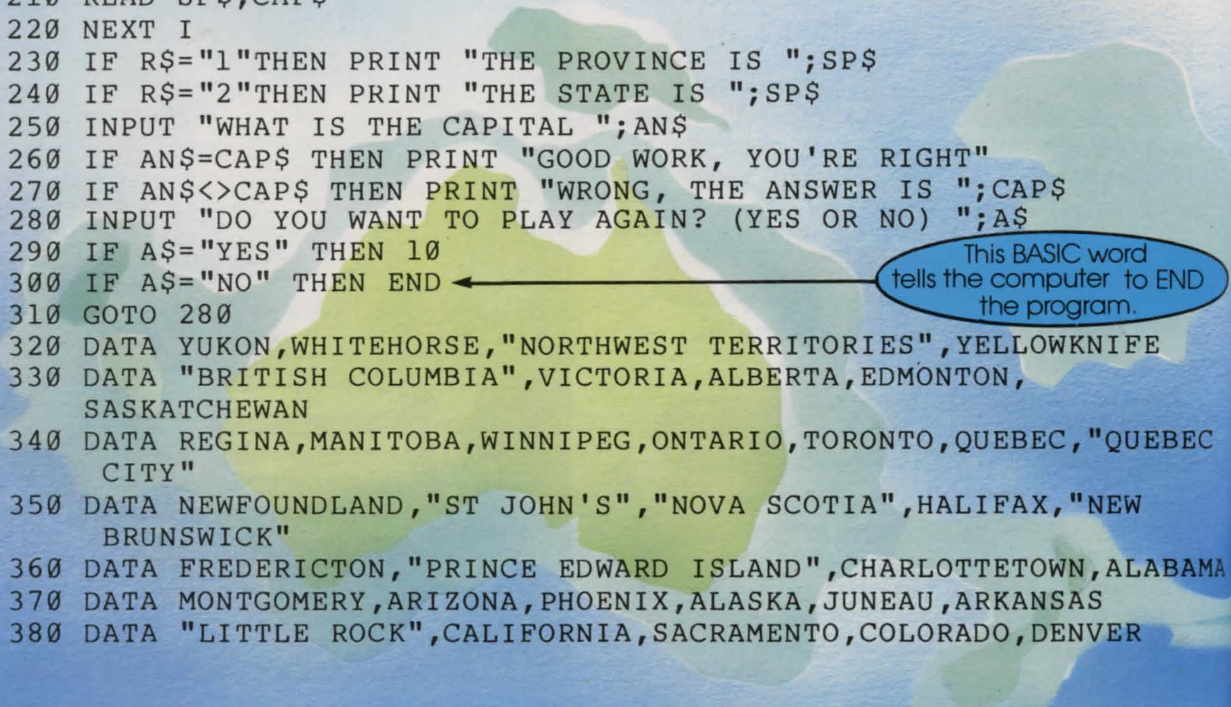
Well, learning is fun, but games are more fun! Let's do a game. The program CAPITALS TESTER given here uses most of these BASIC words and functions. The program will give you a Canadian province or an American state and you have to tell the computer the capital city. Good luck!

Remember to check the Conversion Chart for things like CLS, INKEY$ and random numbers.

ATARI users do not type in quotation marks in DATA statements lines 320 to 550.

```
1Ø CLS
15 RESTORE
2Ø PRINT "CHOOSE ONE:"
3Ø PRINT "1-CANADIAN PROVINCES"
4Ø PRINT "2-AMERICAN STATES"
5Ø R$=INKEY$
6Ø IF R$="1"THEN 1ØØ
7Ø IF R$="2"THEN 15Ø
8Ø GOTO 5Ø
1ØØ N=RND(12)
11Ø GOTO 2ØØ
15Ø N=RND(5Ø)+12
2ØØ FOR I=1TON
21Ø READ SP$,CAP$
22Ø NEXT I
23Ø IF R$="1"THEN PRINT "THE PROVINCE IS ";SP$
24Ø IF R$="2"THEN PRINT "THE STATE IS ";SP$
25Ø INPUT "WHAT IS THE CAPITAL ";AN$
26Ø IF AN$=CAP$ THEN PRINT "GOOD WORK, YOU'RE RIGHT"
27Ø IF AN$<>CAP$ THEN PRINT "WRONG, THE ANSWER IS ";CAP$
28Ø INPUT "DO YOU WANT TO PLAY AGAIN? (YES OR NO) ";A$
29Ø IF A$="YES" THEN 1Ø
3ØØ IF A$="NO" THEN END
31Ø GOTO 28Ø
32Ø DATA YUKON,WHITEHORSE,"NORTHWEST TERRITORIES",YELLOWKNIFE
33Ø DATA "BRITISH COLUMBIA",VICTORIA,ALBERTA,EDMONTON,
    SASKATCHEWAN
34Ø DATA REGINA,MANITOBA,WINNIPEG,ONTARIO,TORONTO,QUEBEC,"QUEBEC
     CITY"
35Ø DATA NEWFOUNDLAND,"ST JOHN'S","NOVA SCOTIA",HALIFAX,"NEW
     BRUNSWICK"
36Ø DATA FREDERICTON,"PRINCE EDWARD ISLAND",CHARLOTTETOWN,ALABAMA
37Ø DATA MONTGOMERY,ARIZONA,PHOENIX,ALASKA,JUNEAU,ARKANSAS
38Ø DATA "LITTLE ROCK",CALIFORNIA,SACRAMENTO,COLORADO,DENVER
```

This BASIC word tells the computer to END the program.



```
39Ø DATA CONNECTICUT,HARTFORD,DELAWARE,DOVER,FLORIDA
4ØØ DATA TALLAHASSEE,GEORGIA,ATLANTA,HAWAII,HONOLULU,IDAHO,BOISE,
    ILLINOIS
41Ø DATA SPRINGFIELD,INDIANA,INDIANAPOLIS,IOWA,"DES MOINES"
42Ø DATA KANSAS,TOPEKA,KENTUCKY,FRANKFORT,LOUISIANA,"BATON ROUGE"
43Ø DATA MAINE,AUGUSTA,MARYLAND,ANNAPOLIS,MASSACHUSETTS,BOSTON
44Ø DATA MICHIGAN,LANSING,MINNESOTA,"ST PAUL",MISSISSIPPI,JACKSON
45Ø DATA MISSOURI,"JEFFERSON CITY",MONTANA,HELENA,NEBRASKA
46Ø DATA LINCOLN,NEVADA,"CARSON CITY","NEW HAMPSHIRE",CONCORD
47Ø DATA "NEW JERSEY",TRENTON,"NEW MEXICO","SANTA FE","NEW YORK"
48Ø DATA ALBANY,"NORTH CAROLINA",RALEIGH,"NORTH DAKOTA",BISMARCK
49Ø DATA OHIO,COLUMBUS,OKLAHOMA,"OKLAHOMA CITY",OREGON,SALEM
5ØØ DATA PENNSYLVANIA,HARRISBURG,"RHODE ISLAND",PROVIDENCE
51Ø DATA "SOUTH CAROLINA",COLUMBIA,"SOUTH DAKOTA",PIERRE
52Ø DATA TENNESSEE,NASHVILLE,TEXAS,AUSTIN,UTAH,"SALT LAKE CITY"
53Ø DATA VERMONT,MONTPELIER,VIRGINIA,RICHMOND,WASHINGTON,OLYMPIA
54Ø DATA "WEST VIRGINIA",CHARLESTON,WISCONSIN,MADISON,WYOMING
55Ø DATA CHEYENNE
```

GRAPHICS

"GRAPHICS" is the name given to the pictures and patterns you've all seen on computer screens. Every computer has its own way of producing graphics. Some computers draw very detailed images using tiny dots that are called "PIXELS." Take a close look at your computer screen right now. Can you see the small dots that make up the letters and numbers? These are the pixels. Computers that make pictures using the dots are said to use "HIGH RESOLUTION GRAPHICS." But some computers draw pictures using BLOCKS of pixels. We'll look at those later.

When you put letters and numbers on the screen, your computer is said to be in "TEXT MODE," and when you are drawing pictures you are in "GRAPHICS MODE." Some computers with hi-res (You're a real computer buff when you use short forms!) have several different graphics modes. The difference between them usually has to do with the detail of the pictures and the number of colors you can use. You must instruct the computer as to which mode you want. Some typical BASIC commands for these are: "GRAPHICS," "GR," and "HGR." Once you have selected a mode you are ready to begin drawing.

There are a certain number of pixels on your screen, there are so many across in each row and there are just so many rows. You select which pixels you want to use in your drawing by calling for them by number. You need two numbers to call up one pixel; one number shows the row that the pixel is in and the other shows what column it is in. These two numbers are called "CO-ORDINATES." Check your owner's manual for the numbers you would use on your computer. (X is used to reference which row and Y is used to reference which column. You use them as X,Y. When you put in your own numbers it could be 10,4. That would be the tenth pixel over from the left in the fourth column).

The BASIC command to turn on or light up a pixel is "PLOT" (or sometimes "SET") followed by the co-ordinates of the desired pixel. On certain computers you must choose a color before you can PLOT a pixel. Each color your computer uses is also given a number reference. Your owner's manual will tell you more about the colors you can use. But often the command for choosing a color is simply "COLOR=" followed by the number reference of the color you want.

Let's see what we've learned so far with a little program. Actually four are given; the first is for Apple and Adam computers, the second is for Atari computers, and the last two are for Radio Shack TRS-80 Color Computers. We'll be drawing or PLOTting pixels.

APPLE and ADAM

```
1Ø INPUT "X (Ø-279) ";X1
2Ø INPUT "Y (Ø-159) ";Y1
3Ø INPUT "COLOR (Ø-7) ";C
4Ø HGR
5Ø HCOLOR=C
6Ø HPLOT X1,Y1
```

Type in "TEXT" to return to text mode.

ATARI

```
1Ø PRINT"PIXEL CO-ORDINATES"
2Ø PRINT"X (Ø-159)"
3Ø INPUT X
4Ø PRINT"Y (Ø-79)"
5Ø INPUT Y
6Ø PRINT"COLOR (Ø-255)"
7Ø INPUT C
8Ø GRAPHICS 7
9Ø COLOR C
1ØØ PLOT X,Y
```

Type in GRAPHICS Ø to return to text mode.

TRS-80 COLOR COMPUTER

```
10 INPUT"X (Ø-63)";X
20 INPUT"Y (Ø-31)";Y
30 INPUT"COLOR (Ø-7)";C
4Ø CLS(Ø)
5Ø SET(X,Y,C)
6Ø GOTO 6Ø
```

Hit the BREAK key to return to text mode.

RANDOM POINTS

If you don't want to figure out which pixels to light up, you can have the computer pick the points and colors itself. It will use its random function to select the number co-ordinates and colors. Here are some pixel drawing programs using random numbers. Remember to type in NEW before you start these.

APPLE and ADAM

```
2Ø X = INT(RND(1)*279+Ø.5)
3Ø Y = INT(RND(1)*159+Ø.5)
4Ø C = INT(RND(1)*7+Ø.5)
5Ø HGR
6Ø HCOLOR=C
7Ø HPLOT X,Y
```

TRS-80 COLOR COMPUTER WITH EXTENDED BASIC

```
1Ø X=RND(256)-1
2Ø Y=RND(192)-1
3Ø C=RND(8)-1
4Ø PMODE 3,1
5Ø SCREEN 3,1
6Ø PCLS
7Ø PSET (X,Y,C)
8Ø GOTO 8Ø
```

TRS-80 COLOR COMPUTER WITH EXTENDED BASIC

```
1Ø INPUT "X (Ø-255)";X
2Ø INPUT"Y (Ø-191)";Y
3Ø INPUT"COLOR (Ø-7)";C
4Ø PMODE 3,1
5Ø SCREEN 3,1
6Ø PCLS
7Ø PSET (X,Y,C)
8Ø GOTO 8Ø
```

Hit the BREAK key to return to text mode.

ATARI

```
3Ø X=INT(RND(1)*159+Ø.5)
5Ø Y=INT(RND(1)*79+Ø.5)
7Ø C=INT(RND(1)*255+Ø.5)
8Ø GRAPHICS 7
9Ø COLOR C
1ØØ PLOT X,Y
```

TRS-80 COLOR COMPUTER

```
1Ø X=RND(64)-1
2Ø Y=RND(32)-1
3Ø C=RND(8)-1
4Ø CLS(Ø)
5Ø SET (X,Y,C)
6Ø GOTO 6Ø
```

LINES

Most computers can draw lines with these BASIC commands: "PLOT," "LINE" or "DRAW." If your computer uses PLOT or LINE, you follow the command with the starting co-ordinate, followed by the word "TO" (or perhaps a dash; check your owner's manual) and finally the ending point co-ordinates. For computers that use DRAW (or possibly DRAWTO), you must first use PLOT to place the starting co-ordinate. DRAWTO is followed by the ending co-ordinate.

In these next three programs, you enter the start and end co-ordinates of a line and the computer draws the line.

APPLE and ADAM

```
1Ø PRINT "STARTING POINT CO-
   ORDINATES"
2Ø INPUT "X (Ø-279) ";X1
3Ø INPUT "Y (Ø-159) ";Y1
4Ø PRINT "ENDING POINT CO-OR
   DINATES"
5Ø INPUT "X (Ø-279) ";X2
6Ø INPUT "Y (Ø-159) ";Y2
7Ø INPUT "COLOR (Ø-7) ";C
8Ø HGR
9Ø HCOLOR=C
1ØØ HPLOT X1,Y1 TO X2,Y2
```

The TRS-80 COLOR COMPUTER without extended BASIC has no command to draw lines.

TRS-80 COLOR COMPUTER WITH EXTENDED BASIC

```
1Ø PRINT"STARTING POINT CO-O
   RDINATES"
2Ø INPUT"X (Ø-255)";X1
3Ø INPUT"Y (Ø-191)";Y1
4Ø PRINT"END POINT CO-ORDINA
   TES"
5Ø INPUT"X (Ø-255)";X2
6Ø INPUT"Y (Ø-191)";Y2
7Ø INPUT"COLOR (Ø-8)";C
8Ø PMODE 3,1
9Ø SCREEN 3,1
1ØØ PCLS
11Ø COLOR C,Ø
12Ø LINE (X1,Y1) - (X2,Y2),P
    SET
13Ø GOTO 13Ø
```

ATARI

```
1Ø PRINT"STARTING POINT CO-O
   RDINATES"
2Ø PRINT"X (Ø-159)"
3Ø INPUT X1
4Ø PRINT"Y (Ø-79)"
5Ø INPUT Y1
6Ø PRINT"END POINT CO-ORDINA
   TES"
7Ø PRINT"X (Ø-159)"
8Ø INPUT X2
9Ø PRINT"Y (Ø-79)"
1ØØ INPUT Y2
11Ø PRINT"COLOR (Ø-255)"
12Ø INPUT C
13Ø GRAPHICS 7
14Ø COLOR C
15Ø PLOT X1,Y1
16Ø DRAWTO X2,Y2
```

Type in GRAPHICS Ø to return to normal.

FOR YOU TO DO: Write a program that uses random numbers to select the starting and ending co-ordinates and the colors of your lines.

Hi-res

Hi-res graphics are useful in many ways: bar and line graphs can be drawn to show comparisons of data. The detailed action graphics of video games are often drawn this way, and design engineers can draw sophisticated cars, planes, bridges and much more.

These are the line changes required for random line and color selection:

TRS-80 COLOR COMPUTER WITH EXTENDED BASIC

```
20 X1=RND(256)-1
30 Y1=RND(192)-1
50 X2=RND(256)-1
60 Y2=RND(192)-1
70 C=RND(8)-1
80 PMODE 3,1
90 SCREEN 3,1
100 PCLS
110 COLOR C,0
120 LINE (X1,Y1)
    - (X2,Y2),PSET
130 GOTO 130
```

ATARI

```
30 X1=INT(RND(1)*159+0.5)
50 Y1=INT(RND(1)*79+0.5)
80 X2=INT(RND(1)*159+0.5)
100 Y2=INT(RND(1)*79+0.5)
120 C=INT(RND(1)*255+0.5)
130 GRAPHICS 7
140 COLOR C
150 PLOT X1,Y1
160 DRAWTO X2,Y2
```

APPLE and ADAM

```
20 X1=INT(RND(1)*279+0.5)
30 Y1=INT(RND(1)*159+0.5)
50 X2=INT(RND(1)*279+0.5)
60 Y2=INT(RND(1)*159+0.5)
70 C=INT(RND(1)*7+0.5)
80 HGR
90 HCOLOR=C
100 HPLOT X1,Y1 TO X2,Y2
```

OTHER GRAPHICS

Does your computer not have hi-res graphics commands? It can still draw pictures with regular characters. Just as there are rows and columns of pixels on your screen, there are also co-ordinates for characters. Each of these screen positions has a number. Look for these in your owner's manual under "SCREEN MEMORY MAPS." Every character that your computer can draw also has a number. Again, look in your manual, this time under "CHARACTER SETS."

POKE

Using the BASIC command "POKE," you can put any character you want anywhere on the screen. If you have a Commodore computer (Commodore 64, VIC 20, or PET) clear your screen and try one of these examples:

`POKE 1Ø24,16Ø`	**C64**
`POKE 768Ø,16Ø`	**VIC 20 with less than 8K of memory**
`POKE 4Ø96,16Ø`	**VIC 20 with 8K or more memory**
`POKE 32768,16Ø`	**PET**

Unless you're using a PET, you won't see anything yet. The other computers also need a color to be put on the screen. The place you tell it to put the color must be the same place you have put your character. The position used above was the first screen position, so the color position must also be the first. As with hi-res graphics, each color has a number. Check your owner's manual for these numbers.

Here are the POKE statements to enter a color into the first color position:

`POKE 55296,2`	**C64**
`POKE 384ØØ,2`	**VIC 20 with less than 8K of memory**
`POKE 37888,2`	**VIC 20 with 8K or more memory**

In the next program you may enter any screen position number, character number, color position and color number. It then POKEs the character into the screen position and the color into the color position. Following the program is a table of the numbers to enter for the various Commodore computers.

```
1Ø INPUT"POSITION";P
2Ø INPUT"CHARACTER";CH
3Ø INPUT"COLOR POSITION";C
5Ø INPUT"COLOR";CO
6Ø CLS
7Ø POKE P,CH
8Ø POKE C,CO
```

TABLE OF COMMODORE SCREEN AND COLOR POSITIONS

COMPUTER	START OF SCREEN POSITIONS	END OF SCREEN POSITIONS	START OF COLOR POSITIONS	END OF COLOR POSITIONS
C64	1024	2023	55296	56295
VIC 20 with less than 8K memory	7680	8185	38400	38905
VIC 20 with 8K or more memory	4096	4601	37888	38399
PET	32768	33767	the PET has no color	

PEEK

The BASIC command word "PEEK" is the opposite of POKE. PEEK is used to see what character is in a certain position on the screen or what color is in a color position. When you PEEK a screen or color location, you will get a number; this number will correspond to the set of numbers that you used with POKE.

If you have a Commodore computer, try one of the following lines to see the number value of the first character on the screen:

`PRINT PEEK(1Ø24)`	C64
`PRINT PEEK(768Ø)`	VIC 20 with less than 8K of memory
`PRINT PEEK(4Ø96)`	VIC 20 with 8K or more of memory
`PRINT PEEK(32768)`	PET

Positions to be PEEKed must always be in round brackets!
You probably saw a value of 32. 32 is the value for a space on Commodore computers.

To find the number value of the first color position on the Commodore 64 or VIC, key in one of these lines:

`PRINT PEEK(55296)`	C64
`PRINT PEEK(384ØØ)`	VIC 20 with less than 8K of memory
`PRINT PEEK(37888)`	VIC 20 with 8K or more of memory

POKE and PEEK are not only used with graphics; they can do many things for you. The extent of what they can do depends on which computer you are using and the values you give them to work with. Look in your owner's manual for more information on how to use POKE and PEEK in your computer.

LOOP d'LOOP

We've covered a lot of ground so far, so maybe you should go back and review LOOPS, because now we're going to build on that knowledge.

It is possible to place loops within loops. "Why?" you ask. With 2 counters ticking away, you can keep track of 2 different things at the same time.

Putting a loop inside another loop is known as "NESTING." The first thing you have to do is give each loop a separate counter or the computer will really become confused. The inner loop is the nested loop. Enter and RUN this example:

```
10 FOR I=1TO5
20 FOR J=1TO7
30 PRINT "I=";I;" J=";J
40 NEXT J
50 NEXT I
```

First loop
Second loop

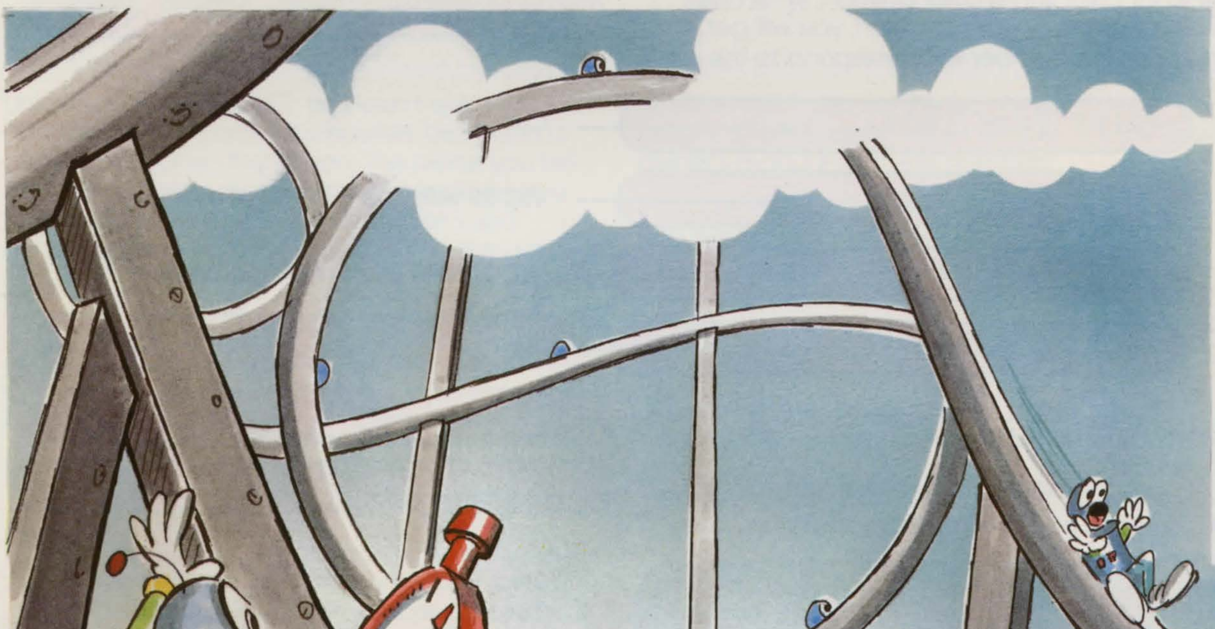
This next program is a practical example of nested loops; it simulates a stop watch.

```
10 FOR M=ØTO59
20 FOR S=ØTO59
30 PRINT M;":";S
40 NEXT S
50 NEXT M
```

To make your stop watch tick off real seconds, add these lines:

```
35 FOR I=1 TO 1ØØØ
37 NEXT I
```

This program will now RUN for an hour if you let it. To stop it, hit the BREAK key.



Now try these examples:

```
10 FOR I=1TO5
20 PRINT
30 FOR J=1TO5
40 PRINT SPC(J)"BASIC IS FUN"
50 NEXT J
60 NEXT I
```

ATARI users see page 60.

This is a BASIC word short for SPaCe. It tells the computer to insert as many spaces as are in the brackets.

```
10 FOR I=1TO1Ø
20 FOR J=1TO6
30 PRINT SPC(J);"COMPUTERS"
40 NEXT J
50 PRINT "...ARE FAST"
60 NEXT I
```

If you have graphics commands, try one of these programs:

TRS-80 COLOR COMPUTER

```
1Ø CLS(Ø)
2Ø FOR Y=Ø TO 31
3Ø FOR X=Ø TO 63
4Ø C=RND(8)-1
5Ø SET(X,Y,C)
6Ø NEXT X
7Ø NEXT Y
8Ø GOTO 8Ø
```

Remember to hit the BREAK key to get out of the program.

ATARI

```
1Ø GRAPHICS 7
2Ø FOR Y=Ø TO 79
3Ø FOR X=Ø TO 159
4Ø COLOR INT(RND(1)*255+Ø.5)
5Ø PLOT X,Y
6Ø NEXT X
7Ø NEXT Y
```

Remember to key in GRAPHICS Ø to return to text mode.

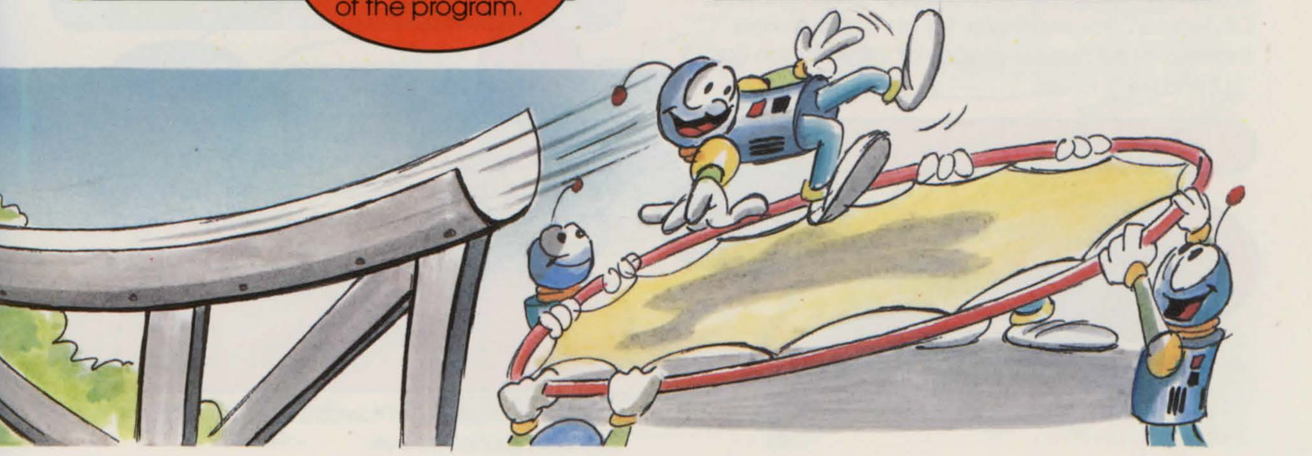
APPLE and ADAM

```
1Ø HGR
2Ø FOR Y=ØTO159
3Ø FOR X=ØTO279
4Ø HCOLOR=INT(RND(1)*7+Ø.5)
5Ø HPLOT X,Y
6Ø NEXT X
7Ø NEXT Y
```

Remember to type in TEXT to get out of the graphics mode.

TRS-80 COLOR COMPUTER WITH EXTENDED BASIC

```
1Ø PMODE 3,1
2Ø SCREEN 3,1
3Ø PCLS
4Ø FOR Y=Ø TO 255
5Ø FOR X=Ø TO 191
6Ø C=RND(8)-1
7Ø PSET(X,Y,C)
8Ø NEXT X
9Ø NEXT Y
1ØØ GOTO 1ØØ
```



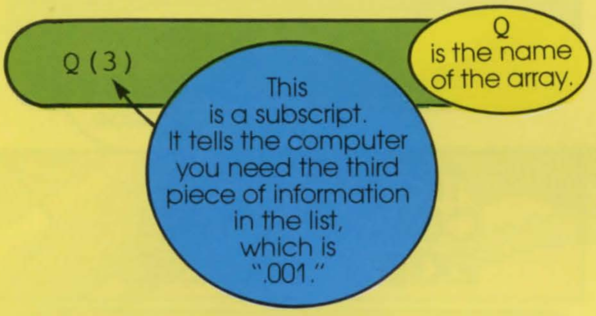
ARRAYS

An "ARRAY" is a group of related pieces of information stored under one variable name.

Q

174	365	.001	–10,000
1	2	3	4

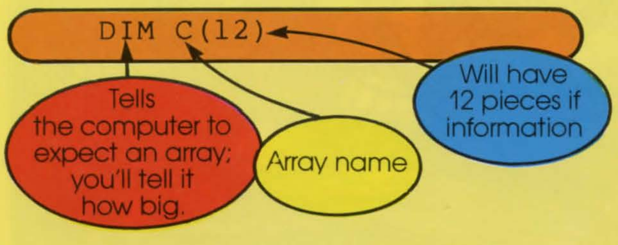
Arrays have names, like other variables, but they also have "SUBSCRIPTS." A subscript, sometimes called an "INDEX," tells the computer exactly which piece of information is required. Subscripts refer to information, or information is "REFERENCED" by subscripts.

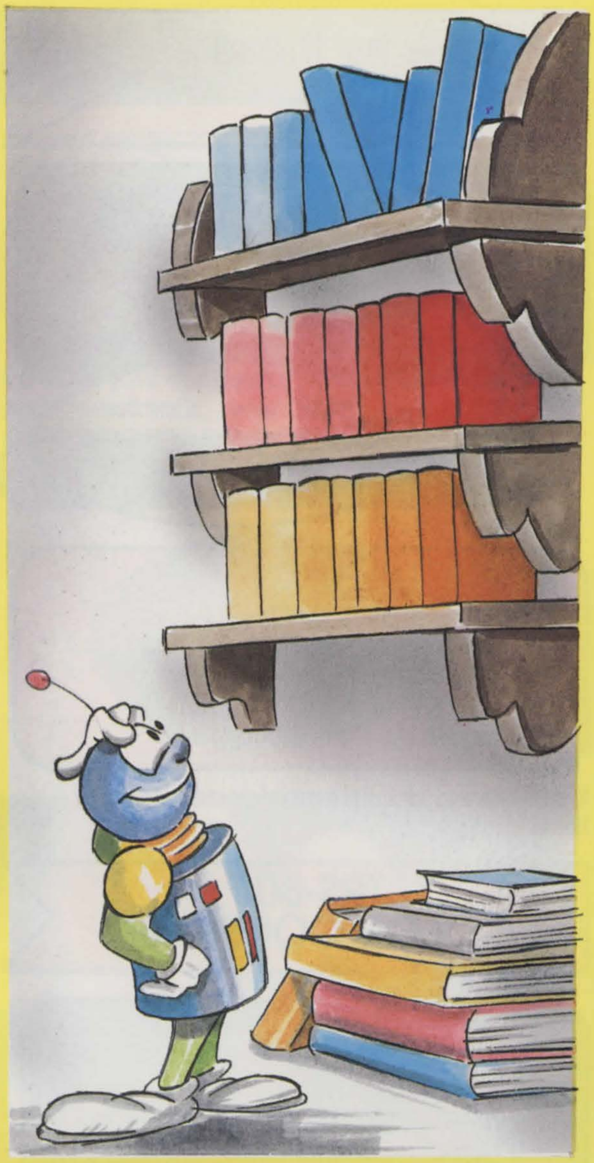

Q(3)
Q is the name of the array.
This is a subscript. It tells the computer you need the third piece of information in the list, which is ".001."

What is referenced by Q(1)? The first piece of information or the first "ELEMENT" in the list.

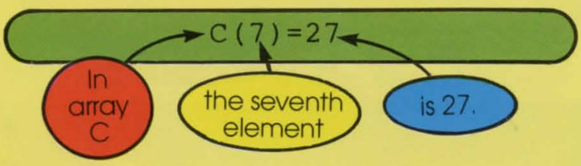
DIM

You can use arrays to store information. But how much information? That's the first thing your computer will want to know too. You tell it how much by using the BASIC word "DIM," short for DIMension. For example, for an array called "C" that will hold twelve pieces of information (or 12 "ELEMENTS"), enter:


DIM C(12)
Tells the computer to expect an array; you'll tell it how big.
Array name
Will have 12 pieces if information



Suppose you wanted the seventh element in C to be 27. You would tell the computer:



If you don't give a value to any of your elements, the computer will. It will give a value of zero to any unassigned elements. See your computer do this by typing:

```
PRINT C(1),C(2),C(3),C(4),
      C(5),C(6),C(7),C(8),
      C(9),C(1Ø),C(11),C(12)
```

The seventh element was 27, just like you told the computer above.

Using ARRAYS

Arrays are useful when you have a list of similar things to keep track of. If you wanted to keep a list of your family's birth years, an array would be a good thing to keep them in. Yes, it's time for another program. This one lets you enter three birth years in an array. Then it asks for the current year. With this information it calculates the person's age. Here is the program:

```
10 DIM BY(3)
20 INPUT "BIRTH YEAR NUMBER
   1 ";BY(1)
30 INPUT "BIRTH YEAR NUMBER
   2 ";BY(2)
40 INPUT "BIRTH YEAR NUMBER
   3 ";BY(3)
50 INPUT "PRESENT YEAR ";Y
60 PRINT "THE CURRENT AGES
   ARE:"
70 PRINT Y-BY(1),Y-BY(2),
   Y-BY(3)
```

ATARI users see page 60.

Arrays are easily handled with loops. Since the only difference in the three INPUT lines is the subscript, a loop could count through them. Here's the same program using a loop:

```
10 DIM BY(3)
20 FOR I=1 TO 3
30 PRINT "BIRTH YEAR #";I
40 INPUT BY(I)
50 NEXT I
60 INPUT "CURRENT YEAR ";Y
70 PRINT "THE AGES ARE:"
80 FOR I=1 TO 3
90 PRINT Y-BY(I),
100 NEXT I
```

ATARI users see page 60.

FOR YOU TO DO: Modify the program to accept 5 birth years and then PRINT the 5 ages.

These are the lines and changes you would have to make:

ANSWER:

```
10 DIM BY(5)
20 FOR I=1 TO 5
80 FOR I=1 TO 5
```

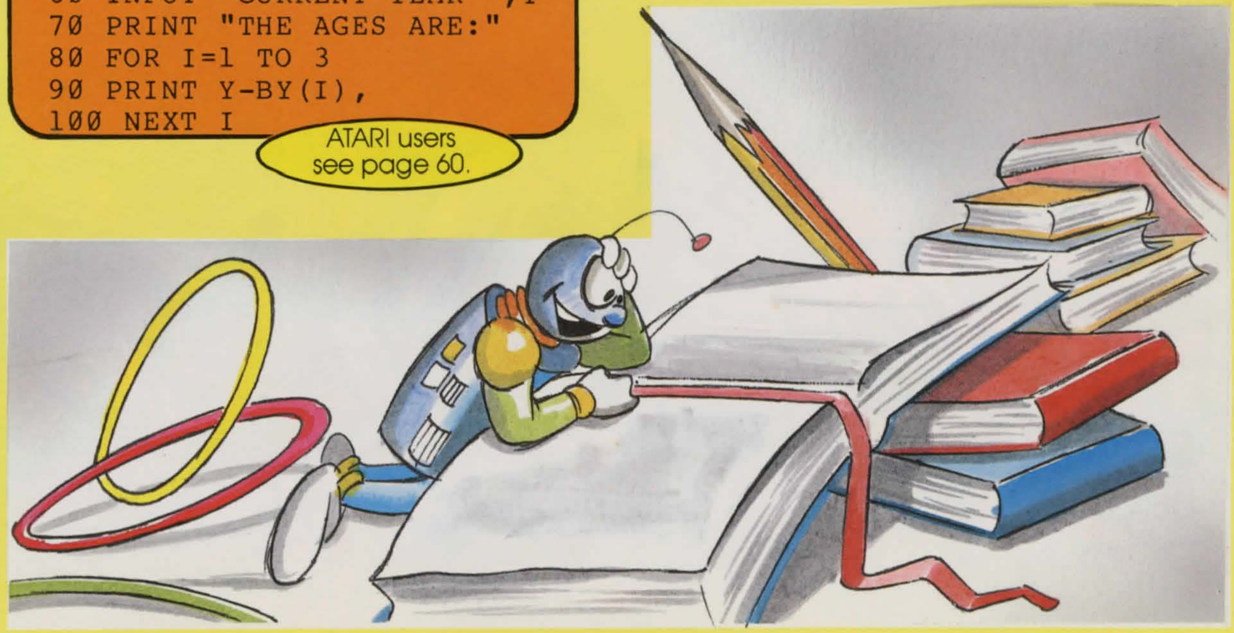
Say you're going to round up all your friends and enter all their birth years, but you don't know how many friends you're going to be able to find on such short notice. Don't worry, you can set up the program to accept as many birth years as you may need. Simply add this line as line 5:

```
5 INPUT "NUMBER OF BIRTH
  YEARS ";N
```

And change lines 10, 20 and 80 to read:

```
10 DIM BY(N)
20 FOR I=1 TO N
80 FOR I=1 TO N
```

Now you can amaze as many friends as you can find.



SUBROUTINES

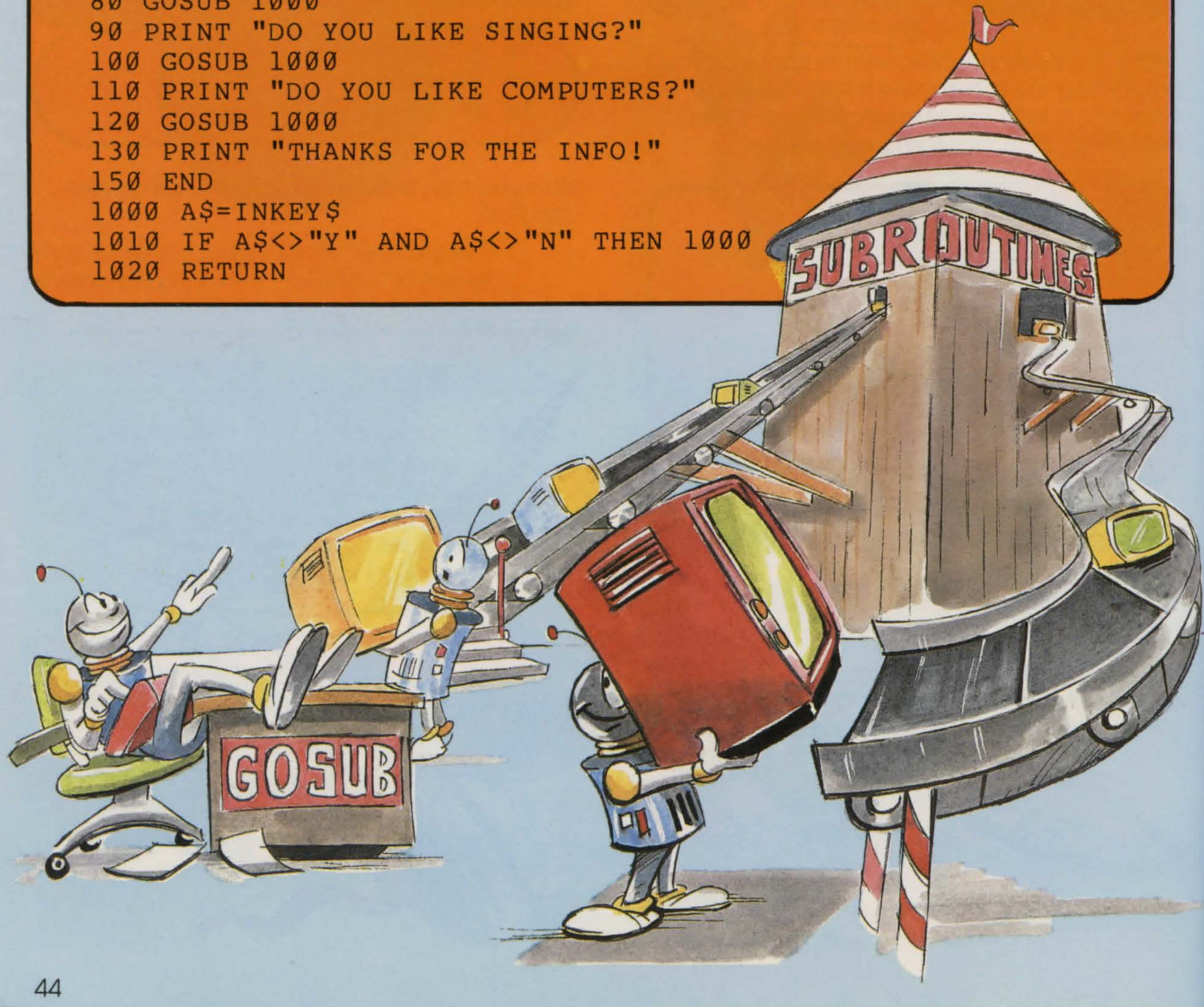
A subroutine is a part of a longer program; it does one thing. It could actually be a short program on its own. Subroutines can be entered once and used as many times as you want. They do simple and frequently needed tasks.

You may have a lot of "YES" and "NO" questions in a program. By using a subroutine you can get answers to all the questions with one INPUT or INKEY$. This next program, SILLY INTERVIEW, uses an INKEY$ subroutine.

SILLY INTERVIEW

```
5 PRINT"ANSWER EACH QUESTION Y FOR YES, N FOR NO"
1Ø PRINT "DO YOU LIKE TV?"
2Ø GOSUB 1ØØØ
3Ø PRINT "DO YOU LIKE TENNIS?"
4Ø GOSUB 1ØØØ
5Ø PRINT "DO YOU LIKE HOCKEY?"
6Ø GOSUB 1ØØØ
7Ø PRINT "DO YOU LIKE SWIMMING?"
8Ø GOSUB 1ØØØ
9Ø PRINT "DO YOU LIKE SINGING?"
1ØØ GOSUB 1ØØØ
11Ø PRINT "DO YOU LIKE COMPUTERS?"
12Ø GOSUB 1ØØØ
13Ø PRINT "THANKS FOR THE INFO!"
15Ø END
1ØØØ A$=INKEY$
1Ø1Ø IF A$<>"Y" AND A$<>"N" THEN 1ØØØ
1Ø2Ø RETURN
```

Are you checking the Conversion Chart to find any changes for your computer?



SUBROUTINES

The BASIC word "GOSUB" directs the computer to your subroutine. Then after the instructions of the subroutine have been performed, the BASIC word "RETURN" sends the computer back to the main program, to the line immediately after the GOSUB. Here's another example:

```
10 PRINT "A SUBROUTINE
    DEMONSTRATION"
20 GOSUB 1000
30 IF R$<>"Y"THEN10
40 END
1000 PRINT "ARE YOU FINISH
     ED? (TYPE Y OR N)"
1010 R$=INKEY$
1020 RETURN
```

How does the second program work? Line 10 PRINTs a message. In line 20 GOSUB sends the computer to line 1000. Line 1000 is where the subroutine starts. There a question is asked. Line 1010 gets a response from the keyboard. The subroutine is then done so line 1020 sends the computer back to the line after GOSUB, line 30. On line 30 the computer checks the answer. If the response was "Y" the program ends; otherwise it goes back to line 10 and starts again.

The subroutines in the demonstration programs were short and simple. They did one thing; they waited for answers to a question. Subroutines should always be as short as possible and do only one thing.

Subroutines are best used in repetitive or complicated programs. When you use subroutines, your programs are shorter and easier to understand. It is also easier to trace a problem in a program that uses subroutines than it is in a very long, unco-ordinated program.

This next example uses 2 subroutines. One subroutine waits for a key to be pressed, the other is a delay loop. RUN it and examine it.

```
5 CLS
10 PRINT "THIS PROGRAM SHOWS"
20 PRINT "HOW A SUBROUTINE CAN BE"
30 PRINT "USED MORE THAN ONCE."
40 GOSUB 1000
45 CLS
50 PRINT "BY USING THE SAME SUBROUTINE"
60 PRINT "YOUR PROGRAMS CAN BE MADE"
70 PRINT "MUCH SHORTER AND SIMPLER"
80 GOSUB 1000
90 CLS
100 PRINT "YOU HAVE FIVE SECONDS TO"
110 PRINT "READ THIS SCREEN!"
120 GOSUB 1100
130 CLS
140 PRINT "AS YOU CAN SEE, SUBROUTINES"
150 PRINT "SAVE TIME AND SPACE"
160 GOSUB 1100
170 CLS
180 PRINT "HAVE FUN WITH BASIC!"
190 END
1000 PRINT "PRESS A KEY TO CONTINUE"
1010 A$=INKEY$
1020 RETURN
1100 FOR I=1TO5000
1110 NEXT I
1120 RETURN
```

SUBROUTINES FOR RANDOM LINES

Remember the program in the GRAPHICS section that drew random lines on the screen? You could use a subroutine to do that. Enter one of these programs and see.

APPLE and ADAM

```
20 X1=INT(RND(1)*279+Ø.5)
3Ø Y1=INT(RND(1)*191+Ø.5)
4Ø X2=INT(RND(1)*279+Ø.5)
5Ø Y2=INT(RND(1)*191+Ø.5)
6Ø C=INT(RND(1)*7+Ø.5)
7Ø GOSUB 11Ø
8Ø GET A$
9Ø TEXT
1ØØ STOP
11Ø HGR2
12Ø HCOLOR=C
13Ø HPLOT X1,Y1 TO X2,Y2
14Ø RETURN
```

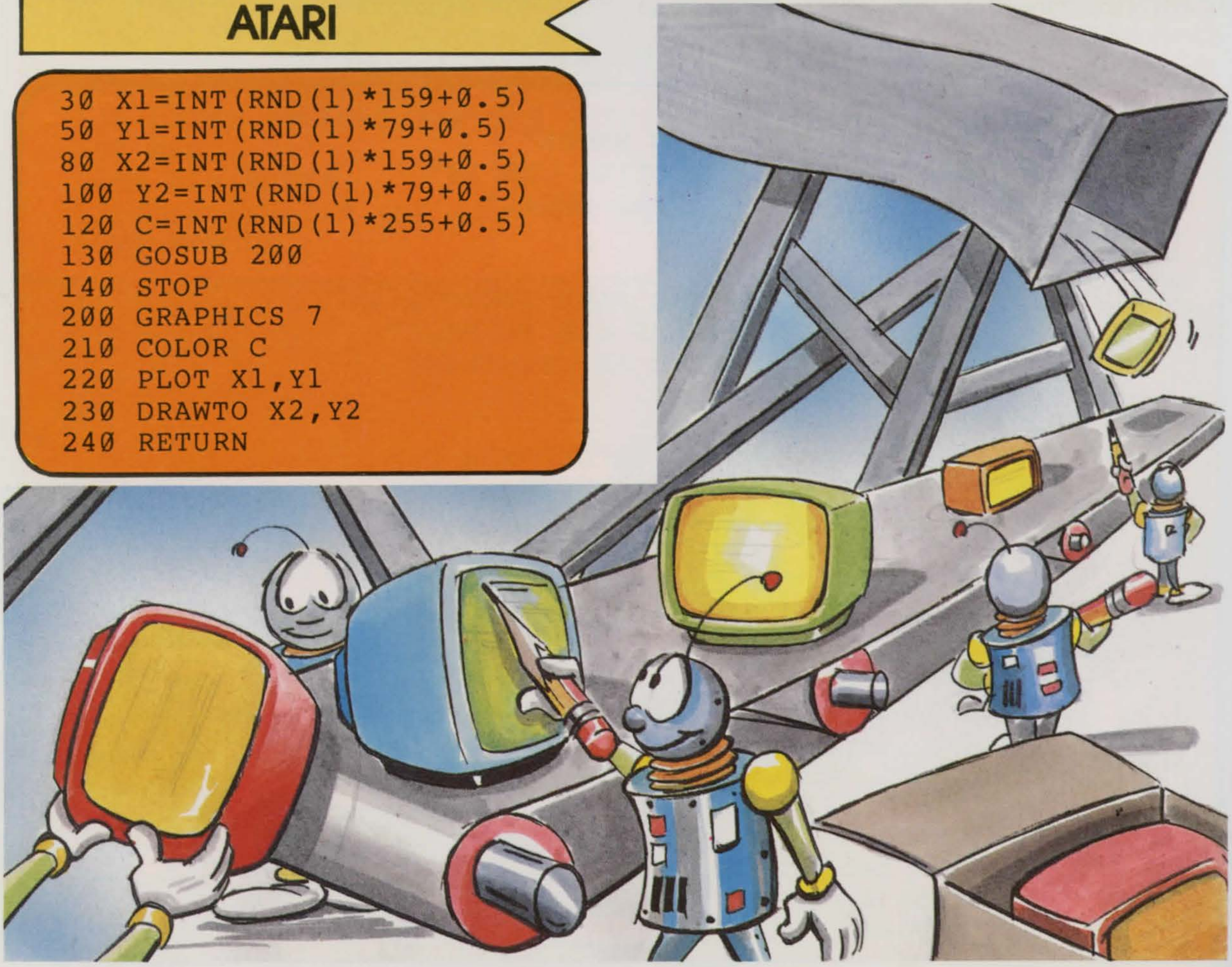
ATARI

```
3Ø X1=INT(RND(1)*159+Ø.5)
5Ø Y1=INT(RND(1)*79+Ø.5)
8Ø X2=INT(RND(1)*159+Ø.5)
1ØØ Y2=INT(RND(1)*79+Ø.5)
12Ø C=INT(RND(1)*255+Ø.5)
13Ø GOSUB 2ØØ
14Ø STOP
2ØØ GRAPHICS 7
21Ø COLOR C
22Ø PLOT X1,Y1
23Ø DRAWTO X2,Y2
24Ø RETURN
```

The TRS-80 COLOR COMPUTER without extended BASIC has no command to draw lines.

TRS-80 COLOR COMPUTER WITH EXTENDED BASIC

```
2Ø X1=RND(256)-1
3Ø Y1=RND(192)-1
5Ø X2=RND(256)-1
6Ø Y2=RND(192)-1
7Ø C=RND(8)
8Ø GOSUB 2ØØ
9Ø GOTO 9Ø
2ØØ PMODE 3,1
21Ø SCREEN 3,1
22Ø PCLS
23Ø LINE(X1,Y1)-(X2,Y2),PSET
24Ø RETURN
```



DECISIONS, DECISIONS

We hope you remember all your IF/THEN decisions information, because we're going to move on from there now.

COMPUTED GOTO

Sometimes several IF/THEN statements can be replaced with what is known as a "COMPUTED GOTO". A computed GOTO sends the computer to a new line, providing a certain condition is met. The BASIC word "ON" is used with GOTO to form a computed GOTO.

Remember the math program on page 22? It can be made shorter with a computed GOTO. Here is the same program using our new trick:

```
1Ø INPUT "X ";X
2Ø INPUT "Y ";Y
3Ø PRINT "CHOOSE ONE:"
4Ø PRINT "1-ADDITION"
5Ø PRINT "2-SUBTRACTION"
6Ø INPUT "1 OR 2 ";A
7Ø ON A GOTO 9Ø,11Ø
8Ø END
9Ø PRINT X + Y
1ØØ STOP
11Ø PRINT X - Y
13Ø STOP
```

The BASIC word "STOP" does what it says; it STOPs the program just as END does.

COMPUTED GOSUB

A GOSUB can also be computed. The word ON is used again. The only difference between a computed GOTO and a computed GOSUB is that the computer RETURNs to the line after the computed GOSUB when the subroutine is finished. The math problem can also be written to use a computed GOSUB.

```
1Ø INPUT "X ";X
2Ø INPUT "Y ";Y
3Ø PRINT "CHOOSE ONE:"
4Ø PRINT "1-ADDITION"
5Ø PRINT "2-SUBTRACTION"
6Ø INPUT "1 OR 2 ";A
7Ø ON A GOSUB 9Ø,11Ø
8Ø END
9Ø PRINT X + Y
1ØØ RETURN
11Ø PRINT X - Y
12Ø RETURN
```

CONDITIONAL "AND"

Sometimes a decision must be based on whether two conditions are true. The BASIC word "AND" allows us this option. Simply state your first condition, then use AND, and then state your second condition. Go back and look at the AGE FINDER program on page 12; it used an AND condition in line 320. Here is another simple example:

```
1Ø INPUT"ARE YOU HUNGRY ";Q1$
2Ø INPUT"IS IT LUNCHTIME ";Q2$
3Ø IF Q1$="YES" AND Q2$="YES"
   THEN 5Ø
4Ø END
5Ø PRINT"HURRY UP AND EAT AND
   GET BACK TO WORK!"
```

Try the program several times with different YES and NO combinations.

CONDITIONAL "OR"

In some cases a decision must be based on whether either one of two conditions is true. The BASIC word "OR" helps with the job. Just insert OR between your two conditions. Here's an OR condition program:

```
1Ø INPUT"IS IT LUNCHTIME ";
   Q1$
2Ø INPUT"IS IT DINNERTIME "
   ;Q2$
3Ø IF Q1$="YES" OR Q2$="YES"
   THEN 5Ø
4Ø END
5Ø PRINT"IT IS TIME TO EAT
   SOMETHING"
```

Try the program several times with different YES and NO combinations.

PROGRAMMING TIPS

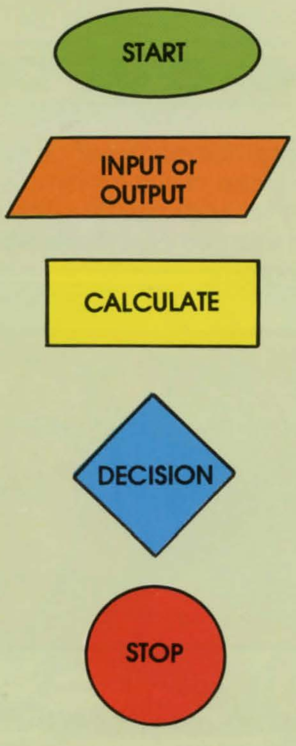
In this section we'll give you some helpful tips on programming. They aren't rules or anything, but we have found that they make life a whole lot easier.

There are two ways to write a program; you can write it directly into the computer as it comes to you or you can take pencil in hand and make a paper plan. Short programs (10 lines or less) don't usually have to be planned; but for a long program it's a good idea to know where you're going.

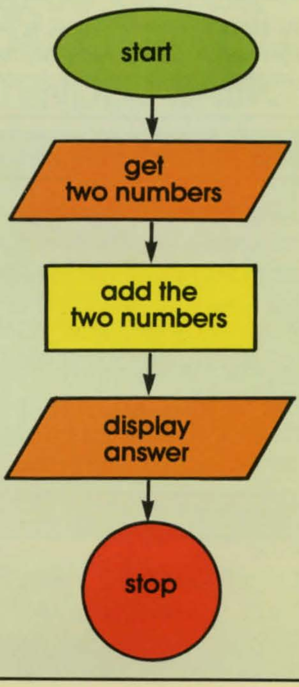
First of all, the basic ideas for your long program should be written down. Think about your objective and decide what you want the computer to do. Note the special features you want the program to have.

With your ideas organized, you can then draw up a flowchart. This is a chart of boxes, circles and diamonds that you will connect with lines and arrows. Each figure represents a part of your program ideas. The purpose of the flowchart is to help you organize your ideas into a logical pattern, and if there's one thing the computer loves, it's logic.

FLOWCHART SYMBOLS:

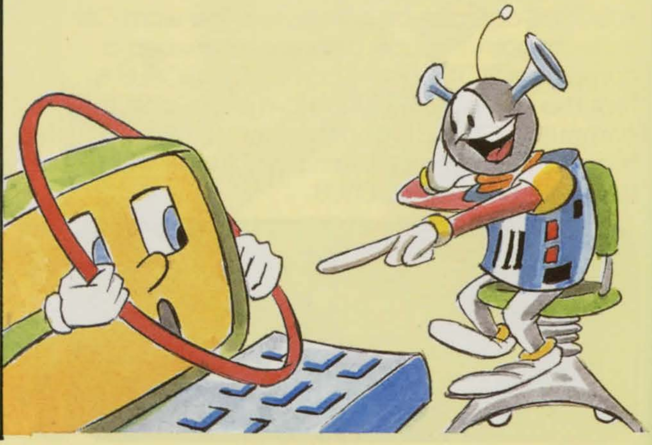


Here is a flowchart for a simple addition program:



In longer, more complicated flowcharts, subroutines can branch off your main program flowchart. Use arrows to show how they can be used again and again. (Remember GOTO, GOSUB and RETURN.)

Back to the actual program. It's best to group your subroutines together. Put them either at the beginning of your program (use GOTO to get to the main program) or at the end (use END or STOP to prevent the computer from continuing into the subroutines).



REMARKS

To help you remember what the various parts of your program do, you can include comments or short explanations. The BASIC word "REM," short for REMark, allows you to do this. When the computer sees REM, it ignores what follows and continues on with the next line. REM lines are for your use, not the computer's. REM is used like this:

```
1Ø REM PROGRAM TO ADD TWO NU
   MBERS
2Ø REM ASK FOR TWO NUMBERS
3Ø INPUT A
4Ø INPUT B
5Ø REM ADD THE TWO NUMBERS
6Ø C=A+B
7Ø REM DISPLAY ANSWER
8Ø PRINT C
```

REM makes understanding a program easier.

VARIABLES

Your program will probably use some variables. Remember to make up names that link the variable to its use. And don't forget that the computer looks at the first two characters. CARS and CABS may be different to you, but the computer sees them both as CA. — And no BASIC words as variables; that really throws the computer.

DATA STATEMENTS

When your programs need DATA statements, group them together. Put them near the beginning or near the end. When they are grouped, it's easier to find them if you need them. (The computer won't have any trouble, but you might). And be sure that the number of items in the DATA statements equals in quantity and kind the number the program will READ.

PRECAUTIONS

Typing in a program is a lot of work; protect yourself against frustration. Power failures can happen at any time, or the cat could knock out your plug. SAVE copies of your program often, at least once an hour. Then if a disaster happens, you only have to LOAD the last SAVEd copy, instead of screaming and then starting all over again.

A good program will move from your head to paper and then to the computer. Here are some good habits to get into:

1 Write down your ideas.

2 Draw up a flowchart.

3 Group subroutines at the beginning or end.

4 Use REM to make comments.

5 Assign suitable variable names.

6 Group DATA statements at the beginning or end.

7 SAVE your program frequently.

BUILDING A GAME

We have another game for you; it's called MEMORY TESTER. The program will flash a short message on the screen. Then you must quickly recall the message. It's not as easy as it sounds!

To show you how the game was made, we'll take you through the same steps we went through.

1) First, the ideas for the program were written down.

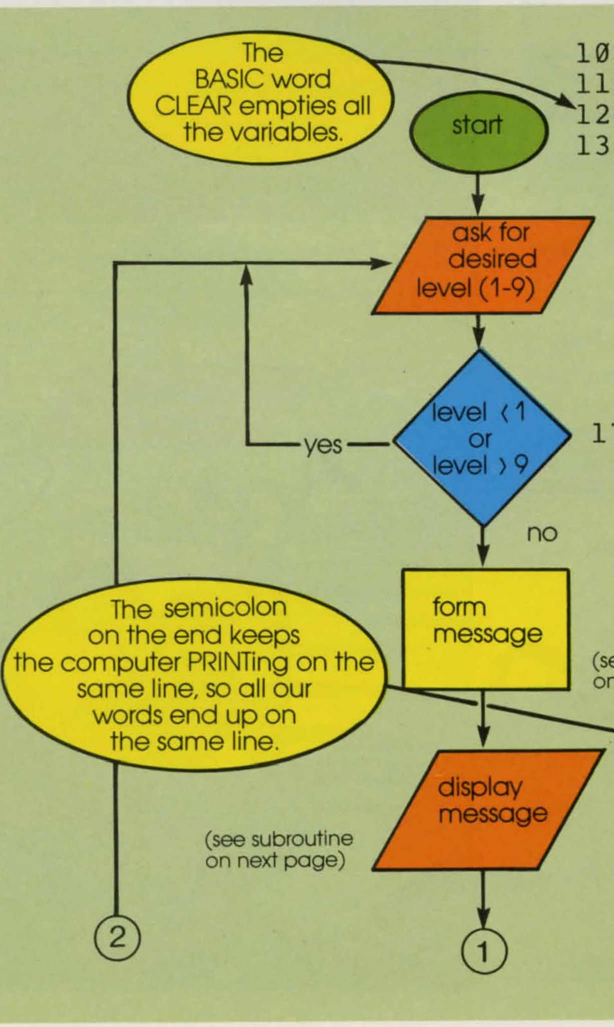
MEMORY TESTER GAME – IDEAS

- message is flashed on screen for a few seconds
- number of seconds is determined by selecting a level (1-9)
- computer then asks how many words were in message
- computer asks what the words were
- words player enters are compared to correct message
- computer evaluates a score for the player
- messages are random word combinations, 4-6 words long

2) The next step was to draw up a flowchart.

3) With the flowchart drawn up, it was time to write the program. We took each part of the flowchart and wrote the program lines. Where possible and practical we used subroutines.

MEMORY TESTER MAIN PROGRAM



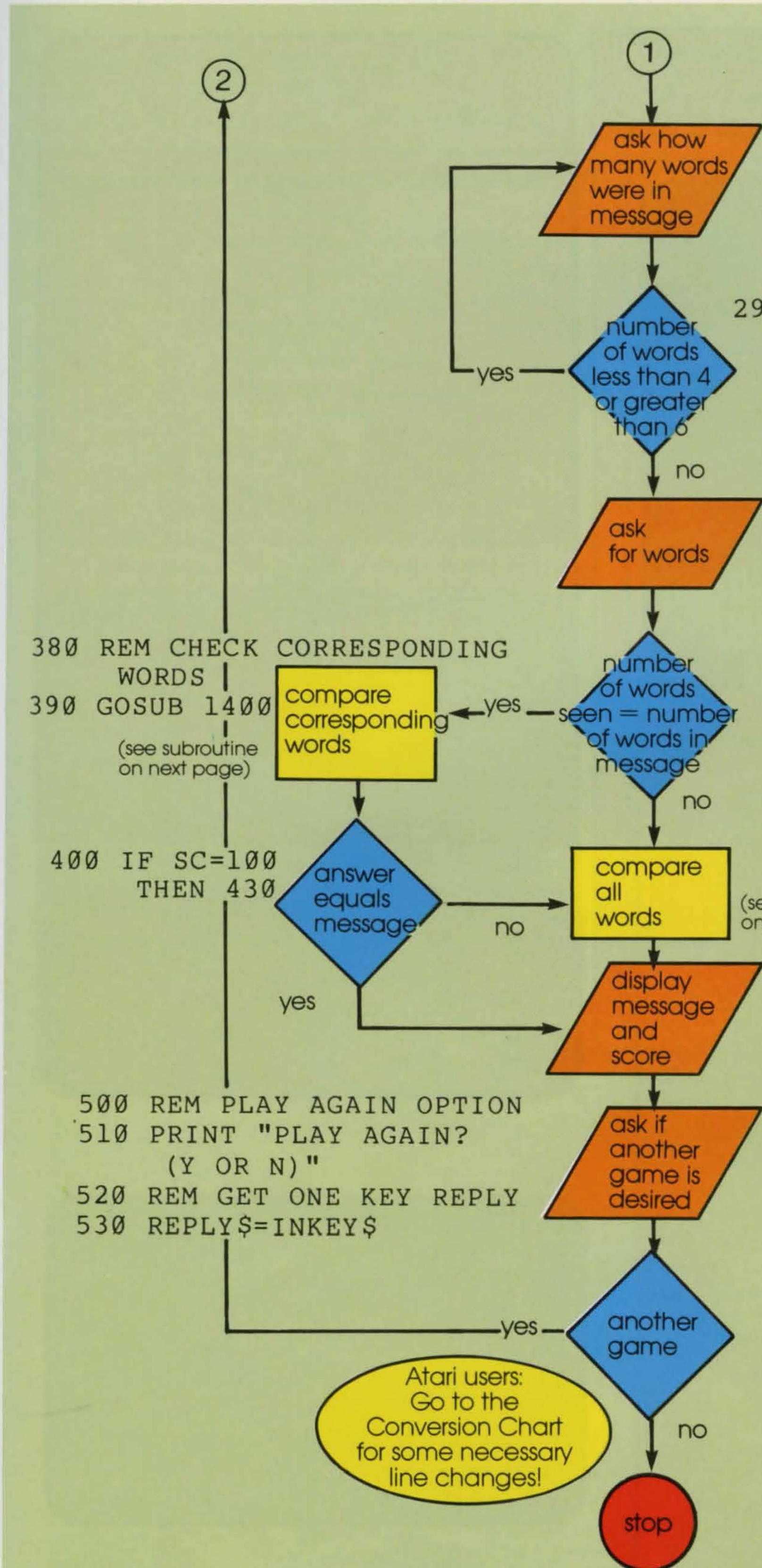
The BASIC word CLEAR empties all the variables.

The semicolon on the end keeps the computer PRINTing on the same line, so all our words end up on the same line.

(see subroutine on next page)

(see subroutine on next page)

```
1ØØ REM MEMORY TESTER
11Ø REM SET UP ARRAYS
12Ø CLEAR
13Ø DIM MESSAGE$(6),WSEEN$(6)

14Ø CLS
15Ø REM GET LEVEL (1-9)
16Ø INPUT "LEVEL (1-9) ";
    LEVEL

17Ø IF LEVEL < 1 OR LEVEL >
    9 THEN 16Ø

18Ø REM FORM MESSAGE
19Ø GOSUB 1ØØØ

2ØØ REM FLASH MESSAGE ON
    SCREEN
21Ø CLS
22Ø FOR I=1TO AW
23Ø PRINT MESSAGE$(I);" ";
24Ø NEXT I
25Ø REM DELAY TIME=LEVEL*
    .5 SECONDS
26Ø GOSUB 12ØØ
27Ø CLS
```


1
2
ask how many words were in message
number of words less than 4 or greater than 6
yes
no
ask for words
number of words seen = number of words in message
yes
no
compare corresponding words
answer equals message
no
yes
compare all words
display message and score
ask if another game is desired
another game
yes
no
stop

```
280 REM GET PLAYER'S
     ANSWER
290 INPUT "NUMBER OF
     WORDS IN MESSAGE
     ";NW
295 IF NW<4OR NW>6THEN290
300 REM INPUT WORDS
310 FOR I=1TO NW
320 PRINT "WORD #";I
330 INPUT WSEEN$(I)
340 NEXT I
350 REM CHECK WHETHER
     NUMBER OF WORDS
360 REM SEEN EQUALS
     NUMBER IN MESSAGE
370 IF NW<>AW THEN 410
380 REM CHECK CORRESPONDING
     WORDS
390 GOSUB 1400
```

(see subroutine on next page)

```
400 IF SC=100
     THEN 430
410 REM CHECK EACH WORD
420 SC=0:GOSUB 1600
```

(see subroutine on next page)

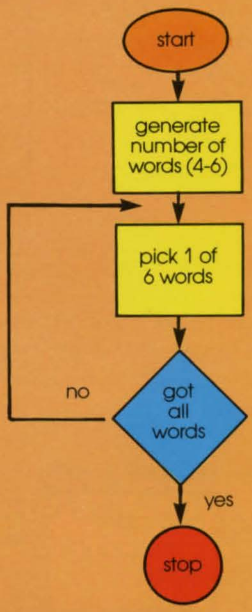
```
430 REM DISPLAY
     MESSAGE AND SCORE
440 PRINT "THE MESSAGE
     WAS..."
450 FOR I=1TO AW
460 PRINT MESSAGE$(I)
     ;" ";
470 NEXT I
480 PRINT
490 PRINT "YOUR SCORE
     IS ";SC;"%"
500 REM PLAY AGAIN OPTION
510 PRINT "PLAY AGAIN?
     (Y OR N)"
520 REM GET ONE KEY REPLY
530 REPLY$=INKEY$
540 IF REPLY$="Y"THEN
     140
550 IF REPLY$<>"N"THEN
     520
560 REM END OF GAME
570 END
```

Atari users: Go to the Conversion Chart for some necessary line changes!

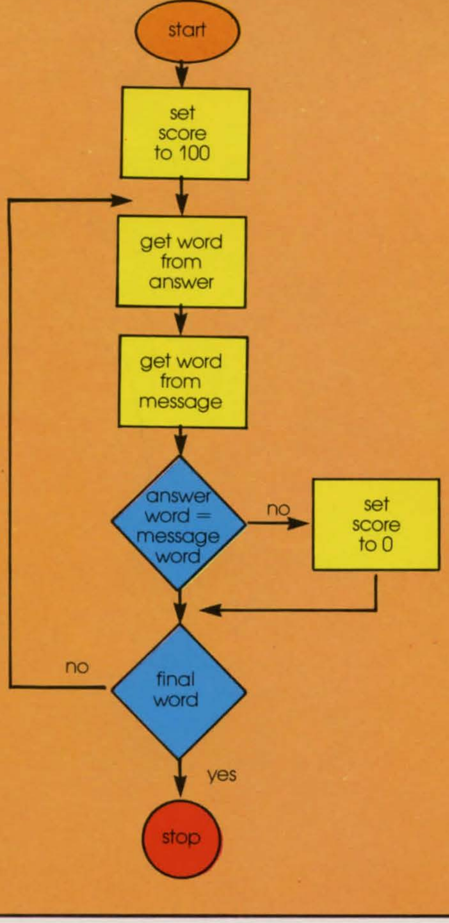
MEMORY TESTER SUBROUTINES

```
1ØØØ REM SUBROUTINE TO
      FORM MESSAGE
1Ø1Ø RESTORE
1Ø2Ø AW=3+RND(3)
1Ø3Ø FOR I=1TO AW
1Ø4Ø W=RND(6)
1Ø5Ø FOR J=1TO W
1Ø6Ø READ MESSAGE$(I)
1Ø7Ø NEXT J
1Ø8Ø IF W=6 THEN 112Ø
1Ø9Ø FOR K=1TO6-W
11ØØ READ DU$
111Ø NEXT K
112Ø NEXT I
113Ø RETURN
```

Flowchart for subroutine to form message



Flowchart for subroutine to check corresponding words



```
12ØØ REM SUBROUTINE TO DELAY
121Ø REM DELAY TIME=LEVEL*
      .5 SECONDS
122Ø FOR I=1TO LEVEL*55Ø
123Ø NEXT I
124Ø RETURN
```

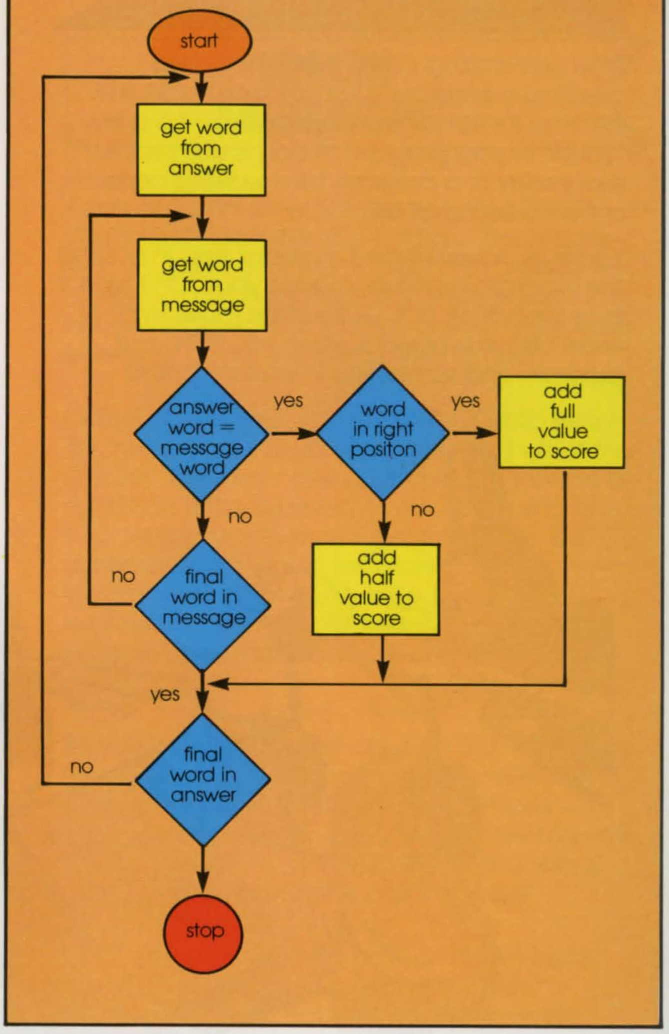
```
14ØØ REM SUBROUTINE TO CHECK
141Ø REM CORRESPONDING WORDS
142Ø REM SET SCORE TO Ø
      IF NOT
143Ø SC=1ØØ
144Ø FOR I=1TO AW
145Ø IF WSEEN$(I)<>MESSAGE$
     (I)THEN SC=Ø
146Ø NEXT I
147Ø RETURN
```

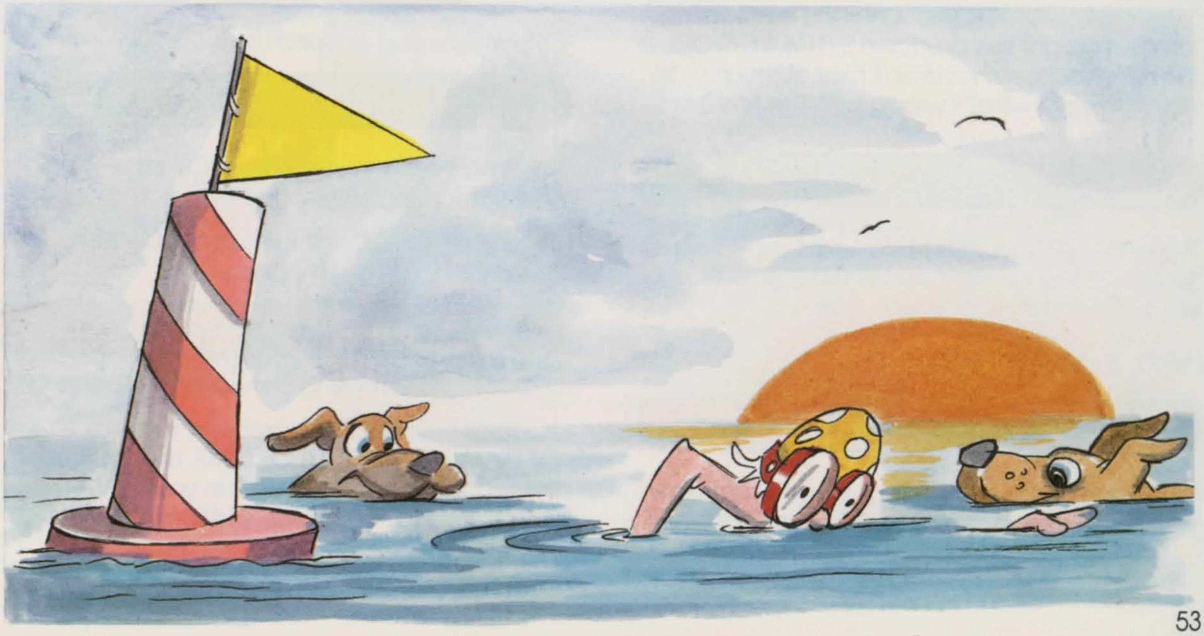
```
1600 REM SUBROUTINE TO
      CHECK WORDS
1610 REM IN PLAYER'S ANSWER
      TO MESSAGE
1620 FOR I=1TO AW
1630 FOR J=1TO NW
1640 IF MESSAGE$(I)<>WSEEN$
     (J)THEN 1700
1650 REM ADD TO SCORE
1660 S=(100/AW)/2
1670 IF I=J THEN S=S*2
1680 SC=SC+S
1690 J=NW
1700 NEXT J
1710 NEXT I
1720 RETURN
```

Data for subroutine to form message

```
2000 REM DATA TO FORM
      MESSAGES
2010 DATA JACK,SUSAN,JEFF,
     SANDY,DAVE,DOUG
2020 DATA QUICKLY,SLOWLY,
     SWIFTLY,BRISKLY,RAPID
     LY,SPEEDILY
2030 DATA RUNS,JUMPS,WALKS,
     SWIMS,GOES,FOLLOWS
2040 DATA SOUTH,NORTH,EAST,
     WEST,UP,DOWN
2050 DATA OVER,UNDER,ACROSS
     ,THROUGH,BETWEEN,
     BENEATH
2060 DATA CARS,CATS,DOGS,
     DARTS,CABS,DOORS
```

Flowchart for subroutine to check words



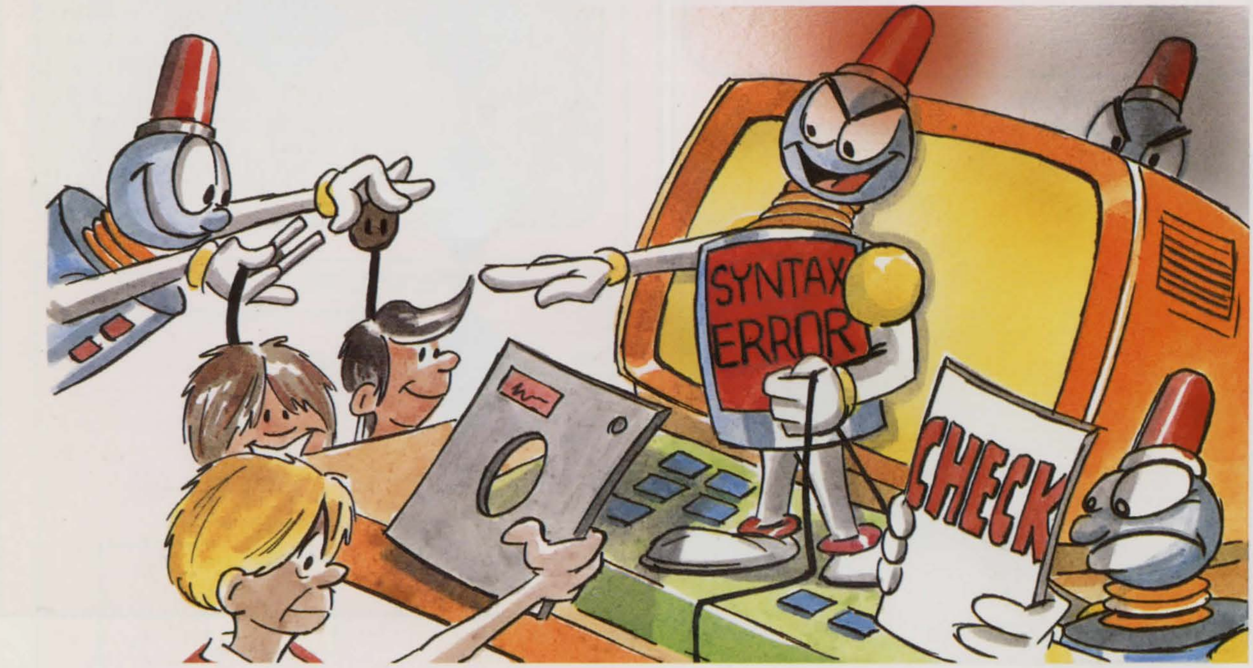


DEBUGGING

After painstaking care in designing and creating a program, it usually won't work the first time it's RUN. There are bound to be a few "BUGS" in your program. A bug is any problem that prevents a program from working correctly, or from working at all.

You may have skipped a crucial step in putting the program together; maybe you didn't type it in or even think of it. Quite often those kinds of errors become obvious when you RUN your program and something important fails to happen. Perhaps on first RUNning the MEMORY TESTER you didn't have time to read the message. Check, maybe you missed the delay routine. Just go back and enter it.

Sometimes errors aren't that obvious. The computer may simply stop executing the program. It will give you the line number where it stopped and an error message. But the computer has a very limited vocabulary so these messages may not always pinpoint the exact problem.



Oh no, you got the dreaded SYNTAX ERROR! Even they sometimes aren't that obvious. Can you figure out what's wrong with this line?

```
21Ø FØR I=1 TO NWO
```

The word FØR should be changed to FOR. Mistaking the number zero (Ø) for the capital letter "O" is quite common.

Here's another tough one:

```
23Ø PR1NT "ANOTHER GAME?"
```

PR1NT should be PRINT. Confusing the number one (1) for the capital letter "I" is also common.

Or perhaps you entered:

```
PRINT ANOTHER GAME"
```

This time the beginning quotes on the character string were missing.

There could be a problem with your DATA statements. Look at this line:

```
2ØØØ DATA FROG,DOG CAT
```

The comma is missing between DOG and CAT.

Another possible DATA problem is the old "OUT OF DATA ERROR." OUT OF DATA means that there are not enough items in your DATA statements to match the READ statements.

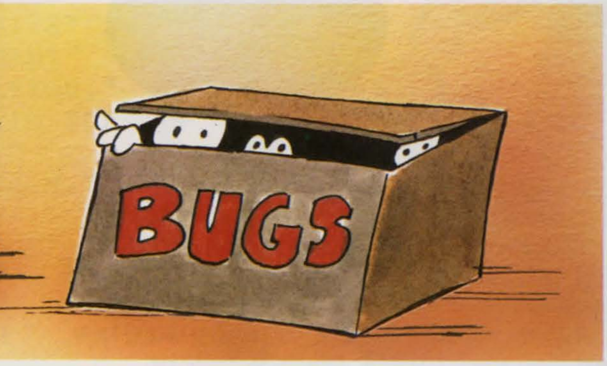
ERROR MESSAGES

Another common error message is "UNDEF'D STATEMENT" OR "LINE NOT FOUND." This means that you used a GOTO or GOSUB to branch to a line that did not in fact exist. Perhaps you haven't entered that line yet, or you might have entered it under the wrong line number.

Did you get a "RETURN WITHOUT GOSUB" error? This means a subroutine was reached without a GOSUB. You may have used a GOTO instead of a GOSUB. If the subroutines are at the end of your program, you may have left out an END statement that would separate the main program from the subroutines. If your subroutines are at the beginning of the program, you may need a GOTO to get to the main program.

Should you get a "NEXT WITHOUT FOR ERROR ," the computer has found one more NEXT than there are FOR statements. You may have forgotten one of your FOR lines, or there may be an extra NEXT line, or the variable used with a FOR does not match the one with NEXT.

Some bugs can be very difficult to find, but if there is a bug, your program won't RUN properly – it might not work at all! When you can't find a bug, try retyping suspected or complex lines. You may correct the problem without even noticing the error.



BUBBLE SORT

Whenever someone hands you a long list of names, he usually wants it put into alphabetical order. Sorting a list like that is not only dull, it's frustrating – you make one mistake and it's all wrong! Well, now you can pass the buck to your computer! With a sorting program it will happily sort your list into perfect order. We're going to be very generous and give you a sorting program. It's called "BUBBLE SORT."

```
5 CLS
1Ø INPUT "NUMBER OF WORDS
    TO SORT";N
2Ø DIM W$(N)
25 PRINT "ENTER WORDS:"
3Ø FOR I=1TO N
35 PRINT "WORD #";I
4Ø INPUT W$(I)
5Ø NEXT I
6Ø PRINT "NOW SORTING...
   PLEASE WAIT"
7Ø PRINT
8Ø FOR I=1TON-1
9Ø FOR J=I+1TON
1ØØ IF W$(I)<W$(J)THEN15Ø
11Ø DU$=W$(J)
12Ø W$(J)=W$(I)
13Ø W$(I)=DU$
15Ø NEXT J
16Ø NEXT I
17Ø FOR I=1TON
18Ø PRINT W$(I)
19Ø NEXT I
```

Atari users: Go to the Conversion Chart for some necessary line changes.

Can you figure out how the bubble sort program works? After all the words have been entered, the computer looks at each word. It compares each word to every word that comes after it in the list. If two words are in the wrong order, they are swapped. When every word has been checked this way, the list is in order.

This method of sorting is the slowest there is, but it's easy to understand. Other sorting methods, such as the "QUICK SORT,"are much faster, but the programs are longer and more complex than the bubble sort. You can find other sort programs in computer books.

FUN TIME

Well, you've done it! You are now a computer person. BASIC is your second language, your first computer language. We haven't taught you everything there is to know about BASIC, but you've learned enough to call yourself a BASIC buff.

All that work deserves some kind of reward, so we've got another game for you now. This game is called CODE CRACKER. The computer creates a three-digit code (actually a number between 100 and 999). You have 15 tries to crack the code and guess the number. As you go along the computer will tell you how many digits you have right; it will also state how many are in the right position. Use this information logically to determine the code. Have fun!

Special Note: The program will not allow double digits, that's too confusing; so numbers like 988 are out. And here's a brief explanation of the hints the computer will give:

DIGIT 0 POSITION 0

– no numbers right

DIGIT 0 POSITION 1

– 1 digit in the right position

DIGIT 1 POSITION 0

– 1 digit right, but in wrong position

DIGIT 1 POSITION 1

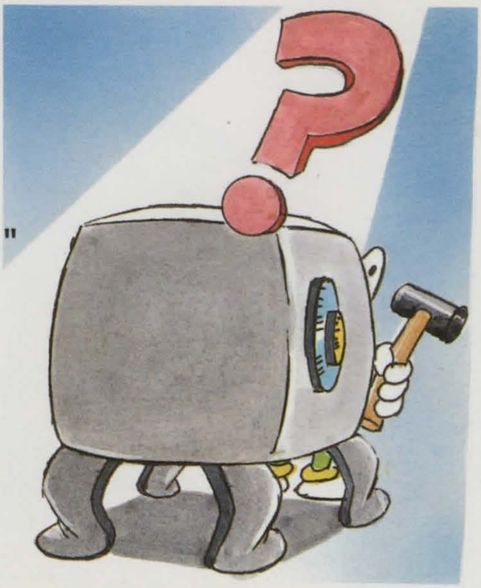
– 1 digit in wrong position and 1 in the right position

GO FOR IT!

CODE CRACKER

Atari users: Go to the Conversion Chart for some necessary line changes.

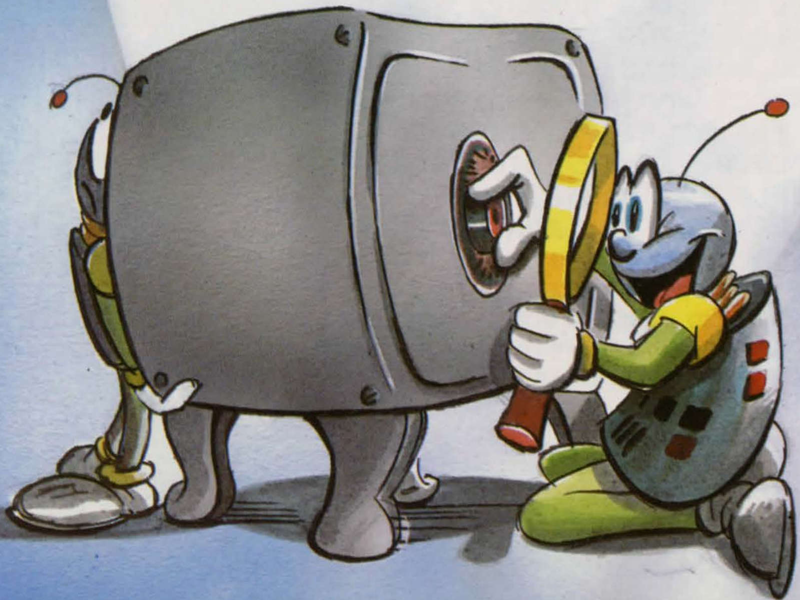
```
100 REM CODE CRACKER GAME
110 REM SET UP ARRAYS
120 DIMG$(15),RP(15),RD(15)
130 REM GIVE INSTRUCTIONS
140 CLS
150 PRINT"IN THIS GAME I MAKE UP A"
160 PRINT"THREE DIGIT CODE.  THE CODE"
170 PRINT"WILL BE BETWEEN 100 AND 999."
180 PRINT"YOU HAVE FIFTEEN GUESSES."
190 PRINT"AFTER EACH GUESS I WILL"
200 PRINT"TELL YOU HOW MANY DIGITS"
210 PRINT"ARE CORRECT AND HOW MANY"
220 PRINT"ARE IN THE RIGHT POSITION."
230 PRINT"IF YOU ENTER AN INVALID GUESS"
240 PRINT"OR A DUPLICATE GUESS,"
250 PRINT"I WILL LET YOU TRY AGAIN."
260 PRINT
270 PRINT"PRESS A KEY TO CONTINUE"
280 REM WAIT FOR KEY PRESS
290 GOSUB1000
300 REM MAKE CODE
310 CLS
320 CODE=RND(999)
330 IFCODE<100THEN320
340 CODE$=RIGHTS(STR$(CODE),3)
```



CODE CRACKER

```
350 REM CHECK FOR DUPLICATE DIGITS
360 D$=CODE$
370 FLAG=Ø
380 GOSUB15ØØ
390 IFFLAG=1THEN32Ø
4ØØ REM GET GUESSES
41Ø FORI=1TO15
42Ø CLS
43Ø REM DISPLAY PREVIOUS RESULTS
44Ø IFI=1THEN5ØØ
45Ø FORJ=1TOI-1
46Ø PRINT"GUESS #";J;"; ";G$(J);
47Ø PRINT" DIGITS: ";RD(J);
48Ø PRINT" POSITIONS: ";RP(J)
49Ø NEXT J
5ØØ REM ENTER NEXT GUESS
51Ø PRINT
52Ø PRINT"ENTER GUESS #";I
53Ø INPUTG
54Ø IFG<1ØØORG>999THEN52Ø
55Ø G$(I)=STR$(G)
56Ø IFLEN(G$(I))>4THEN52Ø
57Ø G$(I)=RIGHT$(G$(I),3)
58Ø REM CHECK FOR DUPLICATE DIGITS
59Ø D$=G$(I)
6ØØ FLAG=Ø
61Ø GOSUB15ØØ
62Ø IFFLAG=1THEN52Ø
63Ø FLAG=Ø
64Ø REM CHECK FOR DUPLICATE ANSWER
65Ø GOSUB14ØØ
66Ø IFFLAG=1THEN52Ø
67Ø REM COMPARE TO GUESS
68Ø GOSUB11ØØ
69Ø REM CHECK IF ANSWER IS RIGHT
```

"FLAG" is a variable that will be either one or zero. It will be 1 for one condition and Ø for another.



```
700 IFRP(I)<>3THEN730
710 PRINT"VERY GOOD!  YOU GOT IT IN ";I;" GUESSES."
720 END
730 NEXTI
740 REM DIDN'T GET THE CODE
750 CLS
760 PRINT"THAT WAS ALL YOUR GUESSES."
770 PRINT"THE CODE WAS ";CODE$
780 END
1000 REM WAIT FOR KEY PRESS
1010 R$=INKEY$
1020 RETURN
1100 REM COMPARE GUESS TO CODE
1110 FORK=1TO3
1120 FORL=1TO3
1130 REM SEPARATE DIGITS IN GUESS
1140 REM AND ANSWER
1150 G$=MID$(G$(I),K,1)
1160 C$=MID$(CODE$,L,1)
1170 REM COMPARE DIGITS
1180 IFG$<>C$THEN1220
1190 REM COMPARE DIGIT POSITIONS
1200 IFK<>LTHENRD(I)=RD(I)+1
1210 IFK=LTHENRP(I)=RP(I)+1
1220 NEXTL
1230 NEXTK
1240 RETURN
1400 REM CHECK FOR DUPLICATE ANSWER
1410 IFI=1THENRETURN
1420 FORK=1TOI-1
1430 IFG$(I)=G$(K)THENFLAG=1
1440 NEXTK
1450 RETURN
1500 REM CHECK FOR DUPLICATE DIGITS
1510 FORK=1TO3
1520 FORL=K+1TO3
1530 IFMID$(D$,K,1)=MID$(D$,L,1)THENFLAG=1
1540 NEXTL
1550 NEXTK
1560 RETURN
```

BASIC CONVERSION CHART

IN BOOK	COMMODORE	APPLE/ADAM
LIST 1Ø - 3Ø	LIST 1Ø - 3Ø	LIST 1Ø - 3Ø
1Ø INPUT"STRING ";A	1Ø INPUT"STRING";A	1Ø INPUT"STRING ";A
CLS	PRINT CHR$(147)	HOME
X$=INKEY$	1Ø GET X$ 2Ø IF X$=""THEN 1Ø	GET X$
D=RND(6)	D=INT(RND(1)*6+1)	D=INT(RND(1)*6+1)
CLEAR	CLR	CLEAR

ATARI users, here are the line changes and the additional lines you require.

AGE FINDER on page 12, make these line changes:

```
120 READ T
125 N(I)=T
```

THE FIRST NESTED LOOP PROGRAM on page 41, make these additions and changes:

```
35 FORL=1TOJ
36 PRINT " ";
37 NEXT L
4Ø PRINT"BASIC IS FUN"
```

THE FIRST ARRAY PROGRAM on page 43, make these additions and changes:

```
2Ø PRINT"BIRTH YEAR NUMBER 1 ";:INP
   UT BY:BY(1)=BY
3Ø PRINT"BIRTH YEAR NUMBER 2 ";:INP
   UT BY:BY(2)=BY
4Ø PRINT"BIRTH YEAR NUMBER 3 ";:INP
   UT BY:BY(3)=BY
```

THE SECOND ARRAY PROGRAM on page 43, make this change:

```
4Ø INPUT BY:BY(I)=BY
```

MEMORY TESTER on page 50, make these line changes:

```
12Ø CLR
13Ø DIMMESSAGE$(7*1Ø),WSEEN$(7*1Ø),
    DU$(1Ø),REPLY$(1),AL(6),ML(6)
14Ø PRINT CHR$(125)
16Ø PRINT"LEVEL (1-9)"
21Ø PRINT CHR$(125)
23Ø PRINT DU$;" ";
27Ø PRINT CHR$(125)
29Ø PRINT"NUMBER OF WORDS IN MESSAG
    E"
33Ø INPUT DU$
46Ø PRINT DU$;" ";
53Ø GET#1,REPLY
1Ø2Ø AW=3+INT(RND(1)*3+1)
1Ø4Ø W=INT(RND(1)*6+1)
1Ø5Ø FOR J=1TOW-1
1Ø6Ø NEXT J
1Ø7Ø MESSAGE$(I*1Ø)=DU$
122Ø FOR I=1TOLEVEL*1ØØ
145Ø IF WSEEN$(I*1Ø,I*1Ø+AL(I))<>ME
     SSAGE$(I*1Ø,I*1Ø+ML(I))THENSC=
     Ø
164Ø IFMESSAGE$(I*1Ø,I*1Ø+ML(I))<>W
     SEEN$(J*1Ø,J*1Ø+AL(J))THEN17ØØ
```

And add these additional lines:

```
1Ø OPEN#1,4,Ø,"K:"
165 INPUT LEVEL
225 DU$=MESSAGE$(I*1Ø,I*1Ø+ML(I))
295 INPUT NW
333 AL(I)=LEN(DU$)-1
335 WSEEN$(I*1Ø)=DU$
455 DU$=MESSAGE$(I*1Ø,I*1Ø+ML(I))
535 REPLY$=CHR$(REPLY)
1Ø45 IF W=1 THEN 1Ø62
1Ø55 READ DU$
1Ø62 READ DU$
1Ø63 ML(I)=LEN(DU$)-1
```

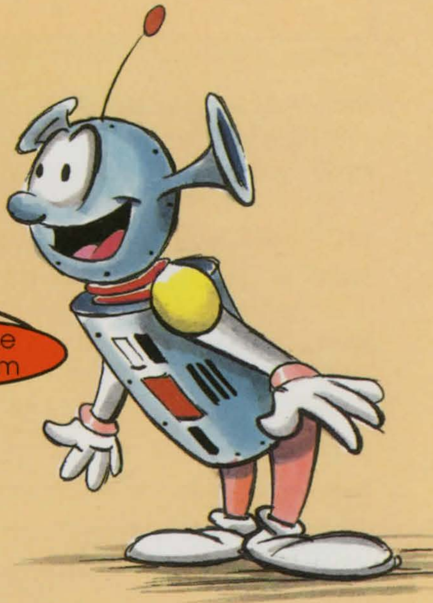
TRS-80	ATARI
`LIST 1Ø - 3Ø`	`LIST 1Ø,3Ø`

`1Ø INPUT"STRING";A`	`1Ø PRINT"STRING";`
	`11 INPUT A`

`CLS`	`PRINT CHR$(125)`

`1Ø X$=INKEY$`
`2Ø IF X$="" THEN 1Ø`
	`1 OPEN#1,4,Ø,"K:"`
	`2 DIMX$(1)`
	`1Ø GET#1,X`
	`2Ø X$=CHR$(X)`
	only do once in a program

`D=RND(6)`	`D=INT(RND(1)*6+1)`
`CLEAR`	`CLR`



CODE CRACKER on page 57, make these line changes:

```
120 DIMG$(16*3),DU$(4),CODE$(3),D$(
    3),C$(3),RP(15),RD(15)
140 PRINT CHR$(125)
310 PRINT CHR$(125)
320 CODE=INT(RND(1)*999+1)
340 CODE$=STR$(CODE)
420 PRINT CHR$(125)
460 PRINT"GUESS #";J;" ";G$(J*3,J*3
    +2)
550 G$(I*3,I*3+2)=STR$(G)
590 D$=G$(I*3,I*3+2)
750 PRINT CHR$(125)
1010 GET#1,A
1150 D$=G$(I*3+K-1,I*3+K-1)
1160 C$=CODE$(L,L)
1180 IFD$<>C$THEN1220
1430 IF G$(I*3,I*3+2)=G$(K*3,K*3+2)
     THEN FLAG=1
1530 IF D$(K,K)=D$(L,L) THEN FLAG=1
```

Delete lines 560 and 570. And add these additional lines:

```
90 OPEN#1,4,0,"K:"
122 FOR I=1 TO 15
124 RP(I)=0
125 RD(I)=0
128 NEXT I
1525 IF K>3 OR L>3 THEN 1540
```

BUBBLE SORT on page 56, make these line changes:

```
5 PRINT CHR$(125)
10 PRINT "NUMBER OF WORDS TO SORT"
20 DIM W$(N*20+20),DU$(20),L(N)
40 INPUT DU$
100 IF W$(I*20,I*20+L(I))<W$(J*20,J
    *20+L(J))THEN150
110 DU$=W$(J*20,J*20+L(J))
120 W$(J*20,J*20+L(I))=W$(I*20,I*20
    +L(I))
130 W$(I*20,I*20+L)=DU$
180 PRINT W$(I*20,I*20+L(I))
```

And add these new lines:

```
15 INPUT N
45 L(I)=LEN(DU$)-1
48 W$(I*20)=DU$
115 L=L(J)
125 L(J)=L(I)
135 L(I)=L
```

GLOSSARY OF BASIC COMMANDS AND COMPUTER TERMS

AND – A BASIC language word, it is a conditional operator; it ensures that two or more conditions are true before any action is taken.

array – A math term also used by computer people, it refers to a collection of related information stored under one variable name.

ASC – A BASIC command word used to convert a character to its ASCII code number.

ASCII – A contraction of "American Standard Code for Information Interchange," pronounced ASK-KEY or ASH-KEY.

BASIC – A contraction of "Beginner's All-purpose Symbolic Instruction Code"; it is a computer language, a variation of which is common to most home computers.

branch – A computer term for the switching from one part of a program to another.

bubble sort – A type of program that organizes a list into alphabetical or numerical order.

bug – A computer term for an error in a program that stops it from running properly or from running at all.

character – A computer term referring to a number, letter, symbol or space.

character string – A computer term referring to a series of characters inside quotation marks.

CHR$ – A BASIC command word used to convert a number into its ASCII code number.

CLEAR – A BASIC command word used to empty variables.

CLS – A BASIC command word used to clear the screen.

computed GOSUB – A computer term that refers to a branch to a subroutine depending on the value of a variable.

computed GOTO – A computer term that refers to a branch to a line number depending on the value of a variable.

co-ordinates – A computer term referring to a set of numbers that specify a row and column on the computer screen.

DATA – A BASIC command word used to store a list of information, to be used by a READ statement.

debug – A computer term that means "to correct errors."

delay – Used in computer terminology to refer to a pause while the computer does nothing for a specified period of time.

delete – Used in computer terminology to mean take something out; take out a character, number, space, a whole line or a program.

DIM – A BASIC command word short for DIMension; it tells the computer to expect an array of a given size.

disk, diskette, floppy diskette – A round flat piece of plastic with a specially magnetized surface on which information is stored.

disk drive – A device that rotates (spins) a disk so that information can be read or written.

element – Used in computer terminology to refer to a single piece of information in an array.

END – A BASIC command word used to stop a program.

flowchart – A computer term that refers to a series of boxes, diamonds and circles which graphically outlines the logical sequence of a program.

FOR/TO/NEXT – BASIC command words for a type of loop that specifies the number of repetitions.

GOSUB – A BASIC command word that sends the computer to a subroutine.

GOTO – A BASIC command word that sends the computer to a line number.

graphics – In computer terminology refers to pictures or graphs on a computer screen.

high resolution – In computer terminology refers to very detailed graphics pictures, usually drawn with individual pixels.

hi-res – A short form for high resolution.

IF/THEN – BASIC command words that tell the computer to make a branch depending on the information it receives.

index – In computer terminology it is used with arrays, the number indicating a particular piece of information, the information is numbered sequentially in the array.

INKEY$ – A BASIC command word that takes a single character keyboard response and assigns it to a variable.

input – In computer terminology, any information that the computer is given to work with.

INPUT – A BASIC command word which tells the computer to expect some in-coming information.

LEFT$ – A BASIC command word that tells the computer to use part of a string, starting from the left side; reference numbers given tell it how many characters to use.

LEN – A BASIC command word short for LENgth; this word tells the computer to count the number of characters in a string.

LINE NOT FOUND – An error message; a line number was referenced that did not exist.

line number – A number given to a line of programming instructions.

LIST – A BASIC command word to have the computer display a line or program.

LOAD – A BASIC command word that tells the computer to put a program into its memory.

loop – A computer term that refers to having the computer do something again and again.

memory – The space where the computer stores information and/or instructions.

MID$ – A BASIC command word that tells the computer to use part of a string, reference numbers given tell it how many characters to use, starting at a specified middle point.

nested loop – A loop within a loop; the inner loop is performed as many times as the outside loop tells it to.

NEW – A BASIC command word that tells the computer to clear its memory.

NEXT – A BASIC command word that tells the computer to keep doing something until it has reached the specified number of times it is supposed to do it; used with FOR, TO and STEP.

NEXT WITHOUT FOR – An error message; the computer found an extra NEXT command after it has finished a loop.

ON – A BASIC command word used with GOTO or GOSUB to perform a computed GOTO or GOSUB.

OR – A BASIC command word that allows one of two or more given conditions to be true before any action is taken.

OUT OF DATA – An error message; the number of variables in a READ statement outnumber the pieces of information to be read from the available DATA statements.

output – A computer term that refers to the result or the answer the computer has arrived at.

PEEK – A BASIC command word used to examine a screen or memory location.

pixel – Short for picture element, it is a tiny dot on the computer screen; groups of these lighted at any given time form letters, numbers and graphics.

POKE – A BASIC command word used to put something into a screen or memory location.

PRINT – A BASIC command word that tells the computer to display something on the screen.

program – A series of instructions for the computer to perform.

quick sort – A type of program that organizes a list into alphabetical or numerical order.

random number – A number that is selected for no apparent reason.

READ – A BASIC command word that takes information from DATA statements and assigns it to variables.

relational operator – Used both in computer and math terminologies, something that compares two or more things as to size, eg. greater than, less than, etc.

REM – A BASIC word short for REMark, which allows the programmer to make notes inside a program; these are not read by the computer.

RESTORE – A BASIC command word that tells the computer to go back to the first DATA statement and start using the information again.

RETURN – When a computer finishes a subroutine, this BASIC word sends it back to the main program.

RETURN WITHOUT GOSUB – An error message; the computer got to a RETURN statement without having gone through a GOSUB statement.

RIGHT$ – A BASIC command word that tells the computer to use a part of a string, starting with the right-most character; a number is then given to tell the computer how many characters to take.

RND – A BASIC command word, a contraction of RaNDom, that is used to create random numbers.

RUN – A BASIC command word that tells the computer to execute the program it has in its memory.

SAVE – A BASIC command word that tells the computer to take the information it has in its memory and store it on either tape or disk.

sort – A computer program to organize a list into alphabetical or numerical order; there are many types.

STEP – A BASIC command word used with loops to specify how to go through the loop; used with FOR, TO and NEXT.

STOP – A BASIC command word used to end a program.

STR$ – A BASIC command word that changes a number variable into a string variable.

string – In computer terminology, a series of characters inside quotation marks.

subroutine – A computer term that refers to a small section of a program, a section that does one specific thing.

subscript – A number which designates a certain piece of information in an array.

SYNTAX ERROR – An error message; the computer has not understood a command due to misspelling or some other mistake.

tape – Even in computer terminology, it refers to ordinary audio cassette tape.

tape recorder – It usually refers to an ordinary cassette player.

text – Written words and numbers, as opposed to graphics.

TO – A BASIC command word used in FOR/TO/NEXT loops.

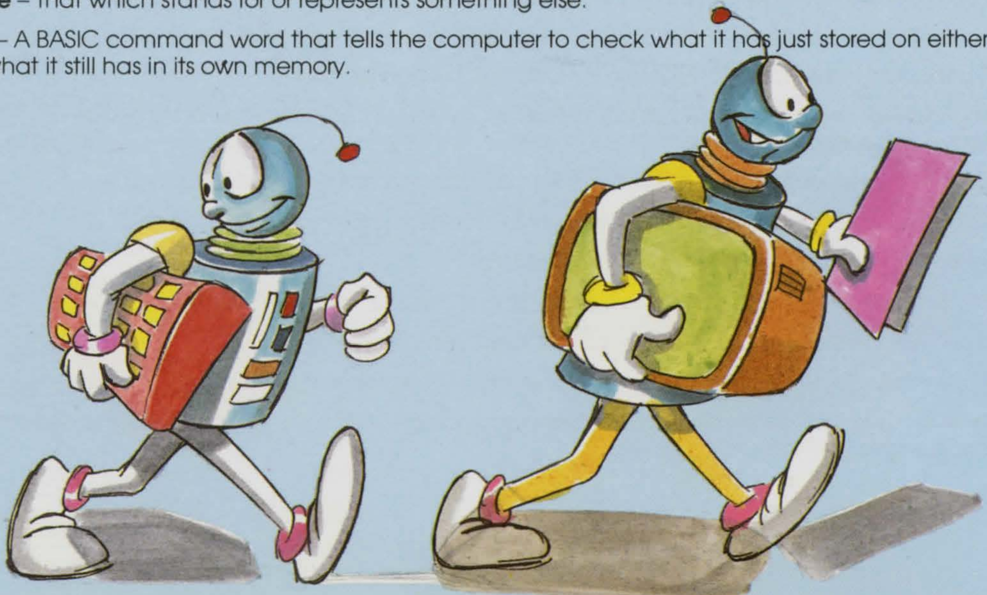
THEN – A BASIC command word that tells the computer to follow the instructions that come after it, if one or more conditions is true.

UNDEF'D STATEMENT – An error message; a line number was referenced that does not exist.

VAL – A BASIC command word short for VALue that converts a number string into a number value.

variable – That which stands for or represents something else.

VERIFY – A BASIC command word that tells the computer to check what it has just stored on either tape or disk with what it still has in its own memory.



Other computer titles from Hayes

MICRO WORLD

An introduction to home computers, this book will help you select the one that's right for you.

WHAT ELSE YOU CAN DO WITH YOUR MICROCOMPUTER

Computers do more than play games. Here are lots of practical programs to be used by every member of the family.

FANTASTIC GAMES for the VIC 20

FANTASTIC GAMES for the Commodore 64

FANTASTIC GAMES for the TRS-80 Color Computer

Each has listings for six arcade style games as well as suggestions for writing your own games.

EXPLORING THE VIC 20

EXPLORING THE COMMODORE 64

A wealth of information and activities for the most popular home computers.

APPLE GRAPHICS

Shows by example how to write arcade style games using Applesoft BASIC.

SPACE ADVENTURES for the VIC 20

SPACE ADVENTURES for the Commodore 64

Listings for six terrific space adventure games.

ISBN 0-88625-075-7

HAYES PUBLISHING LTD.

3312 MAINWAY, BURLINGTON, ONTARIO, CANADA L7M 1A7