ATARI®

Let's Learn BASIC

a kids' introduction to BASIC programming on ATARI® Home Computers

BEN SHNEIDERMAN

Let's Learn BASIC

A Kids' Introduction to BASIC Programming on ATARI® Home Computers

The Little, Brown Microcomputer Bookshelf

BANSE, TIMOTHY
Home Applications and Games for the VIC–20

BANSE, TIMOTHY
Home Applications and Games for the Apple® II, Apple® II Plus, and Apple® IIe Computers

BANSE, TIMOTHY
Home Applications and Games for the ATARI® 400™/800™, 600XL™, 800XL™, 1200XL™, 1400XL™, and 1450XLD™ Home Computers

BARNETT, MICHAEL P. AND GRAHAM K. BARNETT
Personal Graphics for Profit and Pleasure on the APPLE® II Plus Computer

BARNETT, MICHAEL P. AND GRAHAM K. BARNETT
Personal Graphics for Profit and Pleasure on the IBM® Personal Computers

HODGES, WILLIAM S. AND NEAL A. NOVAK
Personal Finance Programs for Home Computers

MORRILL, HARRIET
BASIC for IBM® Personal Computers

MORRILL, HARRIET
Mini and Micro BASIC: Introducing Applesoft®, Microsoft®, and BASIC PLUS

NAHIGIAN, J. VICTOR AND WILLIAM S. HODGES
Computer Games for Businesses, Schools, and Homes

NAHIGIAN, J. VICTOR AND WILLIAM S. HODGES
Computer Games for Business, School, and Home for TRS-80 Level II BASIC

ORWIG, GARY W. AND WILLIAM S. HODGES
The Computer Tutor: Learning Activities for Homes and Schools (for the TRS-80®, Apple®, and PET/CBM® Home Computers)

ORWIG, GARY W. AND WILLIAM S. HODGES
The Computer Tutor: ATARI® Home Computer Edition (for the ATARI® 400/800™, 600XL™, 800XL™, 1200XL™, 1400XL™, and 1450XLD™ Home Computers)

ORWIG, GARY W. AND WILLIAM S. HODGES
The Computer Tutor: IBM® Personal Computer Edition (for the IBM® PC and PCjr)

SHNEIDERMAN, BEN
Let's Learn BASIC: A Kids' Introduction to BASIC Programming on the Commodore 64®

SHNEIDERMAN, BEN
Let's Learn BASIC: A Kids' Introduction to BASIC Programming on IBM® Personal Computers

SHNEIDERMAN, BEN
Let's Learn BASIC: A Kids' Introduction to BASIC Programming on the Apple® II Series

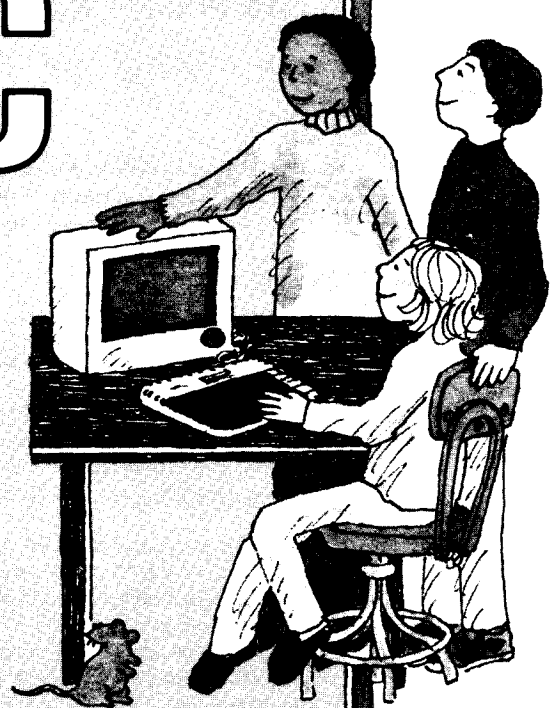
SHNEIDERMAN, BEN
Let's Learn BASIC: A Kids' Introduction to BASIC Programming on ATARI® Home Computers

WINDEKNECHT, THOMAS G.
6502 Systems Programming

Let's Learn BASIC

A Kids' Introduction to BASIC Programming on ATARI® Home Computers

Ben Shneiderman
University of Maryland



Little, Brown and Company
Boston Toronto

Library of Congress Cataloging in Publication Data

Shneiderman, Ben.
Let's learn BASIC.

(The Little, Brown microcomputer bookshelf)
Includes index.
1. Atari computer—Programming. 2. Basic (Computer program language) I. Title. II. Series.
QA76.8.A82S53 1984 001.64'2 84-12540
ISBN 0-316-78722-1

Library of Congress Catalog Card No. 84-12540

ISBN 0-316-78722-1

9 8 7 6 5 4 3 2 1

HAL

Published simultaneously in Canada
by Little, Brown & Company (Canada) Limited
Printed in the United States of America

Illustrations by True Kelley

Preface

Who is this book for? This book was written for eight year olds (third graders) up through fourteen year olds (ninth graders). I believe that children want a challenge in their education and that they are eager to learn if offered stimulating educational materials. This book requires no knowledge of programming or mathematics beyond simple arithmetic. I hope that many adults will enjoy and benefit from this book too.

Why did I write this book? I was genuinely surprised in searching through bookstores for an introduction to BASIC for my eight-year-old daughter Sara. There were several cute books that described the wonderful world of computers but offered only a slim discussion of programming. The books that taught programming used advanced concepts such as scientific notation, exponentiation, sines and cosines, and two-dimensional array operations. These books also used sophisticated examples of bank-interest computation, numerical integration, standard deviations, or elaborate sales-commission calculations.

I decided to write a book that Sara and her friends could use to learn BASIC programming. I wanted to teach the important concepts in programming in a way that made sense to anyone who was interested. My goal is to preserve intellectual rigor while simplifying the presentation by offering a logical sequence, lucid explanations, and adequate examples. I begin with simple `PRINT` commands to teach the step-by-step process of programming. Loops for repeating `PRINT` commands convey some of the power of a computer. Next, `INPUT` commands allow students to create simple interactions. Writing programs that do arithmetic, make decisions, and create stories comes after that. I worked hard to choose interesting and enjoyable examples

that are easy to understand. The examples and exercises use stories, riddles, graphics, games, poetry, and simple computations.

My research on human factors in programming and interactive systems heightened my awareness of human learning and problem-solving abilities with respect to computers. The academic literature on novice programming provided a basis for developing a precise model of learning and a plan for teaching. Classroom and individual testing provided feedback for me to change the wording sequencing, and technical content.

Why BASIC? There is a lively debate among educators about which language to use for teaching new programmers. I agree with critics of BASIC who point to its weaknesses in control structures, data organization, and modular organization. These become important limitations in longer and more complicated programs, but for beginners BASIC has many advantages:

☐ It is very easy to get started. Learning the `PRINT` command is all you need to begin writing programs.

☐ Line numbering organizes short programs. The line numbers emphasize the sequential execution of programs, a difficult concept for new programmers. Line numbers also make adding, changing, and deleting lines easy.

☐ It uses simple control structures and data structures. Some of the very aspects of BASIC that annoy experts make BASIC very useful for beginners. The flexible `PRINT` command, the `FOR–NEXT` loop, the simple `IF–THEN` conditional, and the convenient `INPUT` command allow new programmers to make rapid progress. The confusing distinctions between real or integer variables are avoided and strings are handled in a simple form.

☐ Operations on numbers are easy to program. BASIC is well-designed to accept numbers in an `INPUT` command, store

them, compare them, print them in neat columns, and do arithmetic with familiar notation.

☐ The BASIC environment is simple and understandable. Program editing, storing, retrieval, and printing are easy to learn, remember, and use. Programs are often run rapidly, by just typing RUN, and error messages can be specific and constructive. Tracing during RUNs and fast repairs encourage exploration.

☐ BASIC is widely available. Almost every microcomputer offers BASIC. As children move among classrooms, schools, their home, and their friends' homes, they can be quite confident that the computers they use will offer BASIC in a largely standardized form.

These are important advantages of BASIC, but don't forget the limitations. Once you've mastered BASIC, keep your mind open to learning other languages, especially if you want to do advanced programming.

Some people push for LOGO as the language for new programmers. It is especially appealing because new programmers can easily draw certain kinds of figures and modular programming is more convenient. But LOGO may make it more difficult to edit programs, to input numbers and strings, to do some arithmetic, and to do some kinds of printing. I think that once you learn BASIC you could benefit from learning LOGO, and once you learn LOGO you could benefit from learning BASIC.

Objectives

This book introduces computer programming and shows the important ideas with programs written in BASIC. After reading this book carefully and doing the exercises, you should be a competent novice in BASIC programming. I used only a part of

the BASIC language to simplify learning. The parts of BASIC that I used work on most microcomputers.

This book is especially for the ATARI Home Computer series: the 400, 600XL, 800, and 800XL. (There are versions of this book for other microcomputers.) Sections called Differences Among Computers compare the ATARI Home Computers, Apple II series, Commodore 64, IBM Personal Computers, Timex/Sinclair, TRS–80 Microcomputer, and TI 99/4A.

Acknowledgments

I have to begin by thanking my daughter Sara and her teachers Karen Glantz and Sheila Ford for getting me started on writing this book. Other teachers and parents at the Murch Elementary School in Washington, D.C., helped and encouraged me in many ways. Gary Orwig, Ron Schwartz, and George Richardson offered insightful comments on the manuscript's early stages. Marilyn Barth, Skip MacArthur, Mary Mullins, and Lee Ripley gave very helpful, detailed reviews of the complete book. Valuable comments, encouragement, and classroom testing were provided by Herb Bernstein, Virginia Bradley, Mary Brown, Judy Cook, Walter Ellis III, Charles Kreitzberg, John Lovgren, Patrick Pope, Jerry Weinberg, Howard Weiss, and others. Gordon Lewis of the District of Columbia Public Schools was influential in arranging the use of this book for the 1983 Summer Computer Camps. Jenelle Leonard of the Computer Literacy Training Laboratory at the Takoma Elementary School supported this effort in many ways. This trial with thousands of students and dozens of teachers helped demonstrate and improve the effectiveness of the material.

The University of Maryland Computer Science Center provided computer resources to support the many, many changes made to

the drafts. Deb Stoffel was wonderful in preparing the typesetting. Mildred Johnson helped with many secretarial chores and worked with her husband, Joseph, to prepare one of the solution sets. Len Pedowitz and Nick Roussopoulos contributed to checking the exercises and the solutions. Donald R. Mattison provided the computer picture of the violin.

Tom Casson of Little, Brown and Company was always open to discuss troubling issues and contributed substantially to many critical decisions. Mark Walsh added his expertise during the final phases. Sally Stickney and George McLean diligently pursued the numerous details of the production process.

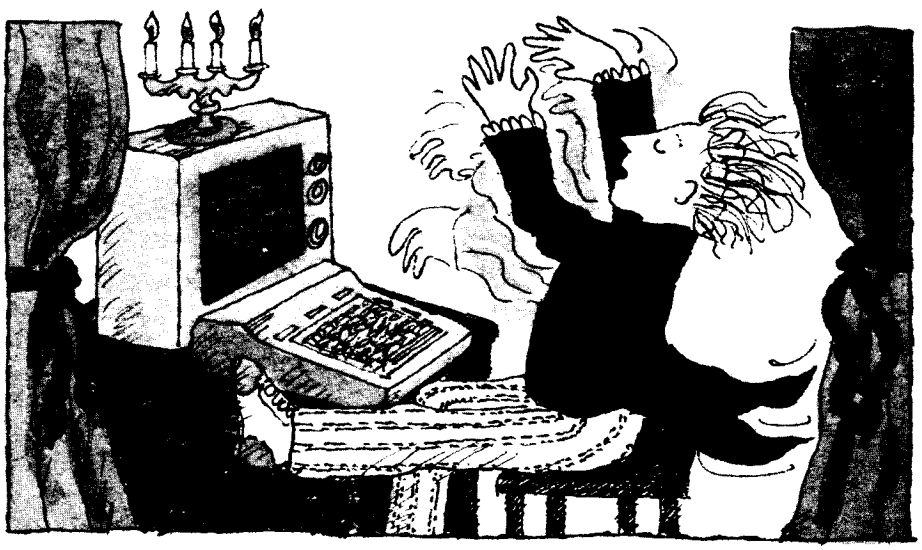
I'm especially pleased to thank the many children who tried out the book, worked out every exercise, and urged me to make it clearer, especially Chris Barth, Walter Ellis IV, Eve and Mema Roussopolous, Sara Shneiderman, and Warren Tildon.

Contents

```
DEDICATED TO SARA AND ANNA
AND THEIR FRIENDS
AND THEIR FRIENDS
AND THEIR FRIENDS
AND THEIR FRIENDS
AND THEIR FRIENDS
AND THEIR FRIENDS
AND THEIR FRIENDS
```

```
10 PRINT "DEDICATED TO SARA AND ANNA"
20 FOR I = 1 TO 7
30 PRINT "AND THEIR FRIENDS"
40 NEXT I
```

Let's Get Started



Computer programming is easier to learn than piano playing but more difficult than tic-tac-toe. In both computer programming and piano playing, you begin with simple patterns and steadily improve over a lifetime. Both skills offer a great deal of satisfaction when you can produce just what you want.

Each time you correctly get the computer to do what you want, you'll feel great. You've learned something new and proved to yourself that you are in charge of the computer. In order to create a program that works, you have to

1. understand each problem completely,
2. decide exactly what you want to do,
3. make a step-by-step plan for action,
4. write commands for the computer to carry out your plan, and then
5. carefully test your commands to make sure that they work.

A group of commands is called a *program*. You may write programs to print pictures, play games, add or subtract numbers, print poems, learn science, count money, or find telephone numbers. You may write learning games to help you or your friends, information programs to explain about your school's computer, or joke-telling programs for fun.

There is a lot to learn about computers. There are many kinds of computers and many different ways to use them. Most people use a microcomputer which sits right on a desk or a computer terminal which is connected to a big computer. Some people use a display screen, which looks like a small television, while other people use a terminal that prints on paper.

Most computers and terminals have a keyboard that looks like a typewriter for you to type commands on. A *command* is an instruction to the computer, such as "add 2 and 3." Sometimes there are game paddles, joysticks, touchscreens, or other ways of giving commands to the computer.

When you play computer games, you use a program written by somebody else. A program is a group of commands that makes the computer work. To give commands to the computer, you need to learn a programming language.

There are several hundred programming languages, but a popular and easy-to-learn one is called BASIC (Beginner's All-purpose Symbolic Instruction Code). It was created by John Kemeny and Thomas Kurtz of Dartmouth College in the early 1960s. This book will help you learn computer programming in BASIC.

Most of the BASIC commands in this book will work on every microcomputer. But each microcomputer is slightly different—the number of letters in a line, the number of lines on the screen, abbreviations, and so on. This version of *Let's Learn BASIC* has examples and exercises that fit the ATARI 400, 600, 800, and 1200

Home Computer series. The solutions to the exercises were run on the ATARI 800.

To get started, turn on your ATARI and make sure that it is connected to your screen. Practice typing the letters on the keyboard. In addition to the letters there are special keys such as RETURN, SHIFT, the space bar, CTRL (CONTROL on the 1200), DELETE/BACK S (DELETE/BACK SPACE on the 1200), and BREAK. (See Figure 1 for photographs of the ATARI 400, 800, and 800XL keyboards.)

There are four arrow keys that let you move (up, down, left, and right) the *cursor* on the screen. To move the cursor, hold down the CTRL key and then press one of the four arrow keys. The cursor on the ATARI is a rectangle that goes over a letter or a blank space. To get started, just turn on the power switch (on some models you need to put in the BASIC cartridge).

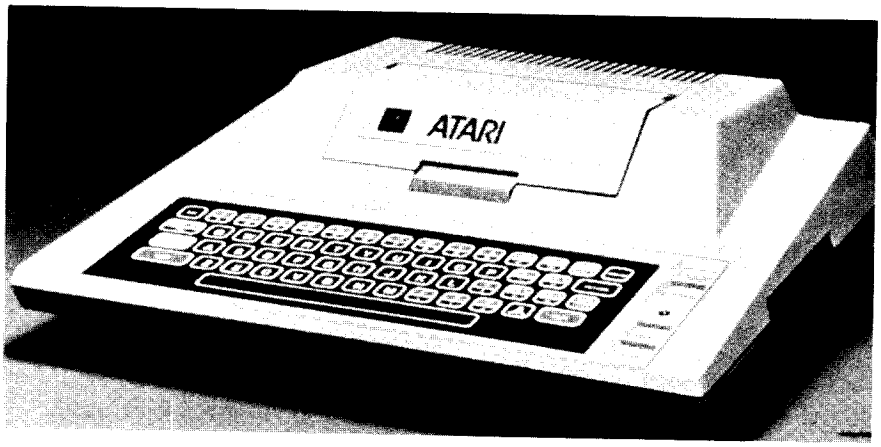
Differences Among Computers. Most of what you learn in this book will work on any computer that uses BASIC:

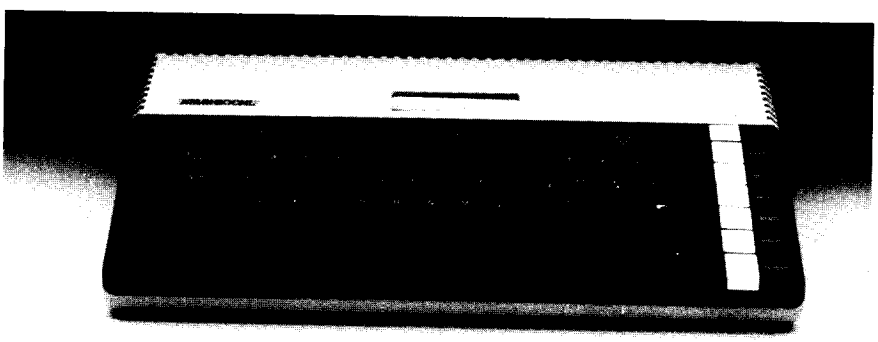
- ATARI 400, 600, 800, and 1200 Home Computer series
- Apple II, II+, and IIe
- Commodore 64
- IBM PC, PCjr, and XT
- Texas Instruments (TI) 99/4A and 99/2
- Radio Shack TRS–80 Color Computer, Model 100, etc.
- Timex/Sinclair 1000, 1500, and 2068
- VIC–20

The differences in the way BASIC works on these computers will be explained in sections like this one.

If someone has been using the computer before you, you may want to turn the power switch off and then back on to clear out their programs.

One of the obvious differences is in the size of the screen. The ATARI


ATARI

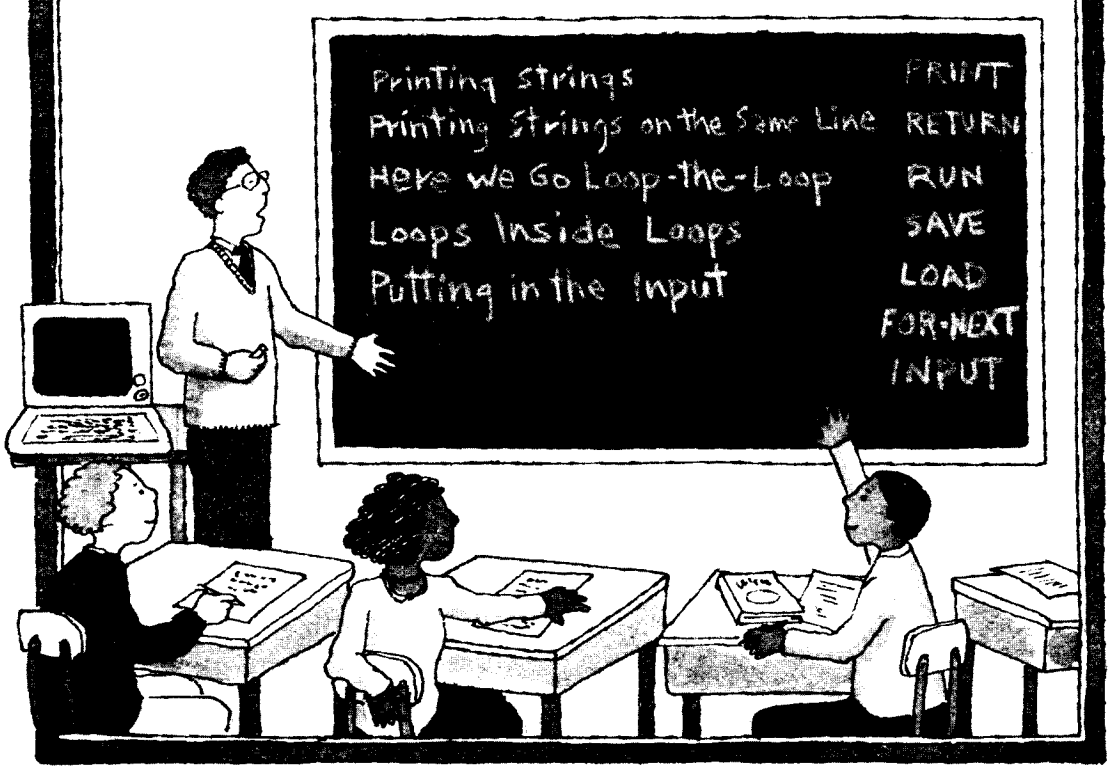


has places for 40 characters on the screen, but only 38 are used in BASIC. The Apple, Commodore 64, and the TRS–80 Model 100 have 40 characters in each line, while the Timex, TI, and TRS–80 Color Computers have 32 characters. The IBM has 40 or 80 depending on the computer and the screen. ■

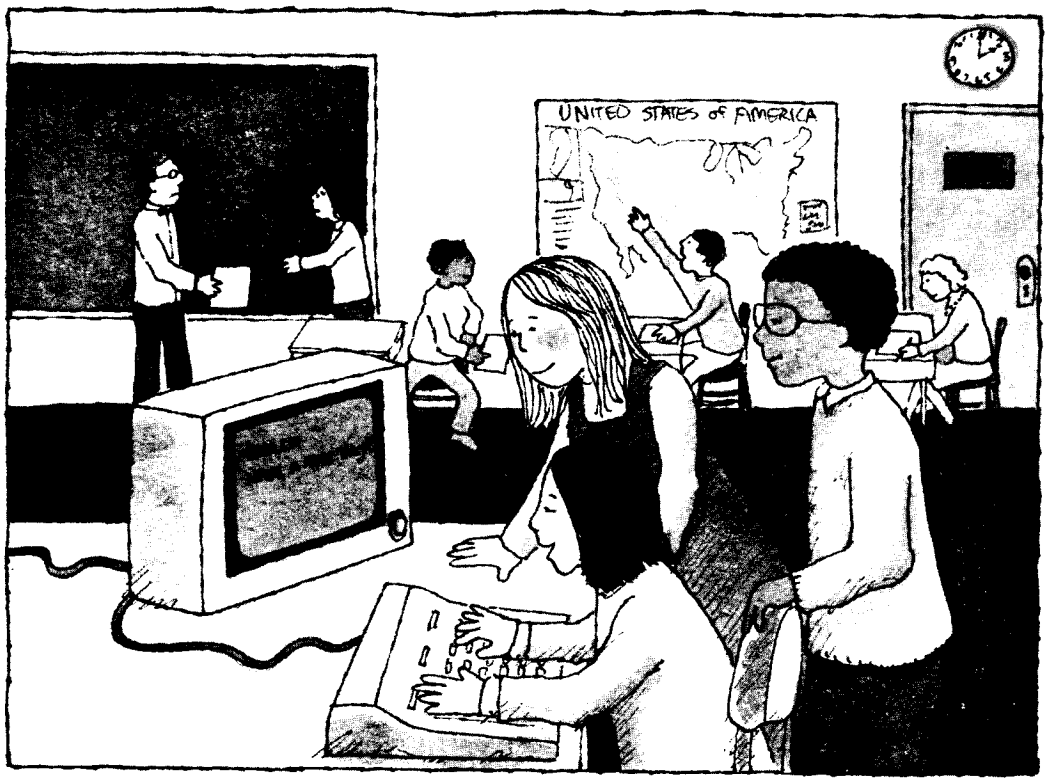
When you have finished studying and have tried the keys, you are ready for Chapter 1. The ATARI is ready for your commands when the word READY appears with the cursor below it.

Figure 1. Keyboards for the ATARI 400 (top), 800 (middle), and 800XL (bottom). (ATARI® 400, 800, and 800XL are trademarks of Atari, Inc. Copyright ©1984.)

Getting the Computer to Do Printing



1 Printing Strings



- **PREVIEW**: The first command you will give to the computer is to print the word HELLO on the screen. You type in the command and then get the computer to carry out your command.
- **NEW IDEAS**: command, line number, PRINT, string, RETURN, stored program, RUN

Your First Command, Your First Program

The first exercise in programming is to get the computer to print words, such as HELLO on the screen. Sit down in front of the keyboard and type the *command*

```
10 PRINT "HELLO"
```

The 10 is the *line number*, and PRINT is the command that gets the computer to print. The word inside the quotes, HELLO, is what the computer will print. A word or any letters between a pair of quotes is called a *string*.

Now, press the key marked RETURN to store your command in the computer. The stored command stays in the computer, and the computer is now ready to take your next command. Your *stored program* is made up of the one PRINT command you typed. To get the computer to carry out your one PRINT command, type

```
RUN
```

and press the RETURN key. The computer prints the word

```
HELLO
```

If it didn't work, try again. Type the 10 PRINT "HELLO" and RUN commands again. If you get a message like ERROR, check the line and retype it. If you notice a mistake while typing a line, just press the DELETE/BACK S key to erase the wrong letters.

Congratulations! You have written and run your first program.

Differences Among Computers. The ATARI, Apple, and Commodore 64 use RETURN, but Timex and TI use ENTER, and IBM uses a picture of an arrow that goes down and to the left.

If you have made a mistake in typing a command, you will get a message that starts with the line number, has the word ERROR, and then shows the part of the line that the computer could not accept. Look

closely at what you have typed, and try again to get it right. Check to make sure you were careful in using the number zero and the letter O. ■

▶ **PREVIEW (for second part of Chapter 1):** You give line numbers to commands so that they can be kept in order, changed, and deleted. Your commands make the computer print words and messages on one or more lines. A group of commands is called a program. You can have the computer show your program on the screen to check it over. You can make the computer carry out your commands and see if your program does what you want it to do.

▶ **NEW IDEAS**: group of commands, LIST, adding, changing, deleting, NEW

Adding a Second Command

You can add a second command to your program by typing

```
20 PRINT "GOODBYE"
```

and pressing RETURN. The line number 20 means that this command comes after line 10. This PRINT command gets the computer to print the string GOODBYE on the next line. To see the entire program, type the command

```
LIST
```

and press RETURN. The computer displays

```
10 PRINT "HELLO"
20 PRINT "GOODBYE"
```

To have the computer RUN your program, type

```
RUN
```

and press RETURN. The computer carries out your two commands and displays

```
HELLO
GOODBYE
```

You can use any whole numbers as line numbers. You can also put as many blanks as you wish before or after the PRINT command. These programs do exactly the same thing:

```
10 PRINT "LUKE"
20 PRINT "SKYWALKER"
```

```
3    PRINT "LUKE"
4 PRINT     "SKYWALKER"
```

```
77 PRINT "LUKE"
4321 PRINT "SKYWALKER"
```

They all print

```
LUKE
SKYWALKER
```

There are a few rules to remember in numbering your commands:

1. each command must have its own number—no two commands can have the same number;
2. the numbers should get larger from one line to the next; and
3. you should leave enough space between numbers so that you can add new commands. Many people like to number lines by tens (10, 20, 30 . . .) so that they can add lines (15, 17, 18 . . .).

Adding More Commands

You can add a third command between the two lines in your

program by typing

```
15 PRINT "HAVE A NICE DAY"
```

and pressing RETURN to store it. Because the number 15 is between 10 and 20, the computer will put the new command in between the two old ones. This new command, number 15, makes the computer print four words: HAVE A NICE DAY. Now type

```
LIST
```

and press RETURN to display the entire three command program:

```
10 PRINT "HELLO"
15 PRINT "HAVE A NICE DAY"
20 PRINT "GOODBYE"
```

To get the computer to carry out your commands, type

```
RUN
```

and press RETURN. The computer shows the results of running your program:

```
HELLO
HAVE A NICE DAY
GOODBYE
```

If you would like an extra blank line to print out between the second and third lines, you would add another command:

```
17 PRINT
```

By now you probably know that you have to press RETURN after every command. If you ask for the LIST and press RETURN, you get

```
10 PRINT "HELLO"
15 PRINT "HAVE A NICE DAY"
17 PRINT
20 PRINT "GOODBYE"
```

Now when you type RUN and press RETURN, you get

```
HELLO
HAVE A NICE DAY

GOODBYE
```

Changing and Deleting Commands

If you decide to change a command, just retype it:

```
10 PRINT "HI"
```

The new command number 10 replaces the old one. This is why it is important to give each line a different number. Now type in LIST to see if your stored program is what you expected:

```
10 PRINT "HI"
15 PRINT "HAVE A NICE DAY"
17 PRINT
20 PRINT "GOODBYE"
```

To get rid of or delete a command, type just the line number:

```
17
```

When you press RETURN, the command will no longer be part of your program. You can delete another command

```
20
```

and check your work by typing

```
LIST
```

which will display the remaining program:

```
10 PRINT "HI"
15 PRINT "HAVE A NICE DAY"
```

Can you predict what happens when you type RUN?

Instead of printing words, you can get your computer to print shapes, too. You can get this little diamond shape

```
  X
 X X
X   X
 X X
  X
```

by simply printing it out with five PRINT commands. Spacing is very important in this program so that you come out with the right shape. In line 10 you must have exactly two blanks between the " and the X. In line 15 there must be exactly one blank after the " and after the first X. In line 20 there must be exactly three blanks between the two X's.

```
10 PRINT "  X  "
15 PRINT " X X "
20 PRINT "X   X"
25 PRINT " X X "
30 PRINT "  X  "
```

It may not be a valuable diamond, but it's a jewel of a program.

Differences Among Computers. The ATARI and many other computers let you abbreviate the word PRINT by a single question mark, ?. ■

Starting Fresh

If you want to clear out all the commands in your program, you could delete one at a time by typing each of the line numbers. A shortcut is to type

```
NEW
```

which deletes all commands at once. Now you can start over and type this program:

```
10 PRINT "ROSES ARE RED"
20 PRINT "VIOLETS ARE BLUE"
30 PRINT "SUGAR IS SWEET"
40 PRINT "AND SO ARE YOU"
50 PRINT
60 PRINT "   BYE BYE"
```

This program has six commands. Five of the commands print a string, but line 50 just prints a blank line. In line 60 the three blanks between the " and the BYE BYE make this closing message appear three spaces over to the right.

Computers can do a lot more than print strings, but you've made a good start in learning programming.

Summary

1. You can use the PRINT command to write letters, words, and messages.
2. Commands are kept in order by line number.
3. The LIST command displays all the lines in a program.
4. The RUN command gets the computer to carry out your program.
5. You can delete each numbered command one at a time. You can delete the whole program with the NEW command.

Exercises

Each chapter ends with exercises for you to try on your computer. Read each exercise until you understand it, make a step-by-step plan, write commands, and test your program carefully.

If the exercise has an asterisk (*) in front of it, that means it is

more difficult. If the exercise has a plus (+) in front of it, that means the solution is in the back of the book. I hope you enjoy doing the exercises.

1. Write a program with one command to print your first name. Then add a second command to print your last name on another line.
2. (+) Write a program to print the alphabet like this:

```
ABCDEFG
HIJKLMNOP
QRSTUVWXYZ
```

3. Write a program to print this saying:

```
A STITCH IN TIME
SAVES NINE
```

4. (+) Write a program to print a rectangle in which there are exactly seven spaces between the two R's in the second and third rows:

```
RRRRRRRRR
R       R
R       R
RRRRRRRRR
```

5. Write a program to print this poem:

```
TWINKLE, TWINKLE LITTLE STAR
HOW I WONDER WHAT YOU ARE
UP ABOVE THE WORLD SO HIGH
LIKE A DIAMOND IN THE SKY
```

6. Now, how about printing this star to go with the program in Exercise 5. There are four blanks before the A on the first line and three blanks before the A on the second line. You will have to count the As and the blanks carefully.

```
    A
   AAA
AAAAAAAAA
 AAAAAAA
AAAAAAAAA
   AAA
    A
```

7. (*+) Write a program to draw this picture of a cat. Be sure to count the spaces carefully—this exercise is tricky.

```
 M     M
M M   M M
I - - - I
I       I
I O   O I
I       I
 I -O- I
 I     I
   III
```

2 Printing Strings on the Same Line

- **PREVIEW**: Sometimes you will want to print two or more strings on one line. Strings can be printed in columns to produce neat lists. Programs can be saved on a cassette tape player or a disk drive.
- **NEW IDEAS**: printing in columns, cassette tape player, disk drive, magnetic tape, magnetic disk, floppy disk, `CSAVE`, `CLOAD`, `SAVE`, `LOAD`.

Printing Lists

Keeping track of information such as school grades, library books, or addresses is a common use of computers. You might want to write a program to print your friends' names and phone numbers. To get rid of your old program and start a new one, give the command

```
NEW
```

Now you are ready to type in program commands. If you have four friends, you might write a program like this:

```
10 PRINT  "ANNA 928-3744"
20 PRINT  "ELIZA 485-4788"
30 PRINT  "SARA 744-0902"
40 PRINT  "WENDY 864-3435"
```

If you type it this way, you put the names in alphabetical order. If you run this program, the output is

```
ANNA 928-3744
ELIZA 485-4788
SARA 744-0902
WENDY 864-3435
```

As the list gets longer, it would look much neater if all the phone numbers are lined up in a column:

```
ANNA        928-3744
ELIZA       485-4788
SARA        744-0902
WENDY       864-3435
```

This output also makes it easier to spot an extra or missing digit in the telephone number.

To print the list in two columns, each print command should have two strings: the name and the telephone number.

```
10 PRINT "ANNA", "928–3744"
20 PRINT "ELIZA", "485–4788"
30 PRINT "SARA", "744–0902"
40 PRINT "WENDY", "864–3435"
```

Each of the two strings is enclosed in quotes and separated by a comma. The blank after the comma is not necessary but makes the program easier to read. The comma is the part of the PRINT command that tells the computer to start a new column for the next string. The ATARI has 10 spaces in each column, so the telephone numbers begin in the 11th space.

Now, if you want to add a new friend to your list and still keep the alphabetical order, you could just choose the right line number. For example, to add JEREMY, any line number in the 20s would be fine:

```
25 PRINT "JEREMY", "244–3809"
```

Your program now looks like this:

```
10 PRINT "ANNA", "928–3744"
20 PRINT "ELIZA", "485–4788"
25 PRINT "JEREMY", "244–3809"
30 PRINT "SARA", "744–0902"
40 PRINT "WENDY", "864–3435"
```

Can you predict what the five lines of output for this program would be?

Differences Among Computers. The ATARI and Commodore 64 have 10 letters in each column. The IBM and TI have 14, and the Apple and Timex have 16. ■

Printing Column Headings

A nice touch would be to add column headings for the list:

```
5 PRINT "NAME", "TELEPHONE"
6 PRINT
```

The comma in line 5 makes the word NAME appear in the first column and the word TELEPHONE in the second column. The PRINT in line 6 leaves a blank line in the output. Remember, the second column on the ATARI begins in the 11th space from the left side. The final output looks neatly labeled by column:

```
NAME        TELEPHONE

ANNA        928–3744
ELIZA       485–4788
JEREMY      244–3809
SARA        744–0902
WENDY       864–3435
```

You can print lists for many different things. You could keep lists of books and authors, homework assignments and due dates, toys you want and their prices, or birthdays of family and friends.

Since your ATARI and many computers have narrow screens (with space for 38 or fewer letters), you may have to print some items on several lines. The title of a book could be on one line, and the author, date of publication, and number of pages could be on the next line. To make the list more readable, you could print a blank line before each book title. The ATARI has room for 40 characters on the screen, but in ATARI BASIC the first two spaces are left blank. That means that you can only display 38 characters on a line.

Saving Your Program

Getting a nice display or printed copy of a list is useful, but if you turn off your computer, the list will be gone forever. To avoid having to retype the program each time you want to use it, you need

to SAVE the program. Your ATARI can be connected to a *cassette tape player* (such as the ATARI 410 Program Recorder) or a *disk drive* (such as the ATARI 810). These devices contain either *magnetic tape* or a *magnetic disk* which can store a copy of your program. Many microcomputers use a soft magnetic disk called a *floppy disk.*

On the ATARI, you can save a copy of your program on a cassette tape player by typing:

```
CSAVE
```

After you press RETURN you will hear two beeps. Put a blank cassette in the Program Recorder and press the PLAY and RECORD keys at the same time. Now press RETURN on the computer. You will hear a high pitched sound, then a raspy sound while the program is being copied to the tape. When the noise stops the word READY should show on your screen. Press the STOP key on the Program Recorder and rewind the tape so that you can read the program back when you want it. Keep track of which programs you have saved by writing the names down on a sheet of paper.

To retrieve an old program that has been stored away, first make sure that the tape is rewound to the beginning. Type the command:

```
CLOAD
```

After you press RETURN you will hear one beep from the computer. Press the PLAY key on the cassette tape player and RETURN on the computer. You should soon begin to hear the raspy sound of your program being loaded into the computer's storage. When the loading is done, you will get the READY message on your screen. Press STOP on the Program Recorder and rewind the tape so it will be ready when you need it again.

Now you can try the LIST command or just go ahead and RUN the program.

If you have a disk drive, then saving and loading programs is

even simpler. When storing on a disk drive you have to give a name for your program such as PHONES. Instead of CSAVE and CLOAD you use these commands:

```
SAVE "D:PHONES"
LOAD "D:PHONES"
```

You will hear the disk drive spinning and the computer will beep. You do not have to press any buttons, the disk drive will come on automatically. To get a list of the programs on your disk, use the Disk Operating System (DOS) menu option "A".

Differences Among Computers. You can check in your computer manual to learn how to SAVE and LOAD programs with your cassette tape player or magnetic disk drive.

The commands for using a disk are different on each computer. For example, you get a list of programs by typing LOAD "$",8 (Commodore 64), CATALOG (Apple), or FILES (IBM). ■

Summary

1. You can print two or more strings on the same line in neat columns.
2. The comma separates the strings in the PRINT command.
3. The CSAVE or SAVE command stores a program on tape or disk.
4. The CLOAD or LOAD command retrieves an old program that has been stored on tape or disk.

Exercises

Remember, an * means that the exercise is more difficult and a + means that the solution is in the back of the book.

1. (+) Write a program to print out this list of abbreviations in two columns:

```
KG          KILOGRAM
CM          CENTIMETER
MM          MILLIMETER
```

2. Write a program to print out this list of birthdates:

```
NAME        MONTH       DAY

ANDREW      MARCH       25
TOM         OCTOBER     6
BETTY       JUNE        12
```

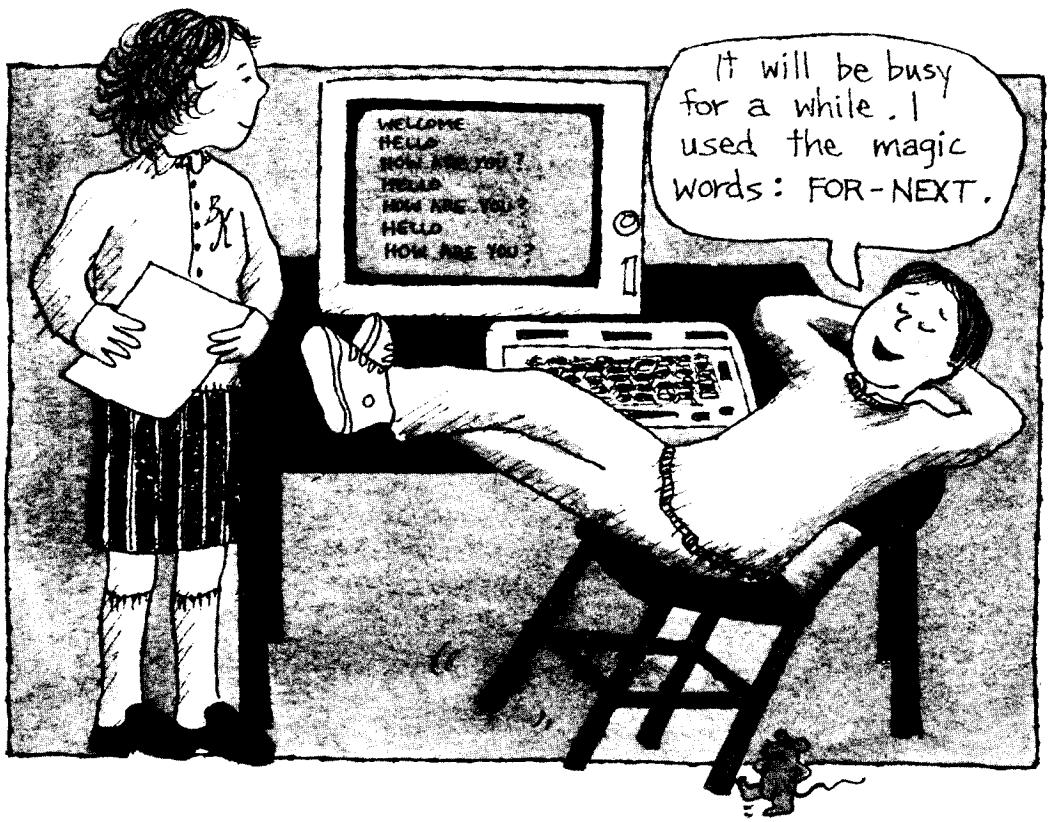
3. (+) Write a program to print these little flags spread out in three columns:

```
+000000     +XXXXXX     +IIIIII
+000000     +XXXXXX     +IIIIII
+000000     +XXXXXX     +IIIIII
+           +           +
+           +           +
+           +           +
```

4. (*) Write a program to print a book list with titles, authors, year of publication, and number of pages. Include books that you have read.

```
THE MAGIC OF OZ
L. FRANK BAUM          1919        208

THE SECRET OF PIRATES HILL
FRANKLIN W. DIXON      1972        178

CHARLOTTE'S WEB
E. B. WHITE            1952        184
```

3 Here We Go Loop-the-Loop



▶ **PREVIEW**: The computer's power is best used when a simple action can be repeated. In this chapter you will learn how to repeat printing a string 2, 10, 500, or more times.

▶ **NEW IDEAS**: loops, counter variable, `FOR–NEXT` loops

The Power of Repetition

The great advantage of using a computer shines through when you can make the computer repeat some action again and again. Once you write a program to print a picture, multiply numbers, make a move in a game, or convert inches to centimeters, it's easy to make the computer repeat the same action many times.

Our first program printed the word HELLO once. If you want to print a string four times, you wrap a repetition *loop* around the command. It's called a loop because the computer goes around and around repeating the same commands again and again.

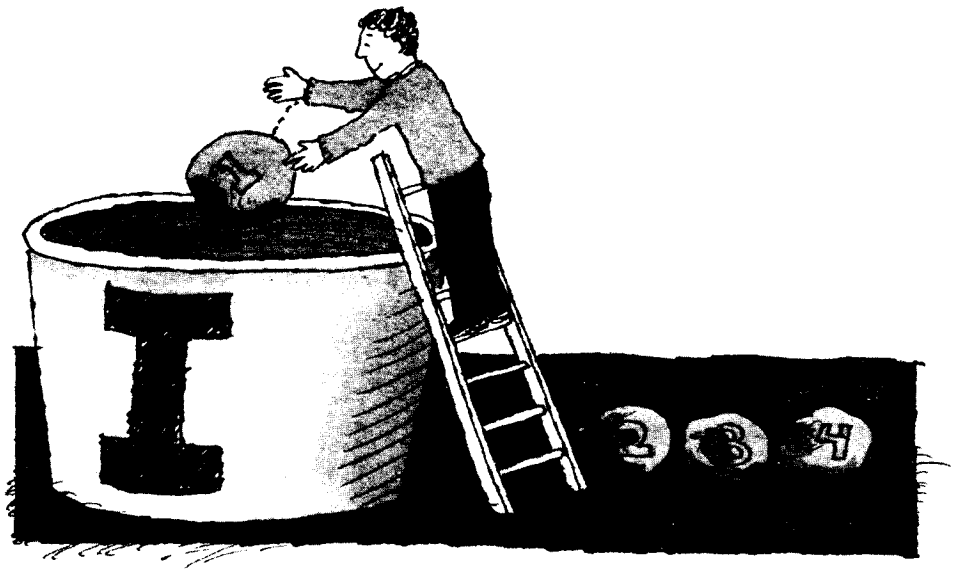
The repetition loop begins with a FOR command and ends with a NEXT command:

```
10 FOR I = 1 TO 4
20 PRINT "HELLO"
30 NEXT I
```

The FOR command at line 10 makes the computer count from 1 up to 4 using the *counter variable* named I. Imagine a cup labeled I into which you drop stones. Every time you drop in one of the stones, the computer carries out the commands until it reaches the NEXT command. If the cup contains the full number of stones, in this case 4, the computer goes on to the line following the NEXT command. If the cup contains fewer than 4 stones, the computer adds one and goes through the loop again. The output of this program is

```
HELLO
HELLO
HELLO
HELLO
```

The counter variable in the FOR command must match the counter variable in the NEXT command. To make it easy, we'll use a single letter for counter variables. The counter variable in this first



program is I. It is the name of the imaginary cup for counting from 1 to 4.

To print HELLO six times, change the FOR command so that I goes from 1 to 6:

```
10 FOR I = 1 TO 6
20 PRINT "HELLO"
30 NEXT I
```

Just to see how this all works, you can add PRINT commands before and after the FOR–NEXT loop

```
7 PRINT "WELCOME"
10 FOR I = 1 TO 6
20 PRINT "HELLO"
30 NEXT I
40 PRINT "GOODBYE"
```

When you type RUN, you get this output:

```
WELCOME
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
GOODBYE
```

To print 20 HELLOs, just change the FOR command to go up to 20.

If you added another command inside the loop, it would get repeated, too. Let's add line 25:

```
7 PRINT "WELCOME"
10 FOR I = 1 TO 6
20 PRINT "HELLO"
25 PRINT "HOW ARE YOU?"
30 NEXT I
40 PRINT "GOODBYE"
```

And then the output is

```
WELCOME
HELLO
HOW ARE YOU?
HELLO
HOW ARE YOU?
HELLO
HOW ARE YOU?
HELLO
HOW ARE YOU?
HELLO
HOW ARE YOU?
HELLO
HOW ARE YOU?
GOODBYE
```

I'm fine, I'm fine, I'm fine, I'm fine, I'm fine, I'm fine.

Shapes, Too

Once you have repetition, you can make shapes much more easily. A rectangle is made by repeating a line of symbols or letters such as the A. The program might be

```
10 FOR I = 1 TO 5
20 PRINT "AAAAAAAAAA"
30 NEXT I
```

When you RUN this program, you get

```
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
```

To get a rectangle that is empty in the middle requires a little more planning. If the goal is

```
RRRRRRRRR
R       R
R       R
R       R
RRRRRRRRR
```

you would have to get the top line done with a simple PRINT command, then loop three times to get the sides (with exactly seven blank spaces between the two R's) and finally get the bottom line with another simple PRINT command

```
10 PRINT "RRRRRRRRRR"
20 FOR I = 1 TO 3
30 PRINT "R        R"
40 NEXT I
50 PRINT "RRRRRRRRRR"
```

What if you wanted the rectangle to be moved over to the right? You could add blanks to the PRINT commands in lines 10, 30, and 50.

```
10 PRINT "          RRRRRRRRRR"
20 FOR I = 1 TO 3
30 PRINT "          R        R"
40 NEXT I
50 PRINT "          RRRRRRRRRR"
```

You could also just print a string with a single blank, before skipping over. This would cause the rectangle to start in the second column:

```
10 PRINT " ", "RRRRRRRRRR"
20 FOR I = 1 TO 3
30 PRINT " ", "R        R"
40 NEXT I
50 PRINT " ", "RRRRRRRRRR"
```

To make a taller rectangle, you could just change the FOR command to go from 1 up to 15, or more.

Snack Time

This section uses the FOR–NEXT loop to print out a fork and a layer cake. Let's print the fork shifted over into the second printing column, like this:

```
F  F  F  F
F  F  F  F
F  F  F  F
F  F  F  F
F  F  F  F
F  F  F  F
FFFFFFFFFF
    FF
    FF
    FF
    FF
    FF
    FF
    FF
    FF
    FF
    FF
    FF
    FF
    FF
    FF
```

The four tines (tips of the fork) are produced by repeating the same pattern for six lines:

```
10 FOR I = 1 TO 6
20 PRINT " ", "F  F  F  F"
30 NEXT I
```

In line 20 the " " and the comma space the fork over to the second column, and there are two blanks between each F. The line that joins the tines is produced by a PRINT command:

```
40 PRINT " ", "FFFFFFFFFF"
```

The long handle is made of 14 lines of two Fs, which are four spaces in from the left side. Line 60 prints a blank in the first column and uses the comma to skip to the second column.

```
50 FOR I = 1 TO 14
60 PRINT " ", "    FF"
70 NEXT I
```

The four blanks before the FF are necessary to center the fork handle.

Now that your appetite is growing, it's time for the three-layer cake:

```
        AAAAAAAA
        AAAAAAAA
        AAAAAAAA
        AAAAAAAA
        AAAAAAAA
    BBBBBBBBBBBBB
    BBBBBBBBBBBBB
    BBBBBBBBBBBBB
    BBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCC
```

Three sets of FOR–NEXT loops will be all that you need to bake this cake. The top layer begins with six blanks and then has eight A's, the middle layer has three blanks and 14 B's, and the bottom layer has 20 C's:

```
10 FOR I = 1 TO 5
20 PRINT "      AAAAAAAA"
30 NEXT I
40 FOR I = 1 TO 4
50 PRINT "   BBBBBBBBBBBBB"
```

```
60 NEXT I
70 FOR I = 1 TO 3
80 PRINT "CCCCCCCCCCCCCCCCCCCC"
90 NEXT I
```

Hearty appetite!

Summary

1. FOR–NEXT loops cause the commands between them to be repeated. The command to repeat six times is FOR I = 1 TO 6. The counter variable I goes from 1 up to 6.
2. With clever use of FOR–NEXT loops and PRINT commands, you can print pictures.
3. Each letter or blank inside the quotes of a PRINT command must be counted carefully.

Exercises

1. Write a program to print your first name 7 times and then your last name once.
2. (+) Write a program to print stripes like these:

```
00000000000000
XXXXXXXXXXXXXX
00000000000000
XXXXXXXXXXXXXX
00000000000000
XXXXXXXXXXXXXX
```

   You'll need two PRINT commands inside one FOR–NEXT loop.
3. Write a program to print this rectangle with dots on the inside:

```
00000000
0......0
0......0
0......0
0......0
0......0
0......0
00000000
```

4. (+) Write a program to print this flag:

```
+---------I
+---------I
+---------I
+---------I
+---------I
+---------I
+
+
+
+
+
+
+
+
```

5. (*) The game called "dots" is played on a paper filled with a grid of dots like this:

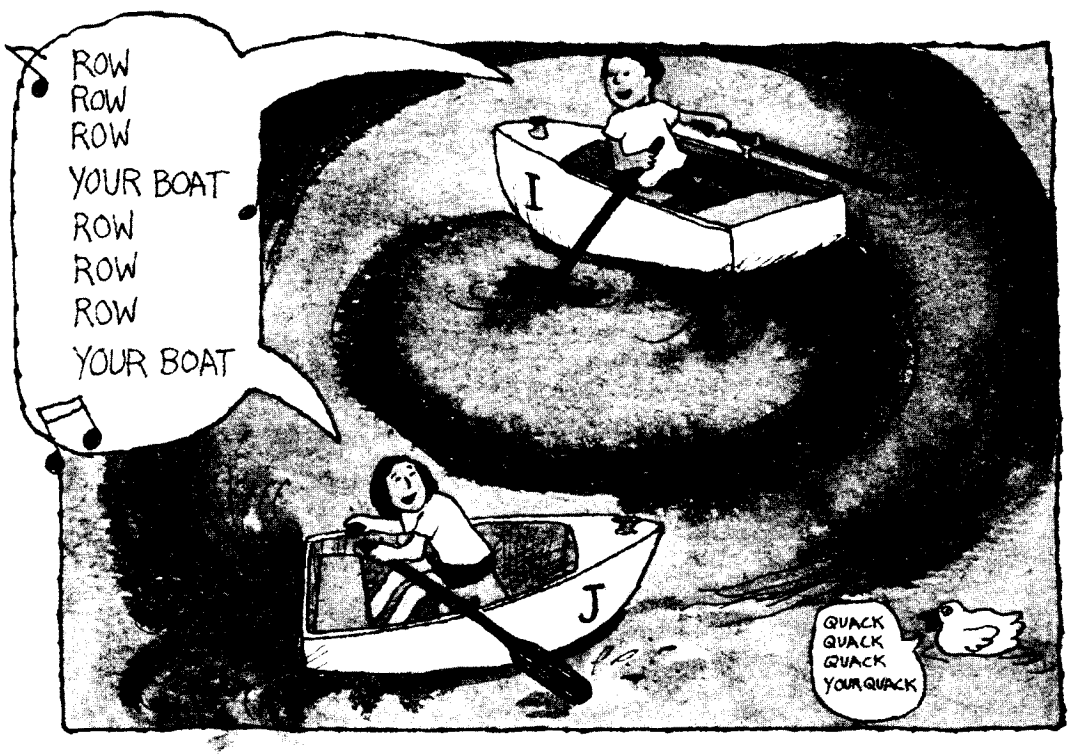
```
.   .   .   .   .   .
.   .   .   .   .   .
.   .   .   .   .   .
.   .   .   .   .   .
```

Using pencils, players take turns adding a line to connect a pair of dots horizontally or vertically. If your line completes a box, you get to put your initials inside the box and you get

another turn. When the board is filled, the player with the most boxes with his or her initials in it is the winner. Write a program to produce the board with four rows of six dots shown above. There are three blanks going across between each dot and a blank line between each row of dots. Then make a board with fifteen rows of six dots.

6. (*) Add another layer of D's to the bottom of the layer cake and a candle to the top.

4 Loops Inside Loops



- **PREVIEW**: In this chapter you will learn how to make some fancier programs by combining loops. You will learn how to control the printing of letters more carefully.
- **NEW IDEAS**: inner loop, outer loop, print control with comma and semicolon

Row, Row, Row Your Boat

Up until now each FOR–NEXT loop was separate from the other loops in the program. Sometimes it is necessary to have one loop inside another. Let's start with a loop around a PRINT command followed by a second PRINT command, to print a line from a familiar song:

```
10 FOR I = 1 TO 3
20 PRINT "ROW"
30 NEXT I
40 PRINT "YOUR BOAT"
```

The output of this program is

```
ROW
ROW
ROW
YOUR BOAT
```

What if you want to repeat this pattern twice? You could write the four commands again. Or you could wrap a FOR–NEXT loop around the whole program:

```
8 FOR J = 1 TO 2
10 FOR I= 1 TO 3      } inner loop
20 PRINT "ROW"        }
30 NEXT I             }
40 PRINT "YOUR BOAT"
48 NEXT J
```

outer loop

The FOR–NEXT loop on lines 10 and 30 is called the *inner loop* and it uses the counter variable I to count from 1 up to 3. The FOR–NEXT loop on lines 8 and 48 is called the *outer loop*, and it must use a different counter variable. The outer loop uses the counter variable J, which goes from 1 up to 2. If you run this program, you get

```
ROW
ROW
ROW
YOUR BOAT
ROW
ROW
ROW
YOUR BOAT
```

If J in line 8 went from 1 up to 7, you would get seven copies of the four-line song. You can change the outer loop counter variable to repeat the inner loop as many times as you want. Remember that the counter variable in the inner loop must be different from the counter variable in the outer loop.

Controlling the Printing

You could have all of the ROWs appear on the same line. You've already learned that a comma between two strings prints them in two columns on the same line. By putting a comma at the end of line 20, you can make the next string print on the same line.

```
8 FOR J = 1 TO 2
10 FOR I = 1 TO 3
20 PRINT "ROW",          — notice the comma
30 NEXT I
40 PRINT "YOUR BOAT"
48 NEXT J
```

With this slight change the program output is

```
ROW         ROW         ROW         YOUR BOA
T
ROW         ROW         ROW         YOUR BOA
T
```

The letter T does not fit on the line, so it is forced onto the next line. Line 40 does not have a comma on the end, so the next string to print will begin on a new line.

You may find that this form is still too spread out and that you would like to put the words right next to each other. To do this, use a *semicolon* (;) instead of a comma. This will make each string print right next to the previous string. The only change is the semicolon in line 20:

```
8 FOR J = 1 TO 2
10 FOR I = 1 TO 3
20 PRINT "ROW";
30 NEXT I
40 PRINT "YOUR BOAT"
48 NEXT J
```

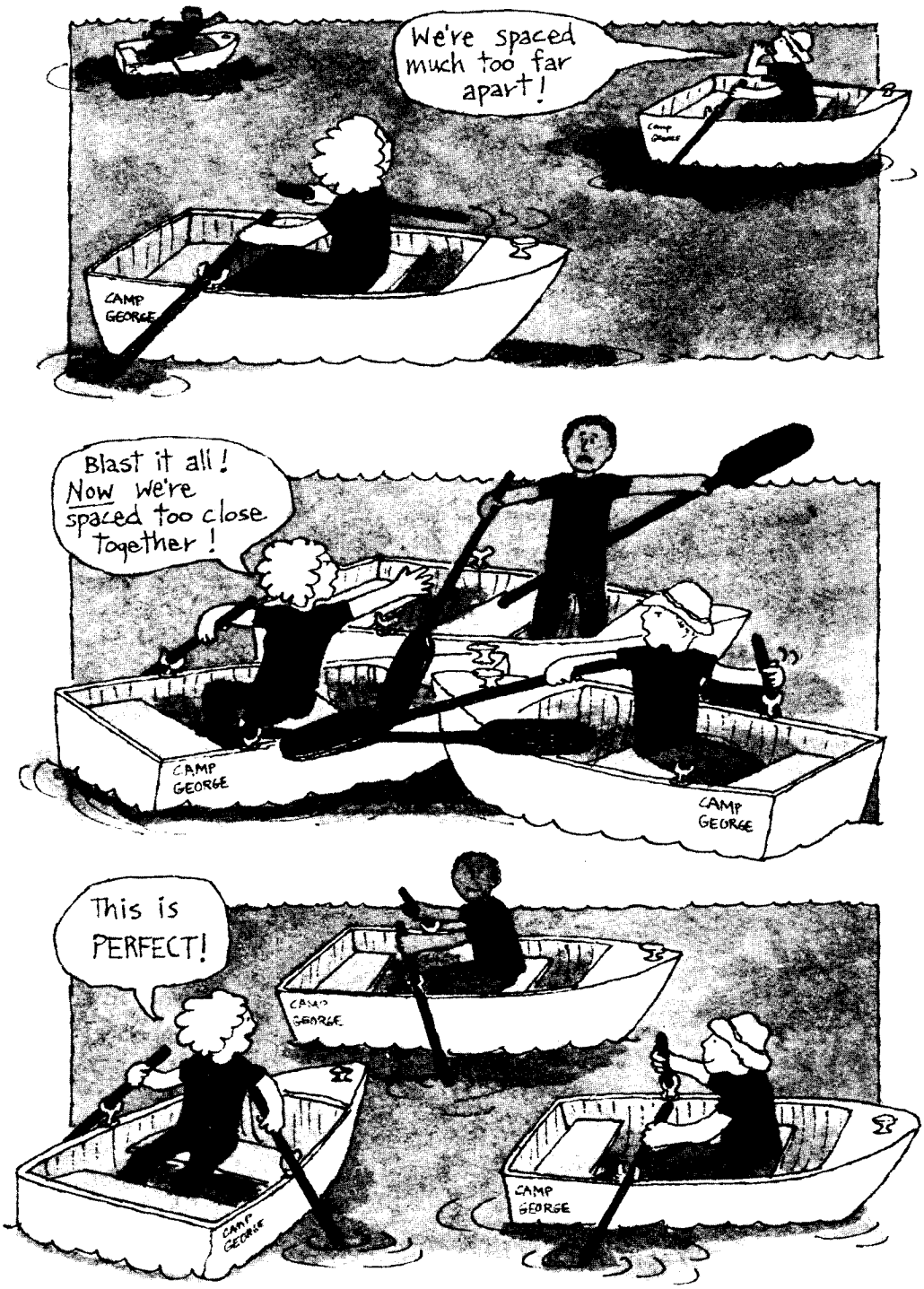
With this change the output is

```
ROWROWROWYOUR BOAT
ROWROWROWYOUR BOAT
```

This output is too squeezed together. There is no space between each of the ROWs. To separate strings you could add a blank right after the W in line 20. You can add one more line to the song with line 60:

```
8 FOR J = 1 TO 2
10 FOR I = 1 TO 3
20 PRINT "ROW ";        — notice the blank after ROW
30 NEXT I
40 PRINT "YOUR BOAT"
48 NEXT J
60 PRINT "GENTLY DOWN THE STREAM"
```

The semicolon at the end of line 20 makes each string print next to the previous string. The first time through the loop "ROW " gets printed on a new line. The next time through the loop "ROW " gets


We're spaced much too far apart!
CAMP GEORGE
Blast it all! Now we're spaced too close together!
CAMP GEORGE
CAMP GEORGE
This is PERFECT!
CAMP GEORGE
CAMP GEORGE
CAMP GEORGE

printed on the same line. Then the third time through the loop "ROW " gets printed again on the same line. When "YOUR BOAT" is printed, it is still on the same line. Line 40 does not end with a semicolon, so "GENTLY DOWN THE STREAM" begins on a new line. The output is

```
ROW ROW ROW YOUR BOAT
ROW ROW ROW YOUR BOAT
GENTLY DOWN THE STREAM
```

Loops are just a dream!

Slow Down, You're Moving too Fast

It's amazing how fast the computer can print out line after line of letters or words, but sometimes you may want to slow things down. Instead of printing

```
HELLO
GOODBYE
```

you may want to have the computer wait a few seconds between printing these two words. One way to do this is to have the computer just count from 1 to 1000 between the PRINT commands:

```
10 PRINT "HELLO"
20 FOR I = 1 TO 1000
30 NEXT I
40 PRINT "GOODBYE"
```

On some computers counting from 1 to 1000 may take less than a second, so you may want to have the computer count up to 10,000. If your computer is slow, you may want to go up to only 100. Try it out on your computer.

One enjoyable use of this idea is to have the computer pretend to be a beating heart:

```
LUB
DUB
(pause here)
LUB
DUB
(pause here)
LUB
DUB
```

You can decide how long a pause you want between each heartbeat. Let's say you find that counting to 500 is about right. To get 20 heartbeats, you could use an outer loop and an inner loop:

```
10 FOR B = 1 TO 20
20 PRINT "LUB"
30 PRINT "DUB"
40 FOR I = 1 TO 500
50 NEXT I
60 NEXT B
```

The counter variable B stands for *beat*. You could speed up or slow down the heartbeat by changing the highest number in the FOR command on line 40.

Your ATARI has a loudspeaker in it. Try to find someone who has used it. Maybe together you can make the sound of a heartbeat.

Summary

1. By wrapping one loop around another loop, you can repeat commands to make complicated patterns.
2. The semicolon in a PRINT command causes two strings to be printed right next to each other.

3. A FOR–NEXT loop without any commands inside the loop can be used to make a slight pause between PRINT commands.

Exercises

1. (+) Write a program to print

```
NO
NO
YES
NO
NO
YES
NO
NO
YES
```

2. Write a program to print

```
SHE LOVES ME
SHE LOVES ME
NOT

SHE LOVES ME
SHE LOVES ME
NOT

SHE LOVES ME
SHE LOVES ME
NOT
```

3. (+) Write a program to print this flag. (Hint: you can do it using three loops.)

```
+XXXXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXXXX
+000000000000000000
+XXXXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXXXX
+000000000000000000
+
+
+
+
+
+
+
```

4. (+) Write a program that imitates a clock by printing

```
TICTOC
TICTOC
TICTOC
TICTOC
TICTOC
TICTOC
TICTOC
TICTOC
TICTOC
TICTOC
```

Figure out how long a pause to put in so that one line is printed every second. The 10 lines should take 10 seconds. You'll need a watch that shows seconds to see if your program is accurate.

5. (*) Here's another challenge. Print this ladder:

```
I           I
I           I
I           I
000000000000
I           I
I           I
I           I
000000000000
I           I
I           I
I           I
000000000000
I           I
I           I
I           I
000000000000
I           I
I           I
I           I
```

Have a safe climb to the top.

5 Putting in the Input

▶ **PREVIEW**: In all the programs you have written until now, you could make changes as you wished until you typed in the RUN command. Sometimes it is useful to control a program while it is running. In this chapter you will learn how to take in numbers and strings to control a running program. You will have to reserve space for input variables that store strings.

▶ **NEW IDEAS**: INPUT, input variables for strings and numbers, controlling loops, BREAK key, DIM

The Riddle Machine

Everyone likes funny riddles. Everyone moans at silly riddles. Now you can turn your computer into a riddle machine which asks a riddle and gives the answer whenever the user presses RETURN.

The computer might display

```
WHAT DID ONE HOT DOG SAY TO THE OTHER?
  PRESS RETURN TO GET THE ANSWER
?
```

and then wait until the user presses RETURN. The question mark shows that the computer is waiting before going on. When the user presses RETURN, the answer is shown:

```
HI, FRANK.
```

To have the computer print a ? and wait, you use the INPUT command. INPUT commands have an *input variable* to store whatever someone types. To reserve space for the input variable you must use a DIM (it stands for *dimension*) command. Since this program only asks the user to press RETURN (the user doesn't type anything), we reserve only one space for A$.

```
10 DIM A$(1)
20 PRINT "WHAT DID ONE HOT DOG SAY ";
30 PRINT "TO THE OTHER?"
40 PRINT "  PRESS RETURN TO GET THE ANSWER"
50 INPUT A$
60 PRINT "HI, FRANK."
```

In line 50 the INPUT command has the input variable A$, which can take a letter, a number, or just the RETURN. The input variable can be any letter of the alphabet followed by a $ (A$, B$, C$ . . .).

A one-riddle riddle machine is nice, but you can expand this program with two riddles or as many as you like:

```
70 PRINT
80 PRINT
90 PRINT "WHAT IS THE BEST THING ";
100 PRINT "TO PUT IN A PIE?"
110 PRINT "PRESS RETURN TO GET THE ANSWER"
120 INPUT A$
130 PRINT "YOUR TEETH."
140 PRINT
150 PRINT
160 PRINT "IF YOU THROW A WHITE STONE IN"
170 PRINT "THE RED SEA, WHAT WILL IT BECOME?"
180 PRINT "PRESS RETURN TO GET THE ANSWER"
190 INPUT A$
200 PRINT "WET."
210 PRINT "THAT'S ALL FOLKS."
```

If the user of the program gets bored, he or she can stop the program by pressing the BREAK key.

Differences Among Computers. On other computers you can end the program after a ? by pressing the BREAK or RESET keys. On the Apple you have to type CTRL–C and RETURN. On the Commodore 64 you have to hold down the RUN/STOP key and press RESTORE. ■

Controlling a Loop

In Chapter 3 the first program printed HELLO four times:

```
10 FOR I = 1 TO 4
20 PRINT "HELLO"
30 NEXT I
```

You could get nine HELLOs by changing the FOR command to `I = 1 TO 9` and then running the program again. Another way of changing a program is to let the user control the FOR loop. An

INPUT command can have an input variable that is used as the upper limit of a FOR loop. The program should begin by asking the user for a number:

```
10 PRINT "HOW MANY HELLOS DO YOU WANT"
20 INPUT N
30 FOR I = 1 TO N
40 PRINT "HELLO"
50 NEXT I
```

The number that the user types is stored in the input variable N. Any letter of the alphabet (A, B, C . . .) can be used for an input variable to store a number. A DIM command is not needed for variables that store a number.

The value the user has given N will also be used as the N in the FOR command in line 30. When you run this program, it will print the message

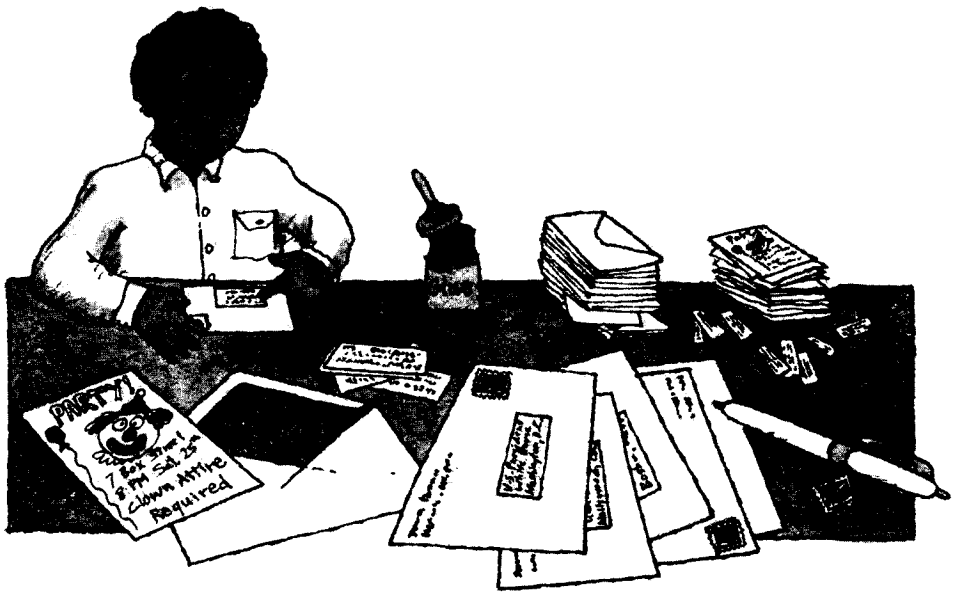
```
HOW MANY HELLOS DO YOU WANT
?
```

and then wait for the user to type a number such as

```
9
```

followed by RETURN. The output is

```
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
```



Printing Address Labels

If you have a printer connected to your ATARI, you can list the output on the printer by using LPRINT instead of PRINT. If you just have a display screen, you can enjoy seeing your address repeated on the screen.

You can control the number of labels printed each time by an INPUT command:

```
10 PRINT "HOW MANY LABELS DO YOU WANT";
20 INPUT N
30 FOR I = 1 TO N
40 PRINT "NANCY DREW"
50 PRINT "48 GHOSTLY LANE"
60 PRINT "MILLTOWN, NY 14226"
70 PRINT
80 NEXT I
```

When this program is RUN, the question in line 10 will be asked. Since there is a semicolon at the end of the PRINT command, the ? from the INPUT command will appear immediately after the question. If the user types 5, then the program loops five times, printing five copies of the address label.

In addition to taking in the number of labels, you could take in the address as well. Start by asking for the number of labels and then ask for the name, street address, city, state, and zip code. Remember that you need to reserve space for input variables which store letters or strings. We'll reserve 30 characters for the name (N$), address (A$), city (C$), and state (S$). The number of labels (N) and the zip code (Z) do not need dimension information.

```
10 DIM N$(30), A$(30), C$(30), S$(30)
20 PRINT "PRESS RETURN AFTER EVERY ITEM"
30 PRINT "HOW MANY LABELS DO YOU WANT ";
40 INPUT N
50 PRINT "TYPE YOUR NAME"
60 INPUT N$
70 PRINT "TYPE YOUR ADDRESS"
80 INPUT A$
90 PRINT "TYPE YOUR CITY"
100 INPUT C$
110 PRINT "TYPE YOUR STATE"
120 INPUT S$
130 PRINT "TYPE YOUR ZIP CODE"
140 INPUT Z
```

The input variable N takes the number of labels. The input variables N$, A$, C$, and S$ will each take a string and store it until it is used. The input variable Z will take the number of the zip code and store it until it is used.

```
150 FOR I = 1 TO N
160 PRINT
170 PRINT N$
```

```
180 PRINT A$
190 PRINT C$; ", "; S$; " "; Z
200 NEXT I
```

The name and street address are printed by lines 170 and 180. The city, state, and zip code are printed by line 190 which puts a comma and a space between the city and state, and a space between the state and the zip code. The semicolons make the strings print next to each other on the same line. Here is how the program runs:

```
PRESS RETURN AFTER EVERY ITEM
HOW MANY LABELS DO YOU WANT ?3
TYPE YOUR NAME
?MISS PIGGY
TYPE YOUR ADDRESS
?77 SESAME STREET
TYPE YOUR CITY
?KERMITSVILLE
TYPE YOUR STATE
?NEW YORK
TYPE YOUR ZIP CODE
?10001

MISS PIGGY
77 SESAME STREET
KERMITSVILLE, NEW YORK 10001

MISS PIGGY
77 SESAME STREET
KERMITSVILLE, NEW YORK 10001

MISS PIGGY
77 SESAME STREET
KERMITSVILLE, NEW YORK 10001
```

We'll use a single letter for variables that store a number. We'll use a single letter followed by a $ for variables that store a string.

Differences Among Computers. On many microcomputers, counter and input variable names can be one letter, two letters, or one letter followed by a number, for example A, AB, or K9. Variables that store strings have a $ on the end, for example A$, AB$, or K9$. On the ATARI you can use longer variable names (up to 120 characters long), for example AGE, INNING9, or FIRSTNAME$. In this book, we'll keep using one letter variable names, because it simplifies typing and reading. Longer and more meaningful variable names are useful when you write programs with tens or hundreds of variable names. ■

Summary

1. In this chapter you learned to use an INPUT command which stops the program until RETURN is pressed.
2. The second use of the INPUT command is to take a number from the user to control a loop or to print later in the program.
3. The third use of the INPUT command is to take a string from the user which can be printed later in the program.
4. When the input variable stores a string, the variable name must have a $ and you must put in a DIM command to reserve enough space.

Exercises

1. Write a program to be a Wisdom Machine. Every time the user presses RETURN, he or she gets another wise saying:

```
A STITCH IN TIME SAVES NINE
  PRESS RETURN FOR ANOTHER SAYING ?
A PENNY SAVED IS A PENNY EARNED
  PRESS RETURN FOR ANOTHER SAYING ?
A WATCHED POT NEVER BOILS
  THE END
```

   You can add another wise saying of your own.

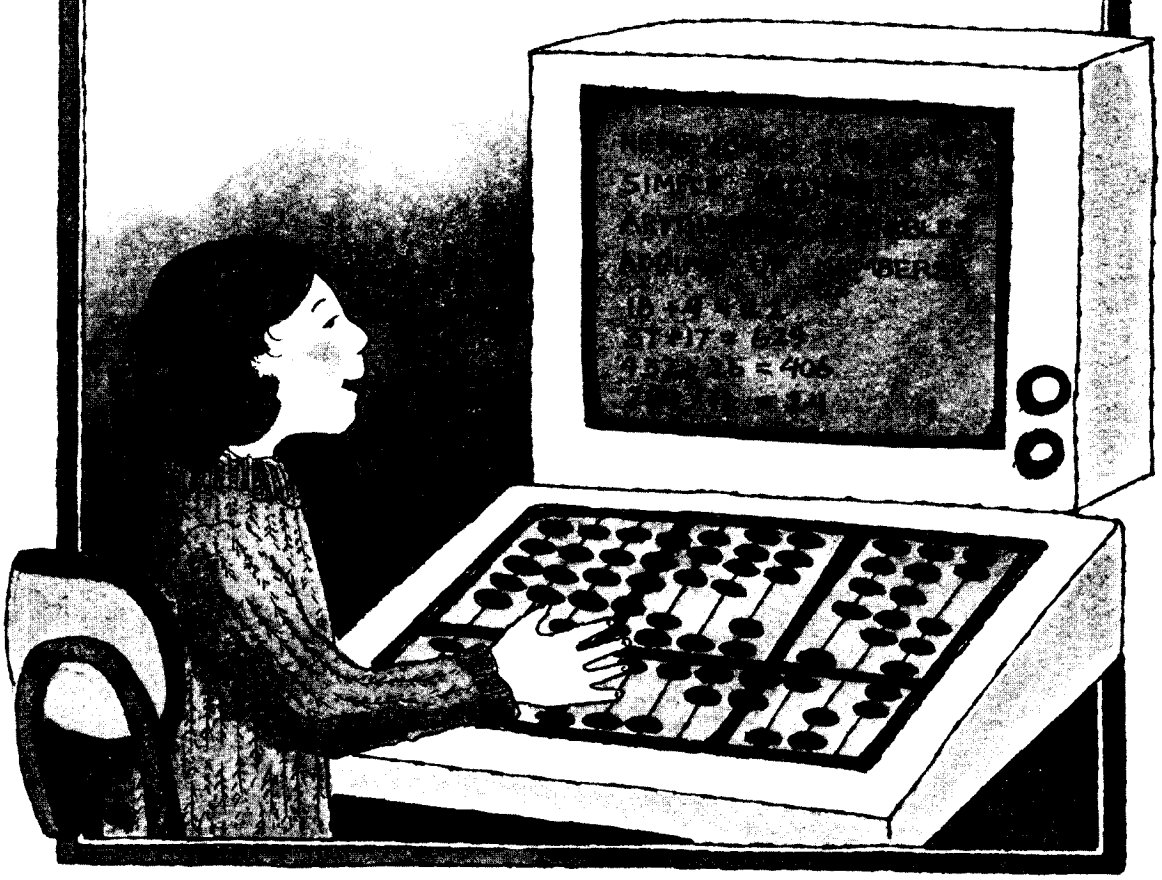
2. (+) In Exercise 4 at the end of Chapter 4 you were asked to make a clock-imitating program that printed TICTOC every second for 10 seconds. Now, ask for the number of seconds that the user wants the clock to run. In this program print TICTOC every second and then print TIME IS UP when the program is done.
3. In Exercise 5 at the end of Chapter 4 you were asked to print a ladder. Write a program that asks the user how many steps he or she would like on the ladder. Then print the right sized ladder.
4. In Exercise 5 at the end of Chapter 3 you were asked to print four rows of six dots each. Can you write a new program with an INPUT command that asks for the number of rows and a second INPUT command asking for the number of dots in each row? Then print the correct number of dots in each row and continue printing for the correct number of rows.
5. (+) Write a program to ask the user to type in a short message, followed by the number of times the message should be printed. For example:

```
TYPE A SHORT MESSAGE AND PRESS RETURN
?MY NAME IS THEA

HOW MANY TIMES DO YOU WANT THIS MESSAGE
?6

MY NAME IS THEA
MY NAME IS THEA
MY NAME IS THEA
MY NAME IS THEA
MY NAME IS THEA
MY NAME IS THEA
```

6. (*) Write a program to ask for a person's first name and then last name. Print out the last name, a comma and a space, and the first name.

```
TYPE YOUR FIRST NAME AND PRESS RETURN
?ROBIN
TYPE YOUR LAST NAME AND PRESS RETURN
?HOOD
YOUR NAME IN THE TELEPHONE BOOK IS
HOOD, ROBIN
```

Making the Computer Do Arithmetic



6 Here Comes the Count

- **PREVIEW**: You've been using the computer to count the number of times through a FOR–NEXT loop. Now you can use this ability to do counting by printing the value of the counter variable. You can also get the computer to count from one number to another number, for example, from 11 to 20.
- **NEW IDEAS**: counting, printing numbers and strings together

Printing the Counter Variable

In Chapter 3 you learned how to print HELLO six times:

```
10 FOR I = 1 TO 6
20 PRINT "HELLO"
30 NEXT I
```

The counter variable I went from 1 up to 6. You can see how many times you print HELLO by printing out the value of the counter variable I. Since you want the value of I, not the letter I, you put the counter variable I in the PRINT command without quotes:

```
10 FOR I = 1 TO 6
20 PRINT "HELLO", I
30 NEXT I
```

When you run this program, it prints

```
HELLO       1
HELLO       2
HELLO       3
HELLO       4
HELLO       5
HELLO       6
```

The string HELLO is printed in the first column and the value of I is printed in the second column.

If you changed the order in the PRINT command

```
20 PRINT I, "HELLO"
```

you would get the numbers in the first column and the string HELLO in the second column. So the output is

```
1           HELLO
2           HELLO
3           HELLO
4           HELLO
```

```
5           HELLO
6           HELLO
```

If you wanted to watch the computer print out the numbers quickly from 1 to 100, you could just have a PRINT command that printed the value of I:

```
10 FOR I = 1 TO 100
20 PRINT I
30 NEXT I
```

But watch carefully. On most computers, this would go very, very quickly.

Printing Strings and Numbers Close Together

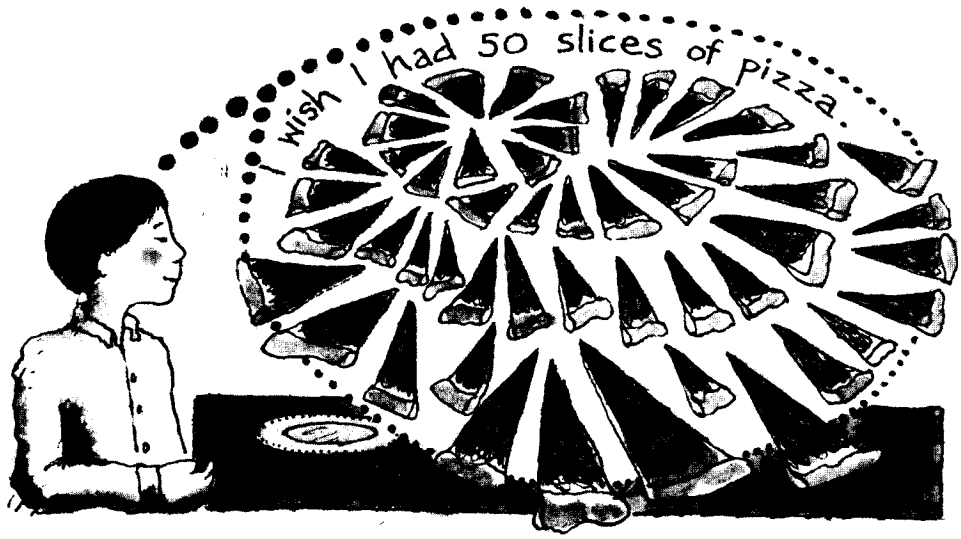
You can bring numbers and strings close together by using the semicolon in the PRINT command. If you put semicolons around the variable, you'll need to put single blanks inside the quotes to make sure that the output has a blank before and after the value of the variable:

```
10 FOR I = 1 TO 4
20 PRINT "I WISH I HAD "; I;
30 PRINT " SLICES OF PIZZA"
40 NEXT I
```

Notice the blank after HAD and before SLICES. When you run this program, you get

```
I WISH I HAD 1 SLICES OF PIZZA
I WISH I HAD 2 SLICES OF PIZZA
I WISH I HAD 3 SLICES OF PIZZA
I WISH I HAD 4 SLICES OF PIZZA
```

It sounds yummy, even though the first line would leave a bad taste with an English grammar teacher.



Printing Numbers Across the Line

So far you've learned how to print strings inside a pair of quotes and the value of a variable. You can also print out numbers that are not inside a pair of quotes. You could just print the number 1984

```
10 PRINT 1984
```

and your computer would simply show the number on the screen. If you printed two numbers separated by a comma

```
10 PRINT 1984, 2001
```

you would get

```
1984        2001
```

The semicolon keeps strings and numbers closely packed together. If you print numbers only, they'll also be kept close together.

```
10 PRINT 1; 2; 3; 4; 5; 6; 7
```

would print as

```
1234567
```

To get the numbers spread out, you will have to include spaces:

```
10 PRINT 1; " "; 2; " "; 3; " "; 4; " "; 5; " "
20 PRINT 6; " "; 7
```

The blanks between the quotes are the ones getting printed, so the line could be written a bit more compactly as

```
10 PRINT 1;" ";2;" ";3;" ";4;" ";5;" ";6;" ";7
```

In either case the output would be:

```
1 2 3 4 5 6 7
```

Sometimes getting the exact spacing that you want can be tricky.

You can think of the comma (,) and semicolon (;) as two forms of glue. The comma glues two strings or numbers together and places them in neat columns. The semicolon is much stronger since it glues two strings or numbers together and places them right next to each other.

Differences Among Computers. The ATARI, Apple, and Timex do not put blanks around numbers. The Commodore 64 and IBM put a blank before and after each number, so there are two blanks between numbers. For example:

```
10 PRINT 1;2;3;4;5;6;7
```

would print as

```
 1  2  3  4  5  6  7
```

on the IBM and Commodore 64. ■

Instead of just printing the numbers, you could use a loop to get the spaced out output:

```
10 FOR I = 1 TO 7
20 PRINT I; " ";
30 NEXT I
```

To see your screen or printer filled with numbers, you could write a FOR command to go up to 100 or 200 or even more. If the numbers fill up a line, the computer just begins a new line and keeps right on going.

Counting from Here to There

You've used the FOR command to do something seven times by counting from 1 to 7. You can also use the FOR command to count from any number up to another bigger number, for example, from 11 up to 20. If you write this program

```
10 FOR I = 11 TO 20
20 PRINT I; "  ";
30 NEXT I
```

you'll get

```
11  12  13  14  15  16  17  18  19  20
```

You could also print all the years from the time you were born, let's say 1975 to 1984.

```
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
```

The program for this would be

```
10 FOR Y = 1975 TO 1984
20 PRINT Y
30 NEXT Y
```

The Y counter variable stands for *year*.

If you were born in 1971 or even earlier, you could print out all the years and remember the good years as they appeared in the output. You could also look ahead to the years from 1985 to 2001 or beyond.

There is a limit to the size of a number that your computer can handle. If you are going to be using very large numbers, read the computer instruction manual and find out the largest number that is allowed on your computer.

Inner and Outer Counts

When you have one loop inside another, it may be helpful to see the counter variables. You could print the numbers by themselves or with a note about which counter variable is being printed:

```
10 FOR I = 1 TO 3
20 PRINT "HERE IS I "; I
30 FOR J = 1 TO 4
40 PRINT "THERE GOES J "; J
50 NEXT J
60 NEXT I
```

When you ran this program, you could see the pattern

```
HERE IS I 1
THERE GOES J 1
THERE GOES J 2
THERE GOES J 3
THERE GOES J 4
HERE IS I 2
THERE GOES J 1
THERE GOES J 2
THERE GOES J 3
THERE GOES J 4
```

```
HERE IS I 3
THERE GOES J 1
THERE GOES J 2
THERE GOES J 3
THERE GOES J 4
```

Have a Seat

Imagine that you are in a theater where the rows are numbered from 1 to 4 and the seats are numbered from 101 to 105. There are four rows of five seats. You could get an idea of the theater by printing out the floor plan:

```
ROW 1   SEAT 101  102  103  104  105
ROW 2   SEAT 101  102  103  104  105
ROW 3   SEAT 101  102  103  104  105
ROW 4   SEAT 101  102  103  104  105
```

The program for this has to be planned carefully to get the spacing just right. You need a plain PRINT command in line 60 to make each theater row begin on a new line of the output:

```
10 FOR R = 1 TO 4
20 PRINT "ROW "; R; "   SEAT";
30 FOR S = 101 TO 105
40 PRINT S; " ";
50 NEXT S
60 PRINT
70 NEXT R
```

Enjoy the performance!

Summary

1. Counting is done easily with the FOR command. The counter variable starts at the lowest value and goes up to the highest value.

2. You can see how this happens by printing the value of the counter variable.
3. Numbers and strings can be printed in columns, using the comma, or close together using the semicolon.
4. Counting can start at any number and go up to any higher number.

Exercises

1. (+) Write a program to print

```
1          POTATO
2          POTATO
3          POTATO
4          POTATO
5          POTATO
6          POTATO
7          POTATO
MORE
```

2. Write a program to print

```
1 LITTLE INDIAN
2 LITTLE INDIAN
3 LITTLE INDIAN
4 LITTLE INDIAN
5 LITTLE INDIAN
6 LITTLE INDIAN
7 LITTLE INDIAN
8 LITTLE INDIAN
9 LITTLE INDIAN
10 LITTLE INDIAN
BOYS AND GIRLS
```

3. (+) Write a program to print

```
I WAS SO HUNGRY THAT
I ATE 1 ICE CREAM CONES
I ATE 2 ICE CREAM CONES
I ATE 3 ICE CREAM CONES
I ATE 4 ICE CREAM CONES
I ATE 5 ICE CREAM CONES
I ATE TOO MANY ICE CREAM CONES
```

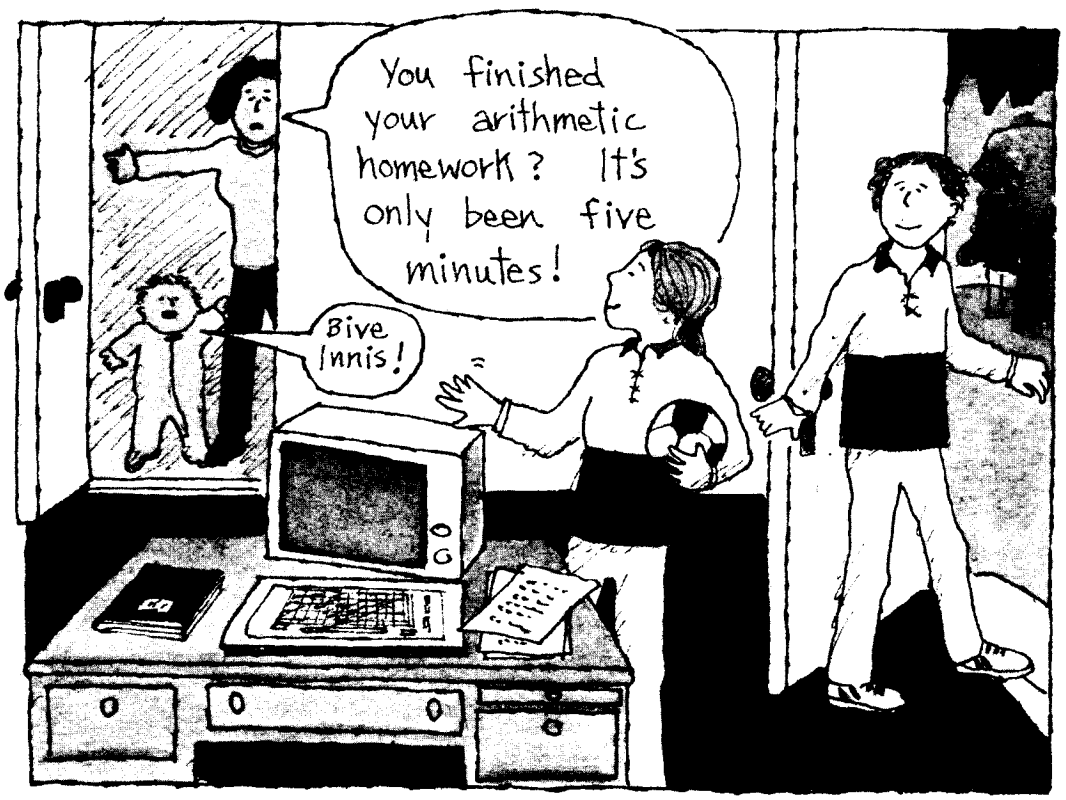
It would take a bit of clever programming to make the second line grammatically correct (`CONE` instead of `CONES`). If you want an extra challenge, try to find a way.

4. (*) Write a program to print this schedule for five weeks of summer camp:

```
WEEK 1  DAY 1  2  3  4  5  6  7
WEEK 2  DAY 1  2  3  4  5  6  7
WEEK 3  DAY 1  2  3  4  5  6  7
WEEK 4  DAY 1  2  3  4  5  6  7
WEEK 5  DAY 1  2  3  4  5  6  7
```

Use an inner loop to print the days from 1 to 7. Use an outer loop to print the weeks from 1 to 5.

7 Simple Arithmetic



▶ **PREVIEW**: Computers are very useful in adding (+), subtracting (–), multiplying (*), and dividing (/). In this chapter you will learn to program simple calculations and to print the results.

▶ **NEW IDEAS**: arithmetic with +, –, *, and /

Calculating and Printing

Computers were first built in the 1940s to do complicated arithmetic. Modern computers can perform millions of additions or multiplications in a second! You can begin to use this power with a very simple calculation:

```
10 PRINT 2+3
```

This one line program will add two and three and display the result when you type RUN:

```
5
```

You can get several calculations done at once:

```
10 PRINT 2+3, 2+7, 6–2, 9–6
```

and get the results when you RUN the program:

```
5          9          4          3
```

The plus sign (+) means addition and the minus sign (–) means subtraction. Multiplication is done with the asterisk (*) and division with the slash (/).

```
10 PRINT 3*4, 8/2
```

produces the output

```
12         4
```

If you use a semicolon, your answers will print closer together. You can print the three's multiplication table across the line by using a semicolon and a blank between each calculation:

```
10 PRINT 3*1; "  "; 3*2; "  "; 3*3; "  "; 3*4;
20 PRINT "  "; 3*5; "  "; 3*6; "  "; 3*7
```

This prints

```
3  6  9  12  15  18  21
```

Can you get the computer to print the four's multiplication table?

```
4  8  12  16  20  24  28
```

You might want to print your three's multiplication table in a column. You can get this by having one calculation in each PRINT command:

```
10 PRINT 3*1
20 PRINT 3*2
30 PRINT 3*3
40 PRINT 3*4
50 PRINT 3*5
60 PRINT 3*6
70 PRINT 3*7
```

When you run this program, it will print

```
3
6
9
12
15
18
21
```

This program is fine, but it uses one command for every line of printed output.

Repeating Calculations

If you wanted to multiply 3 by every number from 1 up to 20, you'd get tired of typing all of the commands. To multiply up to 20 requires 20 PRINT commands! How did you repeat commands before? Did you think of a FOR–NEXT loop? You can write a FOR–NEXT loop that changes the counter variable I from 1 up to 7:

```
10 FOR I = 1 TO 7
20 PRINT 3*I
30 NEXT I
```

The same seven lines of output are printed with less work for you!

Two Column Table

What if someone asked you to use the three's multiplication table to find out what 3 times 5 was? You'd have to count down to the fifth line to find the answer—15. It would be easier if the number 5 was printed next to the number 15. You can do just that by having your computer print a three's multiplication table with two columns:

```
1          3
2          6
3          9
4          12
5          15
6          18
7          21
```

The left column is exactly the values that the counter variable `I` takes, so why not just print `I`:

```
10 FOR I = 1 TO 7
20 PRINT I, 3*I
30 NEXT I
```

A More Meaningful Printout

The multiplication table would be even nicer if it appeared as

```
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
```

To get this requires a few more changes to the PRINT command. You need to PRINT the number 3, then next to it an asterisk, then next to that the value of the counter variable I. All you need now is to print a string with an equals sign and finally the answer:

```
10 FOR I = 1 TO 7
20 PRINT 3; " * "; I; " = "; 3*I
30 NEXT  I
```

Getting tables of calculations should now be "a piece of cake." Speaking of cake, you could put the table-making program idea to good use for Warren. He needs to print out the number of ounces of flour necessary in a recipe that gives instructions by cups. Remember each cup has 8 ounces. To produce this table with a heading

```
CUPS        OUNCES

1           8
2           16
3           24
4           32
5           40
6           48
7           56
8           64
```

you could write this program:

```
10 PRINT "CUPS", "OUNCES"
20 PRINT
30 FOR I = 1 TO 8
40 PRINT I, 8*I
50 NEXT I
```

Now, why not add some chocolate frosting. Yum yum!

Calculations with INPUT Values

Instead of printing a whole table of values every time, you might just want the computer to do a calculation when you need it. The program could ask the user for the number of cups and then print out the number of ounces, like this:

```
THIS PROGRAM WILL CALCULATE THE
NUMBER OF OUNCES IF YOU GIVE THE
NUMBER OF CUPS.
HOW MANY CUPS DO YOU HAVE ?6
THAT MAKES 48 OUNCES
```

The program would begin by giving the instructions:

```
10 PRINT "THIS PROGRAM WILL CALCULATE THE"
20 PRINT "NUMBER OF OUNCES IF YOU GIVE THE"
30 PRINT "NUMBER OF CUPS."
```

Now the program asks for the INPUT:

```
40 PRINT "HOW MANY CUPS DO YOU HAVE ";
50 INPUT C
```

There is a semicolon at the end of the string in line 40. This means that the question mark printed out by the INPUT command will appear on the same line. The INPUT command asks for the value of the variable C, which represents cups.

The calculation and the printing complete the program:

```
60 PRINT "THAT MAKES"; 8*C; "OUNCES"
```

Each time you need to compute ounces given the number of cups, you could RUN this program. Since you may want to make several calculations at once, you could wrap a loop around the whole program. You could have the instructions repeated 10 times:

```
5 FOR I = 1 TO 10
10 PRINT "THIS PROGRAM WILL CALCULATE THE"
20 PRINT "NUMBER OF OUNCES IF YOU GIVE THE"
30 PRINT "NUMBER OF CUPS."
40 PRINT "HOW MANY CUPS DO YOU HAVE ";
50 INPUT C
60 PRINT "THAT MAKES"; 8*C; "OUNCES"
70 NEXT I
```

You could write this program so that the instructions appear only once. Instead of putting the FOR command on line 5, you might put it at line 35. This program stops when you have done exactly 10 calculations. If you want to stop at eight, you press the BREAK key, when the computer is waiting for INPUT.

A typical run might be:

```
THIS PROGRAM WILL CALCULATE THE
NUMBER OF OUNCES IF YOU GIVE THE
NUMBER OF CUPS.
HOW MANY CUPS DO YOU HAVE ?5
THAT MAKES 40 OUNCES
HOW MANY CUPS DO YOU HAVE ?12
THAT MAKES 96 OUNCES
HOW MANY CUPS DO YOU HAVE ?261
THAT MAKES 2088 OUNCES
HOW MANY CUPS DO YOU HAVE ?7
THAT MAKES 56 OUNCES
HOW MANY CUPS DO YOU HAVE ?
   (Press BREAK)
```

and then the program would stop. You can now RUN the program again, do a LIST, or other commands.

Summary

1. Simple arithmetic calculations can be done by using the plus sign (+), minus sign (–), multiplication (*), and division (/).
2. The result of the calculations can be printed across the line in columns, across the line closely together, or down the page.
3. The user of a program can supply a number for a calculation whenever you put an INPUT command in your program.

Exercises

1. Test out the arithmetic calculations by finding 2 plus 2, 2 minus 2, 2 times 2, and 2 divided by 2.
2. (+) Did you ever wonder how many hours there are in a week? Compute the number of hours in a week by multiplying the number of days in a week by the number of hours in a day.

3. Find out how many inches in a mile by multiplying the number of inches in a foot (12) by the number of feet in a mile (5280).
4. (+) Karen was making a ghost costume for Halloween. She figured out that she needed 54 inches of cloth, but the store sells cloth by the foot. Make a table of feet and inches like this:

1	12
2	24
3	36
4	48
5	60
6	72

Now Karen can figure out how many feet of cloth to buy.

5. Expand the feet and inches table to be yards, feet, and inches. There are 3 feet per yard and 36 inches per yard.

YARDS	FEET	INCHES
1	3	36
2	6	72
3	9	108
4	12	144
5	15	180
6	18	216

6. (+) Write a program to ask the user for the input of a number of feet and print out the number of inches. The run might look like this

```
THIS PROGRAM WILL CALCULATE THE
NUMBER OF INCHES IF YOU GIVE THE
NUMBER OF FEET.
HOW MANY FEET DO YOU HAVE ?3
THERE ARE 36 INCHES
```

7. (*) Write a program to ask users for the year in which they were born (call it B) and for this year (call it T). Then print out a table that shows their age in each year. For example:

```
WHICH YEAR WERE YOU BORN IN ?1974
WHAT YEAR IS IT NOW ?1984

YEAR        YOUR AGE

1974        0
1975        1
1976        2
1977        3
1978        4
1979        5
1980        6
1981        7
1982        8
1983        9
1984        10
```

You'll need a loop from B to T and then a subtraction to get the age.

8 Arithmetic Variables

▶ **PREVIEW**: Many arithmetic problems require only a simple calculation and printout of the result. Sometimes you need to save the result of one calculation and use the value in another calculation. This chapter shows how to use arithmetic variables to store the results of a calculation.

▶ **NEW IDEAS**: arithmetic variables, LET, storing with =, printing arithmetic variables

Storing Values in a Variable

David and Lisa set up a lemonade stand. They are charging 15 cents for a large cup and 10 cents for a small cup. Their first customers are the family next door. The parents each have a large cup, and the four children each have a small cup. David and Lisa need to know how much to charge for 2 large cups and 4 small cups of lemonade. The total charge is 2 times 15 plus 4 times 10. You could probably do this in your head, but you might want to try programming this simple problem.

First, you multiply 2 cups times 15 cents for the large cups. You need to store the result in an *arithmetic variable* called L for large. The LET command sets an arithmetic variable to a given value. For example,

```
10 LET L = 2*15
```

does the multiplication of 2 times 15. The value—30—is stored in the arithmetic variable L. The value of L is now 30. The next step is to use another LET command to multiply the 4 small cups times 10 cents and store the result. The value—40—is stored in another arithmetic variable S for small. Now, you can add L and S to get the total. You store this result in an arithmetic variable called T for total. Finally, you print the total:

```
10 LET L = 2*15
20 LET S = 4*10
30 LET T = L+S
40 PRINT "TOTAL COST IS "; T
```

Running this program produces one line of output:

```
TOTAL COST IS 70
```

Line 10 gets the computer to multiply 2 times 15 and store the result—30—in the arithmetic variable L. Line 20 multiplies 4 times 10 and stores the result —40—in the arithmetic variable S. Line 30

adds the values in L and S and stores the answer in the arithmetic variable T. The semicolon in line 40 makes the value print out close to the string.

You could print the results of each step by adding some other PRINT commands:

```
10 LET L = 2*15
15 PRINT "COST FOR LARGE CUPS IS "; L
20 LET S = 4*10
25 PRINT "COST FOR SMALL CUPS IS "; S
30 LET T = L+S
40 PRINT "TOTAL COST IS "; T
```

and rerunning the program:

```
COST FOR LARGE CUPS IS 30
COST FOR SMALL CUPS IS 40
TOTAL COST IS 70
```

LET commands are used to store a value in a variable.

LET commands begin with a line number and the word LET. Then comes a variable followed by an equal sign. To the right of the equal sign there can be a calculation using numbers and variables.

The first step for the computer is to carry out the arithmetic on the right side of the equal sign. Then the second step is to take this value and store it in the variable. A simple LET command is

```
10 LET X = 1
```

which would set the variable X to 1. Another LET command is

```
20 LET P = Q
```

which would set the variable P to the same value as Q. Many times LET commands have arithmetic calculations such as

```
30 LET M = 4*K
```

which multiplies K by 4 and stores the result in M. If K was 6, then after the LET command was done, M's value would be 24. K would still be 6.

Differences Among Computers. With the ATARI and most computers you can leave out the LET part of the LET command. For example:

```
10 X = 1
20 P = Q
30 M = 4*K
```

Some computers, such as the Timex, require the word LET. ■

Lawn Service

Two friends started a lawn mowing and gardening service which had this poem as their advertisement:

> Jack and Jill went up the hill
> To fetch a pail of water.
> They wet the flowers, mowed the lawns,
> And did just what they oughta.

Jack and Jill wrote a computer program to help them add up charges for watering plants, mowing lawns, raking leaves, and trimming hedges. Their program asked for the amount of money for each service and then added up the total:

```
10 PRINT "TYPE IN THE DOLLAR AND CENTS AMOUNTS"
20 PRINT "  FOR EACH SERVICE AND PRESS RETURN"
30 PRINT
40 PRINT "WATERING PLANTS";
50 INPUT W
60 PRINT "MOWING LAWNS   ";
70 INPUT M
80 PRINT "RAKING LEAVES  ";
```

```
90 INPUT R
100 PRINT "TRIMMING HEDGES";
110 INPUT T
120 LET C = W + M + R + T
130 PRINT
140 PRINT "TOTAL COST IS $ "; C
```

Line 120 adds up the four separate charges so that the total cost can be printed in line 140. A typical run might be

```
TYPE IN THE DOLLAR AND CENTS AMOUNTS
  FOR EACH SERVICE AND PRESS RETURN

WATERING PLANTS?1.75
MOWING LAWNS   ?4.50
RAKING LEAVES  ?0.0
TRIMMING HEDGES?0.0

TOTAL COST IS $ 6.25
```

Can you see how to expand the program to include other services such as pulling weeds or to offer a $1.00 discount for advance payment?

Fahrenheit and Celsius Temperatures

The Fahrenheit temperature scale is used in the United States, but most of the countries in the world use the Celsius temperature scale. The freezing temperature of water is 32 degrees Fahrenheit and 0 degrees Celsius. The boiling temperature of water is 212 degrees Fahrenheit and 100 degrees Celsius.

Changing from Fahrenheit to Celsius requires three calculations:

1. Start with the Fahrenheit temperature and subtract 32
2. Multiply by 5
3. Divide by 9 to get the Celsius temperature

The program to change Fahrenheit temperatures to Celsius might start by asking the user to give a Fahrenheit temperature. The three calculations are done, and finally, the result is printed:

```
10 PRINT "TO CHANGE FAHRENHEIT TO CELSIUS"
20 PRINT "TYPE THE DEGREES AND PRESS RETURN"
30 PRINT "FAHRENHEIT DEGREES ";
40 INPUT F
50 LET G = F - 32
60 LET H = G * 5
70 LET C = H / 9
80 PRINT "CELSIUS DEGREES "; C
```

The run might look like this:

```
TO CHANGE FAHRENHEIT TO CELSIUS
TYPE THE DEGREES AND PRESS RETURN
FAHRENHEIT DEGREES ?212
CELSIUS DEGREES 100
```

If the program had more print commands you could show the subtraction of 32 to give 180, the multiplication by 5 to give 900, and the division by 9 to give 100. In this program the arithmetic variables G and H were used just to save the results of steps 1 and 2 of the calculation. These values do not need to be printed out.

Summary

1. The results of arithmetic calculations can be stored in arithmetic variables.
2. The LET command stores a value in a variable. It has a line number, the word LET, an arithmetic variable, an equal sign, and the calculation.
3. Complicated calculations can be done one step at a time.

Exercises

1. Courtney has 2 quarters and 3 dimes. Write a program to find the total amount of money by multiplying 2 times 25 and storing the result. Then multiply 3 times 10 and store the result in another arithmetic variable. Finally, add the two values and print the result.
2. (+) Have you ever wondered how many seconds there are in a day? There are 60 seconds in a minute, 60 minutes in an hour, and 24 hours in a day. First, calculate and print the number of seconds in an hour. Then calculate and print the number of seconds in a day. Be sure to print a message to describe each number.
3. Michael and Matthew have started a car-washing business. They call themselves M and M, and give free candy to their customers. They offer car washing, waxing, and vacuuming as their services. Write a program that asks for the charge for each of these three services and prints out the total cost.
4. Richard has a newspaper route and must figure out the monthly charge for his customers. A weekday newspaper costs 25 cents, and the Sunday newspaper costs 75 cents. Write a program that asks for the total number of weekday papers delivered and the total number of Sunday newspapers delivered. Then calculate and print the total amount in cents:

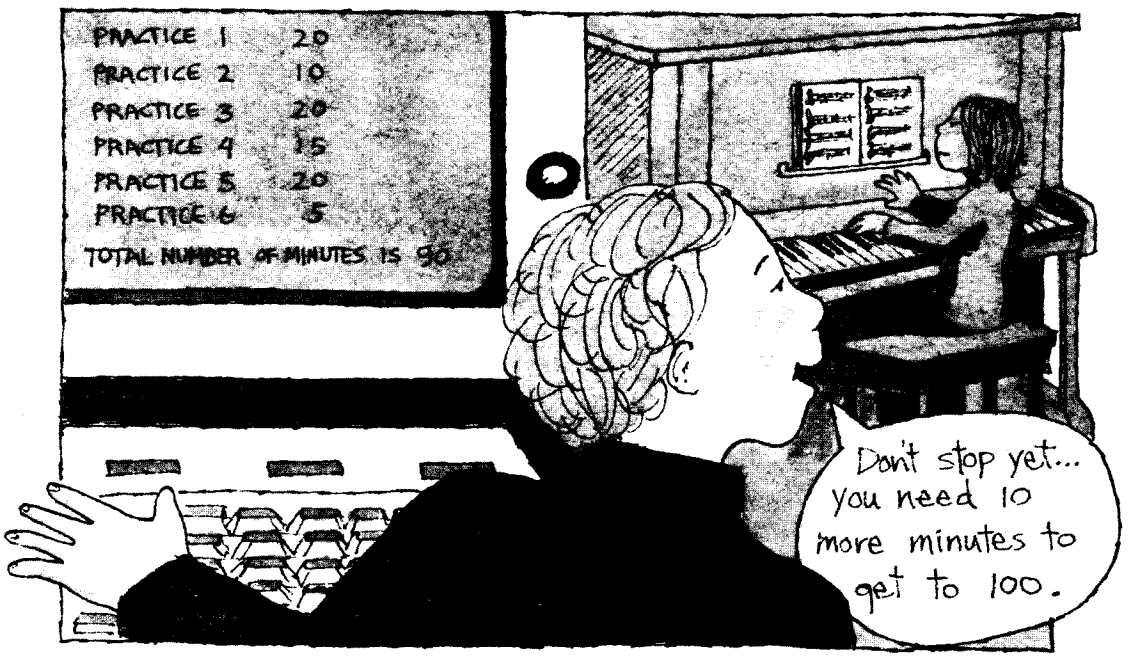
```
TYPE THE NUMBER AND PRESS RETURN
HOW MANY WEEKDAY PAPERS THIS MONTH ?26
HOW MANY SUNDAY PAPERS THIS MONTH ?4
TOTAL COST IN CENTS IS  950
```

5. (*) Changing from Celsius to Fahrenheit has three steps:
   a. Multiply the Celsius temperature by 9
   b. Divide this amount by 5
   c. Add 32

   Write a program that asks for a Celsius temperature, does the calculation, and prints out the Fahrenheit temperature.
6. (*+) Katie collects stamps from four countries: France, England, Italy, and Germany. She likes to keep track of how many stamps she has from each country. Write a program that helps her keep track. It might run like this:

```
TYPE IN THE NUMBER OF STAMPS
  FROM EACH COUNTRY AND PRESS RETURN
FRANCE ?23
ENGLAND ?49
ITALY ?13
GERMANY ?21
TOTAL NUMBER OF STAMPS IS 106
```

9 Adding Up Numbers



▶ **PREVIEW**: Sometimes people use computers to add up a list of numbers. The list may be prices on a supermarket bill or the number of minutes you spent practicing piano each week. In these cases an arithmetic variable can be used for adding up the total.

▶ **NEW IDEAS**: adding up a list of numbers, setting an arithmetic variable to zero

How Much Cheese?

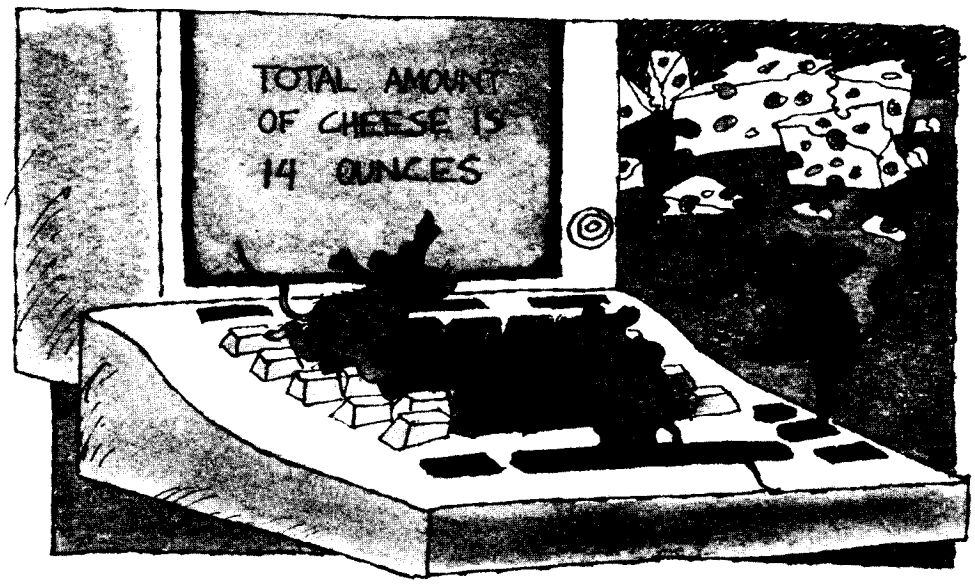
Three mice are saving up cheese and want to know how much they have, so they write a computer program. The program uses an arithmetic variable called T for total, which is set to zero. Then when the amounts of cheese in ounces—C—are typed in, the total variable is increased. Finally, the result is printed:

```
10 PRINT "TYPE IN THE NUMBER OF OUNCES"
20 PRINT "  FOR EACH OF THE THREE MICE"
30 PRINT "  AND PRESS RETURN EACH TIME"
40 LET T = 0
50 FOR M = 1 TO 3
60 INPUT C
70 LET T = T + C
80 NEXT M
90 PRINT "TOTAL AMOUNT OF CHEESE IS "; T;
100 PRINT " OUNCES"
```

The counter variable M goes from mouse number 1 up to mouse number 3. The input variable C takes the number of ounces of cheese for each mouse. Line 70 causes the current total amount—T—to be added to the amount of cheese—C—and the result is stored back in T. If the mice had 3, 7, and 4 ounces each, the run might look like:

```
TYPE IN THE NUMBER OF OUNCES
  FOR EACH OF THE THREE MICE
  AND PRESS RETURN EACH TIME
?3
?7
?4
TOTAL AMOUNT OF CHEESE IS 14 OUNCES
```

The total variable T was first set to 0, then 3 was added. The second time through the FOR–NEXT loop, 7 was added to T to make it 10. The third time through the loop, 4 was added to 10 to make T have the value 14. Then the final value was printed by line 90.



Remember that when a question mark has been displayed, you can stop the program by pressing the BREAK key.

Adding up numbers is one of the first things programmers learn to do. Adding up starts by setting a total variable to zero. Then each time a value has to be added, you take the current value of the total variable and add another amount to it. This sum is stored back into the total variable.

Piano Practice

Claire's piano teacher requires 100 minutes of practice every week, but Claire can spend as long as she wants in each practice period. To make sure that she gets all her practicing done, she gets the computer to total up the number of minutes in each practice period. The program asks for the number of practice periods and then asks for the number of minutes in each practice period. When all the numbers have been typed in, the program prints the total number of minutes:

```
10 PRINT "TYPE THE NUMBER OF PRACTICES"
20 PRINT "  AND PRESS RETURN"
30 INPUT S
40 PRINT "FOR EACH PRACTICE, TYPE THE NUMBER"
50 PRINT "  OF MINUTES AND PRESS RETURN"
60 LET T = 0
70 FOR I = 1 TO S
80 PRINT "PRACTICE "; I;
90 INPUT M
100 LET T = T + M
110 NEXT I
120 PRINT "TOTAL NUMBER OF MINUTES IS "; T
```

The program might run like this:

```
TYPE THE NUMBER OF PRACTICES
  AND PRESS RETURN
?6
FOR EACH PRACTICE, TYPE THE NUMBER
  OF MINUTES AND PRESS RETURN
PRACTICE 1 ?12
PRACTICE 2 ?31
PRACTICE 3 ?5
PRACTICE 4 ?15
PRACTICE 5 ?25
PRACTICE 6 ?22
TOTAL NUMBER OF MINUTES IS 108
```

As you can see, Claire did more than enough piano practice.

Summary

1. Adding up lists of numbers can be done by using an arithmetic variable to total up the numbers one at a time.
2. A LET command is used to set the total variable to zero.
3. Another LET command is used to add in each number.

4. A FOR–NEXT loop controls the number of items added in to get the total.

Exercises

1. What is the output of this program?

```
10 LET T = 0
20 FOR I = 1 TO 5
30 LET T = T + I
40 NEXT I
50 PRINT T
```

   Run it and see if your guess was right.

2. (+) Every weeknight Jenna gets a math worksheet with 100 problems on it. She has 8 minutes to do as many as she can. Write a program that will help Jenna add up the number of problems she completes in a five-day week.

3. Billy's ski team has four racers. They each ski down the trail as fast as they can. The team score is the sum of the time in seconds for all four racers. Write a program to ask for and add up time for each racer.

```
TYPE THE NUMBER OF SECONDS
  AND PRESS RETURN
RACER 1 ?79
RACER 2 ?86
RACER 3 ?77
RACER 4 ?80
TOTAL TEAM TIME IS 322 SECONDS
```

4. Becky likes to check the bill she gets when she goes shopping. Write a program that asks how many shopping items were purchased. Then set the total cost variable to zero. Now loop from 1 to the number of items, printing a

message to request the price, and finally print the total:

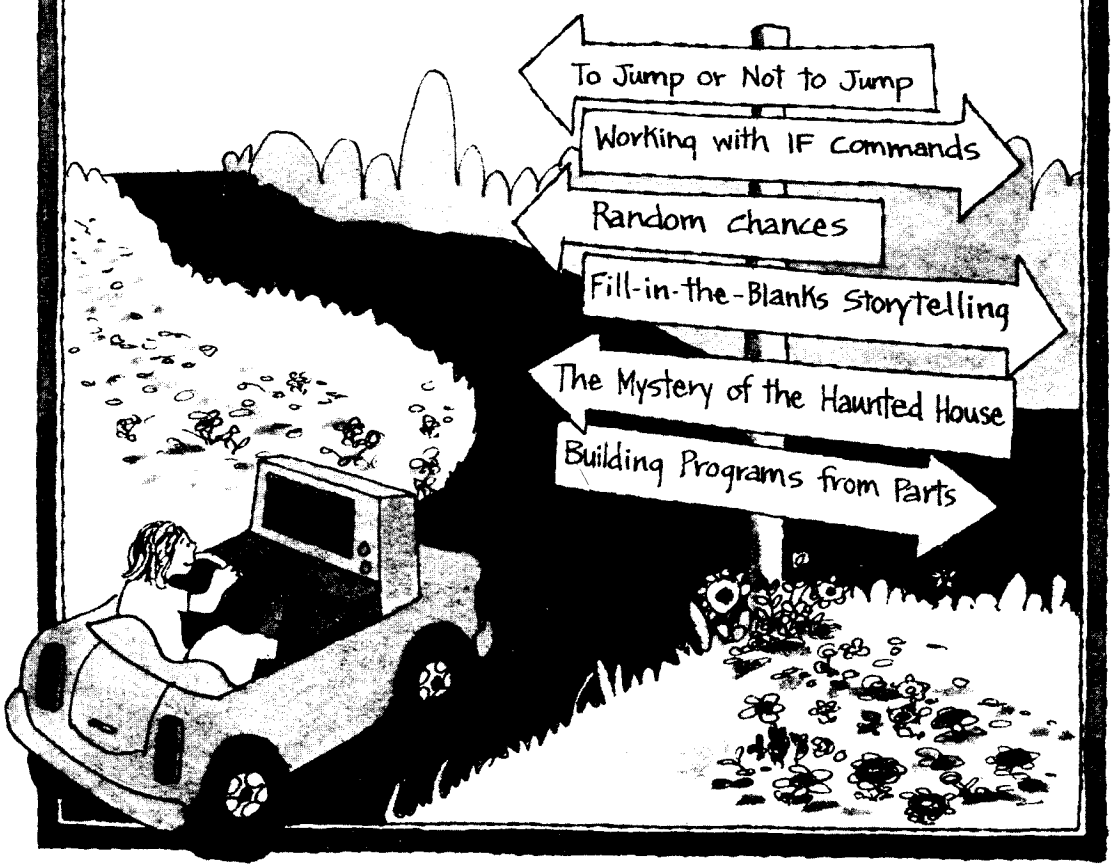
```
TYPE THE NUMBER OF ITEMS YOU BOUGHT
  AND PRESS RETURN
?4
ITEM  1 PRICE WAS ?1.25
ITEM  2 PRICE WAS ?3.87
ITEM  3 PRICE WAS ?.63
ITEM  4 PRICE WAS ?12.00
TOTAL COST WAS $  17.75
```

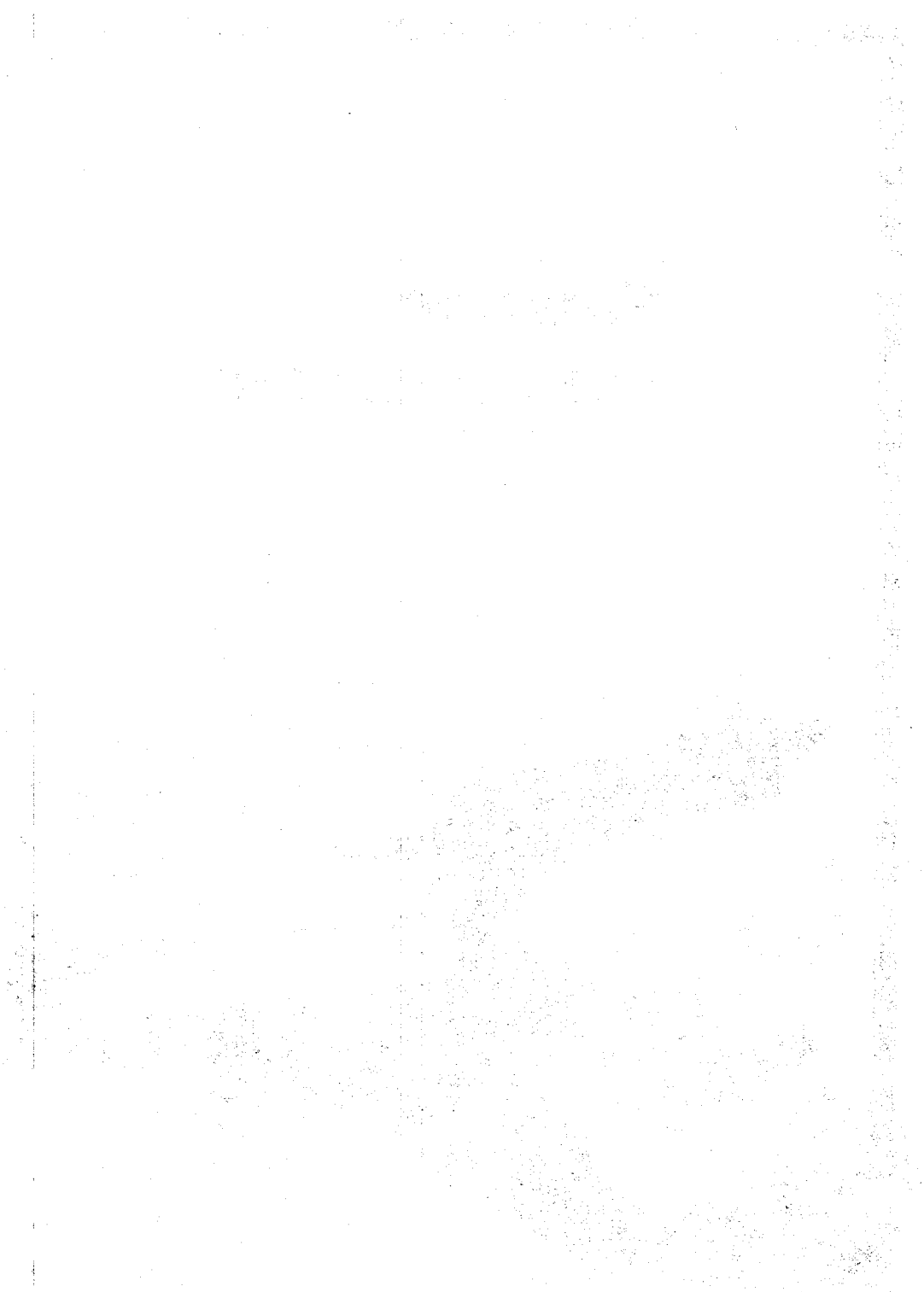
5. (*) In some games, like Scrabble, players take turns and score points. Write a program that will keep score and show the score of each player after each move. Set arithmetic variables to zero for player A and player B. Ask for each score and add it to the score for each player.

```
TYPE IN THE SCORE FOR EACH PLAYER
  AND PRESS RETURN

PLAYER A SCORE IS 0
PLAYER B SCORE IS 0

HOW MANY POINTS DID PLAYER A SCORE ?8
HOW MANY POINTS DID PLAYER B SCORE ?9

PLAYER A SCORE IS 8
PLAYER B SCORE IS 9

HOW MANY POINTS DID PLAYER A SCORE ?11
HOW MANY POINTS DID PLAYER B SCORE ?6

PLAYER A SCORE IS 19
PLAYER B SCORE IS 15

HOW MANY POINTS DID PLAYER A SCORE ?
```

(Press BREAK to stop the program.)

Steering the Computer





10 To Jump or Not to Jump

▶ **PREVIEW**: People often use computers to help make decisions. In your programs you can test if a string in an input variable matches a string in your program. You can also test arithmetic variables such as a temperature to see if it is above or below 32 degrees Fahrenheit and then print `ABOVE FREEZING` or `BELOW FREEZING`. You can compare two numbers and jump over some commands. It's like steering the computer around the commands.

▶ **NEW IDEAS**: decision making, `IF`, `GOTO`, `THEN`, jumping over commands, `END`

Rumpelstiltskin Is My Name

So far you've used a computer to print words and pictures and to do arithmetic. Now you will learn how to make decisions by steering the computer around the commands.

In the story of Rumpelstiltskin a dwarf helps a girl spin straw into gold. He forces her to promise to give her first baby to him. When the baby comes, she doesn't want to give it up. Rumpelstiltskin says she can keep the baby if she can guess his name.

We can write a program that won't stop until someone types in the right name. Line 5 reserves up to 40 characters for the string input variable A$. The program keeps asking for a name and tests if the name is RUMPELSTILTSKIN. If it is, then the program jumps to line 50. If the name does not match, then line 40 sends the program back to line 10 and the question is repeated.

```
5 DIM A$(40)
10 PRINT "CAN YOU GUESS MY NAME ";
20 INPUT A$
30 IF A$ = "RUMPELSTILTSKIN" THEN 50
40 GOTO 10
50 PRINT "YOU GUESSED MY NAME, SO"
60 PRINT "  YOU CAN KEEP YOUR BABY."
```

If you ran this program, it might go like this:

```
CAN YOU GUESS MY NAME ?POLLY
CAN YOU GUESS MY NAME ?MIGUEL
CAN YOU GUESS MY NAME ?MEI LEE
CAN YOU GUESS MY NAME ?RUMPELSTILTSKIN
YOU GUESSED MY NAME, SO
  YOU CAN KEEP YOUR BABY.
```

There are two new commands in this program. The IF command lets you compare a variable to a string to see if it matches. If A$ is RUMPELSTILTSKIN the program will jump to line 50. If there is no match, the program goes right on to line 40. The GOTO 10 command sends the program back to repeat the question. This

program ends if you guess the right name or if you press the BREAK key. IF commands and GOTO commands will be used in most programs that you write.

Do I Need a Warm Coat?

You might want to use a computer to give instructions about what to wear. If the temperature is warmer than 32 degrees Fahrenheit (the freezing temperature of water), you print ABOVE FREEZING. WEAR A JACKET. If the temperature is 32 degrees or colder, print IT'S FREEZING. WEAR A COAT AND SCARF. Your program begins by putting in the temperature

```
10 PRINT "WHAT IS THE TEMPERATURE ";
20 INPUT T
```

which is stored as T, for temperature. Next comes the decision whether the temperature is warmer than 32 degrees Fahrenheit. We use the symbol > to mean greater than.

```
30 IF T > 32 THEN 60
```

The IF command compares T to the number 32. THEN is the second part of the IF command. It means that if T is warmer than 32 (greater than 32), the next command is on line 60.

```
40 PRINT "IT'S FREEZING. WEAR A COAT AND SCARF."
50 GOTO 70
60 PRINT "ABOVE FREEZING.  WEAR A JACKET."
70 PRINT "HAVE A NICE DAY."
```

If T is 32 degrees or colder (less than or equal to 32), the computer does not jump but simply goes on to the next line—line 40—and prints IT'S FREEZING. WEAR A COAT AND SCARF. The computer moves on to line 50 which makes it GOTO line 70. Line 70 is where the program goes after printing either one of the

messages. If the temperature is 88 degrees, the program works like this:

```
WHAT IS THE TEMPERATURE ?88
ABOVE FREEZING.  WEAR A JACKET.
HAVE A NICE DAY.
```

The computer went through lines 10, 20, 30, 60, and 70. The THEN part of line 30 steered the computer around line numbers 40 and 50 by jumping to line 60. If the temperature is 23 degrees, this is what happens:

```
WHAT IS THE TEMPERATURE ?23
IT'S FREEZING.  WEAR A COAT AND SCARF.
HAVE A NICE DAY.
```

In this run the computer went through lines 10, 20, 30, 40, 50, and 70. The GOTO command in line 50 jumped the computer right to line 70. This program always uses two of the three PRINT

commands. Here's the full listing:

```
10 PRINT "WHAT IS THE TEMPERATURE ";
20 INPUT T
30 IF T > 32 THEN 60
40 PRINT "IT'S FREEZING. WEAR A COAT AND SCARF."
50 GOTO 70
60 PRINT "ABOVE FREEZING.  WEAR A JACKET."
70 PRINT "HAVE A NICE DAY."
```

Decisions, Decisions, Decisions

The first section used one form of the IF command: `IF T > 32`. The greater than sign is one of six signs that you can use in BASIC. The equal sign (=) allows you to compare two values to test if they are the same.

```
IF T = 32 THEN 80
```

tests if the temperature is exactly 32 degrees. The full set of six comparisons that you can use are:

`IF T > 32 THEN 80`	greater than
`IF T = 32 THEN 80`	equal to
`IF T < 32 THEN 80`	less than
`IF T >= 32 THEN 80`	greater than or equal to
`IF T <= 32 THEN 80`	less than or equal to
`IF T <> 32 THEN 80`	not equal to (less than or greater than, but not equal to)

The THENs in the IF command show the line number to jump to if the test is true. If the test is not true, then the computer goes on to the next line after the IF command.

Differences Among Computers. On the ATARI Home Computers and most other computers there are a few ways to write IF commands. These commands do the same thing:

```
20 IF T > 32 THEN 80
20 IF T > 32 THEN GOTO 80
20 IF T > 32 GOTO 80
```

On most computers you can put a PRINT, LET, or other command instead of a line number. For example:

```
20 IF T > 32 THEN PRINT "ABOVE FREEZING" ■
```

GOTO commands can be used anywhere in the program by themselves to steer the computer around some commands. For example, this strange program has a secret message in it:

```
10 PRINT "THERE ";
20 GOTO 50
30 PRINT "CE LIK";
40 GOTO 70
50 PRINT "IS NO"
60 GOTO 90
70 PRINT "E HOME"
80 GOTO 110
90 PRINT "PLA";
100 GOTO 30
110 PRINT "SAID DOROTHY"
```

When this program is run, the hidden message will appear. Can you figure it out?

Checking Temperature Ranges

Normal body temperature is 98.6 degrees Fahrenheit. Most doctors agree that one degree above or below is still healthy. To check if a person's temperature was in the normal range of 97.6 to 99.6, you would need two IF commands:

```
10 PRINT "WHAT IS YOUR TEMPERATURE ";
20 INPUT T
```

```
30 IF T < 97.6 THEN 100
40 IF T > 99.6 THEN 100
50 PRINT "NORMAL BODY TEMPERATURE"
60 GOTO 200
100 PRINT "NOT NORMAL BODY TEMPERATURE"
200 END
```

This program tests only if the temperature is normal or is not normal. If the temperature is too low or too high, the program jumps to line 100. If the temperature passes both of the tests in lines 30 and 40, the program goes right on to line 50.

The GOTO in line 60 jumped to line number 200, which is a new command—the END command. When there is nothing more to do in a program, you just END the program. We will make the END command the highest numbered line in a program. It will always be the last command in a program. The END command lets readers of your program know that the program is through. We will use END commands for most programs, especially the longer ones.

The body-temperature program could be improved to print out whether the temperature was above or below normal. Instead of having both IF commands jumping to line 100, the second IF command could jump to line number 150. Then you could have a below normal message at line 100 and an above normal message at line 150. The program would still end on line 200.

```
10 PRINT "WHAT IS YOUR TEMPERATURE ";
20 INPUT T
30 IF T < 97.6 THEN 100
40 IF T > 99.6 THEN 150
50 PRINT "NORMAL BODY TEMPERATURE"
60 GOTO 200
100 PRINT "BELOW NORMAL BODY TEMPERATURE"
110 GOTO 200
150 PRINT "ABOVE NORMAL BODY TEMPERATURE"
200 END
```

Cool Pool

The Little Falls Pool charges 40 cents for swimmers under 16 years and 90 cents for those 16 or older. After 6:00 P.M. the price is cut in half for everyone. This program might be useful to the ticket taker in figuring out how much to charge.

```
HOW OLD ?12
IS IT BEFORE 6:00 PM ?YES
PRICE IS 40 CENTS
```

This program has two IF commands and two INPUT commands:

```
10 PRINT "HOW OLD ";
20 INPUT A
30 IF A >= 16 THEN 60
40 LET P = 40
50 GOTO 70
60 LET P = 90
70 PRINT "IS IT BEFORE 6:00 PM ";
80 INPUT A$
90 IF A$ = "YES" THEN 110
100 LET P = P/2
110 PRINT "PRICE IS"; P; "CENTS"
```

The arithmetic variable A holds the age. The variable A$ holds the input string YES or NO. A and A$ are different variables.

Some programmers use a clever shortcut to replace lines 30, 40, 50, and 60 with just two lines

```
30 LET P = 40
40 IF A >= 16 THEN LET P = 90
```

In this program, the price is set to 40 cents. If the age is greater than 16, the price is set to 90 cents.

IF commands and GOTOs can be put together to form complicated programs. You should practice with simple programs first. Then you can move on to programs which have many IFs and GOTOs.

To understand more complicated programs, you may want to print out the variables after some commands. For example, you may want to add PRINT commands for A, P, and A$ after lines 20, 40, 60, and 80. This information can be very helpful if your program does not work correctly the first few runs. Sometimes it takes many runs and changes until you get your program right. Think carefully and keep trying.

Summary

1. IF commands are a very important part of programming. A single IF command lets you test if a number is above, below, or equal to another number.
2. Two IF commands can be put together to test if a number is within a range of numbers.
3. The GOTO command lets you jump to a line number.
4. We will make END the last command in a program. It is used when you need to GOTO a line that ends the running of a program.

Exercises

1. Write a program that keeps asking for the secret password until the user guesses it. It might run like this:

```
TYPE THE SECRET PASSWORD ?COOKIE
TYPE THE SECRET PASSWORD ?MONSTER
TYPE THE SECRET PASSWORD ?HEFFALUMP
TYPE THE SECRET PASSWORD ?SESAME
YOU GUESSED IT...WELCOME TO THE CLUB
```

2. (+)To qualify for the neighborhood swim team, the Dolphins, you must swim at least 16 laps of the pool. Write a program to let the user type in the number of laps with an INPUT

command. Then test if the number is greater than or equal to 16 and print the message QUALIFIES or DOES NOT QUALIFY.

3. John keeps the school records for best times in sports events. The school record for running around the track is 94 seconds. Write a program to INPUT a new time in seconds. Then test the time and print either CONGRATULATIONS, YOU SET A NEW RECORD or GOOD WORK, BUT THE OLD RECORD STANDS.
4. On some highways there is a minimum speed limit of 40 miles per hour and a maximum speed limit of 55 miles per hour. Write a program to INPUT a speed and test if the speed is legal. Print the message LEGAL SPEED or ILLEGAL SPEED.
5. (+) Change the program in Exercise 4 so that it prints one of these three messages:

```
LEGAL SPEED
ILLEGAL SPEED - TOO SLOW
ILLEGAL SPEED - TOO FAST
```

6. (*) The third-grade teacher, Ms. Glantz, rewarded her best students by allowing them to spend an extra hour in the library reading books of their choice if the sum of their three weekly tests (0 to 10 points on each test) was at least 25. Write a program to take in the three test scores with three INPUT commands, then add up the three values, print the sum, and after testing the sum, print one of these two messages: YOU MAY GO TO THE LIBRARY or YOU MUST REDO ALL THE WRONG ANSWERS ON YOUR TESTS.

11 Working with IF Commands

- **PREVIEW**: You will find that most programs use IF commands. Some programs have dozens of them. This chapter uses IF commands to build a quiz program in which the user must choose the correct answer. As your programs get longer, you will want to add comments to explain how they work. REMarks let you tell the reader something about your program. The second program, which is about simple banking, uses IF commands to check values and to steer the program from one step to another.
- **NEW IDEAS**: quiz questions, counting answers, REMarks, banking

Quiz Time

Multiple-choice quizzes can be fun, and you can learn from them, too. In this first example the program prints out a question and four possible answers. Then the user makes a choice and finds out if it is correct. It goes like this,

```
WHO INVENTED THE PHONOGRAPH
1  JAMES WATT
2  BENJAMIN FRANKLIN
3  SAMUEL MORSE
4  THOMAS EDISON
TYPE THE NUMBER AND PRESS RETURN ?4
YOU ARE CORRECT
```

If any other number were chosen, the computer would print

```
THE INVENTOR OF THE PHONOGRAPH WAS
THOMAS EDISON
```

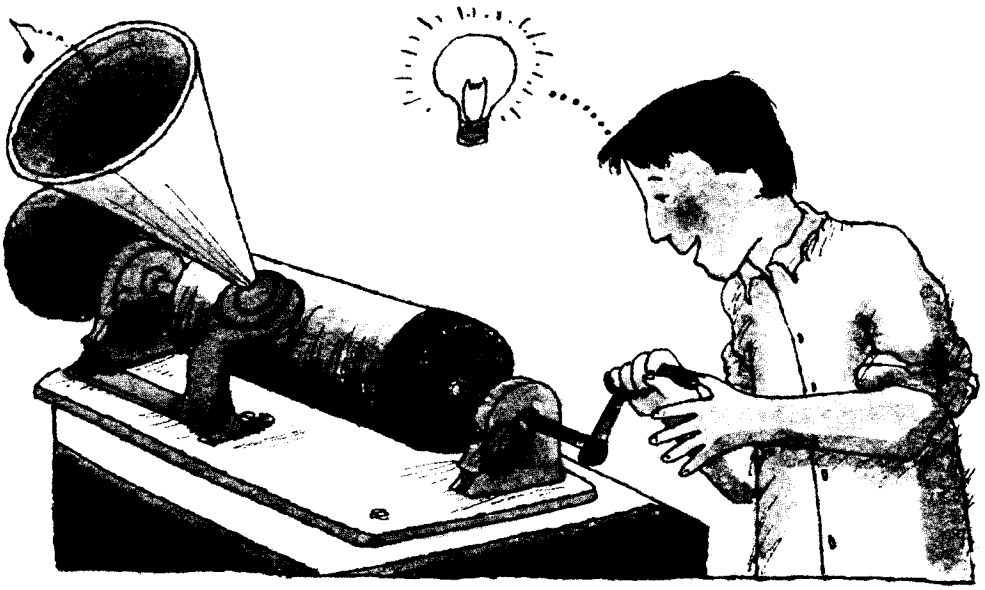
To make this program, you must first print out the question:

```
10 PRINT "WHO INVENTED THE PHONOGRAPH"
20 PRINT "1  JAMES WATT"
30 PRINT "2  BENJAMIN FRANKLIN"
40 PRINT "3  SAMUEL MORSE"
50 PRINT "4  THOMAS EDISON"
60 PRINT "TYPE THE NUMBER AND PRESS RETURN ";
```

That's the easy part. Next, your program must take the answer and test to see if the right answer was chosen:

```
70 INPUT A
80 IF A = 4 THEN 150
90 PRINT "THE INVENTOR OF THE PHONOGRAPH WAS"
100 PRINT "THOMAS EDISON"
110 GOTO 200
150 PRINT "YOU ARE CORRECT"
```

Your program could now go on to another question:



```
200 PRINT "WHO INVENTED THE TELEPHONE"
210 PRINT "1  THOMAS EDISON"
220 PRINT "2  ALEXANDER GRAHAM BELL"
230 PRINT "3  LEE DE FOREST"
240 PRINT "4  GEORGE WESTINGHOUSE"
250 PRINT "TYPE THE NUMBER AND PRESS RETURN ";
260 INPUT A
270 IF A = 2 THEN 310
280 PRINT "THE INVENTOR OF THE TELEPHONE WAS"
290 PRINT "ALEXANDER GRAHAM BELL"
300 GOTO 350
310 PRINT "YOU ARE CORRECT"
350
```

At line 350 you could go on to a third question. By putting together a series of questions like this, you could help teach yourself and your friends about inventors or about state capitals. You could write programs about presidents or biology or whatever you are interested in.

Counting Answers

With a little bit of extra work, you could keep track of how many correct and incorrect answers were given and print the results at the end. Start by setting two counter variables to zero, C for correct answers and I for incorrect answers:

```
5 LET C = 0
6 LET I = 0
```

Put in a command to increase the C counter by 1 after a correct answer, just after line 150:

```
155 LET C = C + 1
```

This LET command makes the computer add 1 to C and then store the result back into C. You also need to increase the I counter by 1 after an incorrect answer, just after line 100:

```
105 LET I = I + 1
```

You would need to add these commands for the second question:

```
315 LET C = C + 1
295 LET I = I + 1
```

and similar commands for each question you have. Then at the end of the questions, you print the results:

```
950 PRINT C; " CORRECT ANSWERS"
960 PRINT I; " INCORRECT ANSWERS"
990 END
```

Now let's put all these pieces together and add some REMarks which tell the reader about the program. If a line number is followed by the letters REM (for remarks), the rest of the line can be used to tell the reader anything about the program. REMarks are shown when you LIST a program, but they are skipped over when

you RUN a program. REMarks should have information for the programmer, not for the user of a program.

You can use REMarks to give a title to your program (line 1), put your name in (line 2), and tell about the variables that are used (lines 3 and 4). You can add REMarks anywhere else in the program (see lines 8, 190, and 950):

```
1 REM - INVENTORS QUIZ -
2 REM     WRITTEN BY BEN SHNEIDERMAN
3 REM   C COUNTS CORRECT ANSWERS
4 REM   I COUNTS INCORRECT ANSWERS
5 LET C = 0
6 LET I = 0
8 REM   PRINT THE FIRST QUESTION
10 PRINT "WHO INVENTED THE PHONOGRAPH"
20 PRINT "1  JAMES WATT"
30 PRINT "2  BENJAMIN FRANKLIN"
40 PRINT "3  SAMUEL MORSE"
50 PRINT "4  THOMAS EDISON"
60 PRINT "TYPE THE NUMBER AND PRESS RETURN ";
70 INPUT A
80 IF A = 4 THEN 150
90 PRINT "THE INVENTOR OF THE PHONOGRAPH WAS"
100 PRINT "THOMAS EDISON"
105 LET I = I + 1
110 GOTO 200
150 PRINT "YOU ARE CORRECT"
155 LET C = C + 1
190 REM   PRINT THE SECOND QUESTION
200 PRINT "WHO INVENTED THE TELEPHONE"
210 PRINT "1  THOMAS EDISON"
220 PRINT "2  ALEXANDER GRAHAM BELL"
230 PRINT "3  LEE DE FOREST"
240 PRINT "4  GEORGE WESTINGHOUSE"
250 PRINT "TYPE THE NUMBER AND PRESS RETURN ";
260 INPUT A
```

```
270 IF A = 2 THEN 310
280 PRINT "THE INVENTOR OF THE TELEPHONE WAS"
290 PRINT "ALEXANDER GRAHAM BELL"
295 LET I = I + 1
300 GOTO 350
310 PRINT "YOU ARE CORRECT"
315 LET C = C + 1
350  . . .
     . . .
     . . .     (more questions)
     . . .
950 REM   PRINT THE FINAL SCORE
960 PRINT C; "CORRECT ANSWERS"
970 PRINT I; "INCORRECT ANSWERS"
990 END
```

If you retype this program you may want to renumber the lines because of the additions and uneven numbering. You can make up quizzes about cartoon characters, your favorite books, popular singers, or computers.

Dollar Banking

Real banks have many rules about how much money you can deposit or withdraw, about interest payments, extra expenses, and penalties. Just for fun and to learn a little about banking, we'll keep the rules simple. You start with $1000 and can deposit or withdraw money. You might do the following:

```
WELCOME TO THE FIRST BANK OF ATLANTIS

YOU HAVE 1000 DOLLARS
DO YOU WISH TO
1  WITHDRAW
2  DEPOSIT
3  GO HOME
```

```
TYPE A NUMBER AND PRESS RETURN ?1
HOW MUCH DO YOU WANT TO WITHDRAW
TYPE THE AMOUNT AND PRESS RETURN ?147

YOU HAVE 853 DOLLARS
DO YOU WISH TO
1  WITHDRAW
2  DEPOSIT
3  GO HOME
TYPE A NUMBER AND PRESS RETURN ?2
HOW MUCH DO YOU WANT TO DEPOSIT
TYPE THE AMOUNT AND PRESS RETURN ?22

YOU HAVE 875 DOLLARS
DO YOU WISH TO
1  WITHDRAW
2  DEPOSIT
3  GO HOME
TYPE A NUMBER AND PRESS RETURN ?1
HOW MUCH DO YOU WANT TO WITHDRAW
TYPE THE AMOUNT AND PRESS RETURN ?900
900 DOLLARS IS MORE THAN YOU HAVE
YOU CANNOT WITHDRAW THAT AMOUNT

YOU HAVE 875 DOLLARS
DO YOU WISH TO
1  WITHDRAW
2  DEPOSIT
3  GO HOME
TYPE A NUMBER AND PRESS RETURN ?3
THE FIRST BANK OF ATLANTIS THANKS
YOU FOR YOUR BUSINESS.  COME AGAIN.
```

The program uses IF commands to check the numbered choice and the amount withdrawn. Repeating the list of choices is done by having a GOTO that jumps to the early part of the program. The arithmetic variable D is the number of dollars you have, and A is the amount you withdraw or deposit.

```
10 LET D = 1000
20 PRINT "WELCOME TO THE FIRST BANK OF ATLANTIS"
25 PRINT
30 PRINT "YOU HAVE ";D;" DOLLARS"
40 PRINT "DO YOU WISH TO"
50 PRINT "1  WITHDRAW"
60 PRINT "2  DEPOSIT"
70 PRINT "3  GO HOME"
80 PRINT "TYPE A NUMBER AND PRESS RETURN ";
90 INPUT N
100 IF N = 1 THEN 200
110 IF N = 2 THEN 300
120 IF N = 3 THEN 400
130 GOTO 40
200 PRINT "HOW MUCH DO YOU WANT TO WITHDRAW"
210 PRINT "TYPE THE AMOUNT AND PRESS RETURN "
220 INPUT A
230 IF A > D THEN 260
240 LET D = D - A
250 GOTO 25
260 PRINT A; "DOLLARS IS MORE THAN YOU HAVE"
270 PRINT "YOU CANNOT WITHDRAW THAT AMOUNT"
280 GOTO 30
300 PRINT "HOW MUCH DO YOU WANT TO DEPOSIT"
310 PRINT "TYPE THE AMOUNT AND PRESS RETURN "
320 INPUT A
330 LET D = D + A
340 GOTO 25
400 PRINT "THE FIRST BANK OF ATLANTIS THANKS"
410 PRINT "YOU FOR YOUR BUSINESS.  COME AGAIN."
500 END
```

A program like this might be fun to use for teaching someone about banking. You might make some changes to this program and use it as part of a computer game based on Monopoly.

When you use several IF commands, be very careful that you have put in the right line numbers for the THEN and GOTO

commands. Make sure that after each group of lines there is a GOTO that takes you to the proper line for the next command. It's a good idea to think of several tests and check your program carefully. You take the job of the computer and walk through the lines of your program. Try out small parts of your program as you build it. Make sure that at each step the program is doing what it's supposed to do.

Summary

1. By using IF commands, you can build programs that change course based on what the user does. You can get the computer to guide the user and to respond to the user's choices.
2. You should know how to offer a set of choices to the user and then steer the computer to the commands that carry out the correct steps.
3. By careful use of GOTOs, you can get the computer to go back and repeat part of the program.
4. REMarks let you tell the reader something about your program.

Exercises

1. Write a program to print this multiple-choice question and let the users know if they got it right or not:

```
IN WHICH MOVIE DOES JABBA THE HUTT APPEAR
1  STAR WARS
2  THE EMPIRE STRIKES BACK
3  RETURN OF THE JEDI
TYPE THE NUMBER AND PRESS RETURN ?
```

   In case you forgot, the answer is choice 3.

2. (+) Write a program to print a multiple-choice question on a state capital. Give the name of a state and four choices for the state capital.
3. Make up a program to print three multiple-choice questions on characters in books or movies. Give the name of the character and then four choices. Let the users know if they got it right or not. Leave three blank lines between each question.
4. Make up a program for a younger brother or sister who needs practice in counting. Get the computer to type out some asterisks and then ask the child to type the number of asterisks. The computer will type NO or YES and go on to the next problem, like this:

```
* * *
?3
YES
* * * *
?2
NO
```

   You will probably have to help your brothers or sisters use this program. You might have to help them recognize the NO or YES. Your program should have at least six problems.
5. (+) Write a program to offer this shape-matching test:

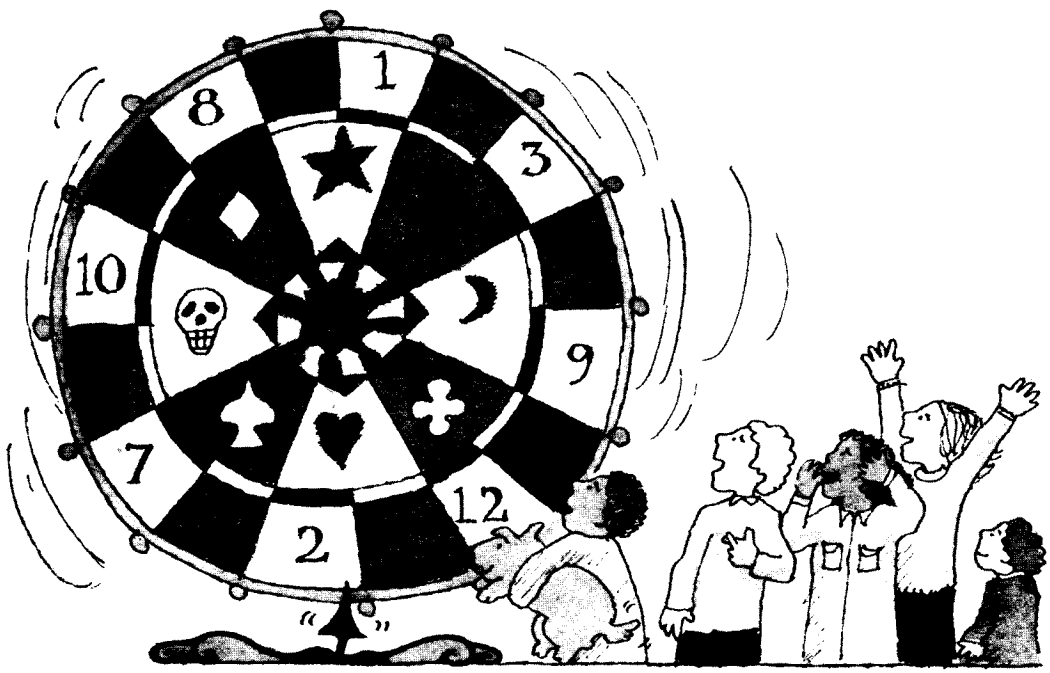
```
TYPE THE NUMBER WHICH MATCHES THIS
  SHAPE AND PRESS RETURN
        ****
           *
        *******


*******    ****       ********   ****
   *         *              *      *
****       *******    ********      ****
 (1)        (2)         (3)         (4)
```

The instructions and shapes are printed by simple PRINT commands. You will have to count spaces carefully. Let the user know if he or she got it right or not.

6. (*) Write a program like the dollar-banking program to keep track of the number of jelly beans you have. Instead of asking for WITHDRAW or DEPOSIT, ask for EAT SOME or BUY SOME MORE.

12 Random Chances

(Advanced chapter—optional reading)



▶ **PREVIEW**: The computer can print a number for you to use in dice playing games, or to make up arithmetic exercises. A number which the computer generates is called a *random number*. You can find out what the random number is by printing it out, or you can use it in your programs.

▶ **NEW IDEAS**: rolling dice, random numbers, `RND(1)`, function, `INT`, arithmetic exercises

Rolling Dice

Many board games use dice or a spinner to decide how many moves you get or which player wins points. You can get the computer to roll a die (singular of dice) or become the spinner for your games. The computer can generate a *random number* that is a decimal fraction between 0.0 and 1.0. Each time you ask for a random number, you get another number. If you run this program

```
10 PRINT RND(1)
```

you get one random number, such as:

```
0.378851
```

RND(1) is a *function* which gets a random number. Functions are special BASIC commands that you use as part of a PRINT command or a LET command.

Differences Among Computers. You can use RND(1) on the ATARI, Apple, and Commodore 64. RND works on Timex/Sinclair, IBM PC, and the TI 99/4. On the TRS–80, it is RND(0). ■

If you ask for five random numbers

```
10 FOR I = 1 TO 5
20 PRINT RND (1)
30 NEXT I
```

you will get five decimal numbers between 0.0 and 0.999999. The numbers you get may be different, but they might be like this:

```
0.637022
0.761092
0.018769
0.941335
0.719427
```

Fractions are fine, but for dice you need a number between 1 and 6. To get the numbers between 0.0 and 5.999999, multiply the fractions by 6 to get

```
3.822132
4.566552
0.112614
5.64801
4.316562
```

and then add 1 to these numbers to get numbers between 1.0 and 6.999999

```
4.822132
5.566552
1.112614
6.64801
5.316562
```

and finally throw away the decimal fractions part

```
4
5
1
6
5
```

You can do all this in your program by storing results with LET commands. You will need a special *function* called the *integer function* which throws away decimal fractions. For example, if N = 4.759219, then INT(N) is 4 without the fraction. Now you are ready to write a program which creates random numbers one step at a time.

```
10 FOR I = 1 TO 5
20 LET A = RND (1)
30 LET B = 6 * A
```

```
40 LET C = B + 1
50 LET D = INT(C)
60 PRINT D
70 NEXT I
```

You could also do all the arithmetic in the PRINT command:

```
10 FOR I = 1 TO 5
20 PRINT INT( 6*RND(1)+ 1 )
30 NEXT I
```

If you had a game that needed a spinner with numbers from 1 to 8, you could get the computer to be the spinner with this program:

```
10 PRINT "TO GET A NUMBER FROM 1 TO 8"
20 PRINT "AND PRESS RETURN "
30 INPUT A$
40 LET N = INT( 8*RND(1)+ 1 )
50 PRINT "YOUR NUMBER IS ", N
60 GOTO 10
```

This program will keep on going until you stop it. In line 40, RND(1) is multiplied by 8 because you need a number in the range of 1 to 8.

Indianapolis 500 Car Race

Imagine that the famous Indianapolis 500 car race had only two cars. The race goes for 500 miles. We'll call the cars the GREEN MACHINE and the SPEEDY SPIDER. Our program will get a random number between 1 and 50 to show how many miles each car has gone in the last 15 minutes. When one of the cars goes over 500 miles the race is over. The program might go like this:

```
DISTANCE COVERED

GREEN      SPEEDY
MACHINE    SPIDER

19         33
55         61
60         88
105        109
116        144
129        177
177        189
201        220
243        255
277        261
308        308
355        353
357        372
399        401
422        417
451        455
488        460
492        487
522        GREEN MACHINE IS THE WINNER
```

We'll use G to count the total distance for the GREEN MACHINE. S will count the total distance for the SPEEDY SPIDER. The IF commands will check to see if there is a winner yet. The program loops until there is a winner.

```
10 G = 0
20 S = 0
30 PRINT "DISTANCE COVERED"
40 PRINT
50 PRINT "GREEN", "SPEEDY"
60 PRINT "MACHINE", "SPIDER"
70 PRINT
```

```
80 LET A = INT( 50*RND(1) + 1)
90 LET G = G + A
100 PRINT G,
110 IF G >= 500 THEN 200
120 LET B = INT( 50*RND(1) + 1)
130 LET S = S + B
140 PRINT S
150 IF S >= 500 THEN 300
160 GOTO 80
200 PRINT "GREEN MACHINE IS THE WINNER"
210 GOTO 400
300 PRINT "SPEEDY SPIDER IS THE WINNER"
400 END
```

Could you change this program to have a third car? You can invent horse racing or other games using random numbers.

Number Guessing Game

A number guessing game that helps teach number facts can be easily written once you have a random number generator. The computer gets a random number in the range 1 to 100, and then you have to guess what that number is. You can write a program to make the computer print a message showing whether your guess was TOO HIGH or TOO LOW. The run might go like this:

```
THE COMPUTER HAS A NUMBER FROM 1 TO 100
GUESS THE NUMBER AND PRESS RETURN
?17
TOO LOW
?44
TOO HIGH
?29
TOO LOW
?35
TOO HIGH
```

```
?32
TOO LOW
?34
CORRECT. YOU GOT IT!
```

The program begins with the message and the generation of the random number from 1 to 100. Then two IF commands test the guess and jump to print a message. The GOTOs send the computer back up to get another guess:

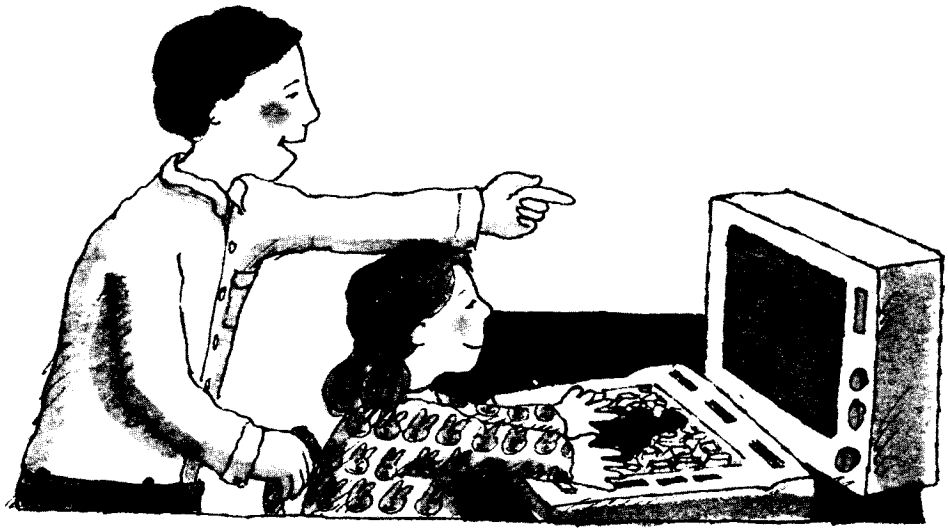
```
10 PRINT "THE COMPUTER HAS A NUMBER";
15 PRINT " FROM 1 TO 100"
20 PRINT "GUESS THE NUMBER AND PRESS RETURN"
30 LET X = INT( 100*RND(1) + 1 )
40 INPUT N
50 IF N > X THEN 90
60 IF N < X THEN 110
70 PRINT "CORRECT. YOU GOT IT!"
80 GOTO 10
90 PRINT "TOO HIGH"
100 GOTO 40
110 PRINT "TOO LOW"
120 GOTO 40
```

Once the correct number is guessed, the computer goes back to the beginning and starts the game over again with a new random number.

Arithmetic Expressions

Learning addition or multiplication takes practice, but it is hard to find someone who will sit for hours, offer new problems, and grade the answers. Aha! Why not get the computer to do it? The first program will be a simple addition exercise program that uses only the numbers from 1 to 5. This might be useful for kindergarteners or first graders who are just learning addition.

The computer will get two random numbers in the range 1 to 5,



type out the problem, and wait for the answer. If the answer is correct, the computer just types YES and goes on to the next problem. If the answer is wrong, the computer types NO, gives the correct answer, and then goes on to the next problem. Here's a part of the run:

```
3 + 1 = ?4
YES
1 + 4 = ?5
YES
4 + 5 = ?8
NO  9
```

Since this program is for children who may not be able to read, it makes no sense to print instructions. Someone older will have to get the child started and explain the symbols. Young children learn fast and can spend a half hour practicing. The program is

```
10 LET A = INT( 5*RND(1) + 1 )
20 LET B = INT( 5*RND(1) + 1 )
30 PRINT A; " + "; B; "= ";
40 INPUT C
50 IF A + B = C THEN 80
```

```
60 PRINT "NO  "; A+B
70 GOTO 10
80 PRINT "YES"
90 GOTO 10
```

The GOTO 10 commands send the computer back up to the first command to generate another problem. To stop this program, just press the BREAK key.

If you know children who need practice in arithmetic, you might let them try this program.

Summary

1. The RND(1) function generates random numbers between 0.0 and 0.999999.
2. You can use this function to produce random numbers in any range you want.
3. The INT function throws away the fraction of a number.
4. Dice rolling, number guessing, and arithmetic practice are just three uses of random number generators.

Exercises

1. (+) Use the RND(1) function to generate a random number in the range 1 to 2. If the result is 1, print the word HEADS; if it is 2, print the word TAILS.
2. Use the coin-tossing program from Exercise 1, and flip the coin 100 times. Count up and print out the number of HEADS and the number of TAILS.
3. (+) Change the number guessing program so that the program asks the user for the upper limit on the numbers. That should make the game more of a challenge for the user. How many guesses do you think it would take to find out the number if it was in the range of 1 to 1000?

4. Change the addition exercise program to ask for the biggest number that could be used in a problem. As users learn the addition table up to 5, they can then go up to 7, 10, 20, 100, or 1000.
5. Change the addition exercise program so that it is now a multiplication exercise program.
6. (*+) Change the addition exercise program so that it is now a subtraction exercise program. If A is smaller than B, don't use the numbers, because you would get a negative result. Get another pair of random numbers.
7. (*) You can get the computer to compose music with a random number generator. Let's say that each of the notes A, B, C, D, E, F, and G is matched up with the numbers 1, 2, 3, 4, 5, 6, and 7. Now write a program to print out 10 random numbers in between 1 and 7, for example:

```
PLAY THESE NOTES FOR THE NUMBERS
  A B C D E F G
  1 2 3 4 5 6 7
PRESS RETURN TO GET 10 NOTES ?
THE NOTES ARE 4  4  7  3  4  2  7  6  3  1
PRESS RETURN TO GET 10 NOTES ?
```

In this example, the notes generated by the computer are DDGCDBGFCA. Try to play these on a piano or other instrument. Extra added attraction: can you add seven IF commands to this program so that the program will print out the letters instead of the numbers?

13 Fill-in-the-Blanks Storytelling

- **PREVIEW**: In this chapter you will see how to ask for names, colors, or numbers from the user of the program. The names, colors, and numbers can be used to make up a story. People enjoy seeing their names used in a story printed on the computer.
- **NEW IDEAS**: writing stories, using `INPUT` strings in a printed story

Compunicorn Story Writer

A good storyteller knows how to change a story to fit the audience. A personal story which has danger and a hero who saves everyone can be a lot of fun. The computer can simply print out a whole story, but you could make it special. The computer can print a story that lets the user fill in words.

The Compunicorn Story Writer was created by my daughter Sara for her eighth birthday party. She decided to ask for a person's name, a favorite number, two foods, two colors, an animal, and an ice cream flavor. Sara used the answers in a story that she wrote. Sometimes the story can come out sounding funny and sometimes it's just plain silly, but everyone finds it fun.

This is what it looks like:

```
HELLO
THIS IS THE COMPUNICORN STORY WRITER
AFTER YOU ANSWER THESE QUESTIONS,
YOU WILL GET A STORY.
PRESS RETURN AFTER EVERY ANSWER.

WHAT IS YOUR NAME ?JENNA
WHAT IS YOUR FAVORITE
  NUMBER BETWEEN 1 AND 50 ?17
  FOOD ?PIZZA
  ANOTHER FAVORITE FOOD ?BROCCOLI
  COLOR ?BLUE
  ANOTHER FAVORITE COLOR ?PINK
  ANIMAL ?GIRAFFE
  FLAVOR OF ICE CREAM ?CHOCOLATE CHIP

THE STORY OF 17 UNICORNS

   ONCE UPON A TIME THERE WAS A
GROUP OF 17 UNICORNS GRAZING IN
A PATCH OF BLUE PIZZA.
```

```
ONE DAY THE YOUNGEST UNICORN NAMED
JENNA WHO WAS 17 YEARS OLD
LED THE GROUP TO SEARCH FOR A NEW
GRAZING PLACE.  THEY FOUND A PATCH OF
TASTY PINK BROCCOLI.  ALL OF A SUDDEN,
A BIG GIRAFFE CAME OUT AND SAID,
'TELL ME YOUR FAVORITE FLAVOR OF
ICE CREAM OR I WILL KILL YOU ALL!'
THE UNICORN NAMED JENNA SAID,
'I WILL TELL HIM THAT OUR FAVORITE
FLAVOR OF ICE CREAM IS CHOCOLATE CHIP.'
SHE DID, BUT THE GIRAFFE PUT THEM IN
A CAGE ANYWAY!  JENNA WAS SO LITTLE
THAT SHE COULD SLIP THROUGH THE BARS.
SHE TOUCHED HER HORN TO THE LOCK AND
THE UNICORNS ALL CAME OUT.
THEY WENT BACK TO THEIR GRAZING AND
LIVED HAPPILY EVER AFTER.
```

This program has three parts: the introduction, the questions, and the story. The introduction is simple since it has only DIM and PRINT commands:

```
5 DIM A$(30), C$(30), D$(30), E$(30), F$(30)
6 DIM G$(30), H$(30)
10 PRINT "HELLO"
20 PRINT "THIS IS THE COMPUNICORN STORY WRITER"
30 PRINT "AFTER YOU ANSWER THESE QUESTIONS,"
40 PRINT "YOU WILL GET A STORY."
50 PRINT "PRESS RETURN AFTER EVERY ANSWER."
60 PRINT
```

The INPUT variables go in alphabetical order, with $ signs for the string variables, A$, C$, D$, E$, F$, G$, and H$, and just a plain B for the number. Next, the questions are made of PRINT commands followed by INPUT commands.

```
70 PRINT "WHAT IS YOUR NAME ";
80 INPUT A$
90 PRINT "WHAT IS YOUR FAVORITE"
100 PRINT "  NUMBER BETWEEN 1 AND 50 ";
110 INPUT B
120 PRINT "  FOOD ";
130 INPUT C$
140 PRINT "  ANOTHER FAVORITE FOOD ";
150 INPUT D$
160 PRINT "  COLOR ";
170 INPUT E$
180 PRINT "  ANOTHER FAVORITE COLOR ";
190 INPUT F$
200 PRINT "  ANIMAL ";
210 INPUT G$
220 PRINT "  FLAVOR OF ICE CREAM ";
230 INPUT H$
```

All of the values of the variables are now ready to be used in the story. First skip two lines and print the title. The value of the variable B is used in the title. By using semicolons around the variable B, you make sure that the words get printed close to the number.

```
240 PRINT
250 PRINT
260 PRINT "THE STORY OF "; B ; " UNICORNS"
270 PRINT
```

On the ATARI you can clear the screen and have the story begin on the top by replacing line 240 with

```
240 PRINT "<ESC> <CTRL> <CLEAR>"
```

Within the quotes you press the ESC key, then the CTRL and CLEAR keys. Your screen will show a small curving arrow inside the quotes.

The story starts with an indented line and goes on by using the variables where they are needed:

```
280 PRINT "    ONCE UPON A TIME THERE WAS A"
290 PRINT "GROUP OF "; B ; " UNICORNS GRAZING IN"
295 PRINT "A PATCH OF "; E$; " "; C$; "."
300 PRINT "ONE DAY THE YOUNGEST UNICORN NAMED"
310 PRINT A$; " WHO WAS "; B; " YEARS OLD"
320 PRINT "LED THE GROUP TO SEARCH FOR A NEW"
330 PRINT "GRAZING PLACE.  THEY FOUND A PATCH OF"
340 PRINT "TASTY "; F$ ; " " ; D$ ;".";
345 PRINT "ALL OF A SUDDEN,"
350 PRINT "A BIG "; G$ ;" CAME OUT AND SAID,"
360 PRINT "'TELL ME YOUR FAVORITE FLAVOR OF"
370 PRINT "ICE CREAM OR I WILL KILL YOU ALL!'"
```

Since your ATARI has room for only 24 lines on the screen, it's a good idea to stop the story and let people read until they are ready to go on. You can do this by printing a message and waiting for INPUT of any character:

```
380 PRINT "  (TO GO ON, PRESS RETURN)"
390 INPUT Z$
```

This is an exciting point in the story. The pause builds interest in finding out what will happen. On with the story:

```
400 PRINT "THE UNICORN NAMED "; A$ ;" SAID,"
410 PRINT "'I WILL TELL HIM THAT OUR FAVORITE"
420 PRINT "FLAVOR OF ICE CREAM IS "; H$ ;".'"
430 PRINT "SHE DID, BUT THE "; G$ ;" PUT THEM IN"
440 PRINT "A CAGE ANYWAY! "; A$ ;" WAS SO LITTLE"
450 PRINT "THAT SHE COULD SLIP THOUGH THE BARS."
460 PRINT "SHE TOUCHED HER HORN TO THE LOCK AND"
470 PRINT "THE UNICORNS ALL CAME OUT."
480 PRINT "THEY WENT BACK TO THEIR GRAZING AND"
490 PRINT "LIVED HAPPILY EVER AFTER."
500 END
```

This program uses only INPUT and PRINT commands, but there is a lot of careful planning to make sure that the story comes out making sense. When you make up your own stories, write out the questions first and then create a story using the answers to the questions. The story is more enjoyable if the answers get used several times.

It's often fun to see how the story turns out. Strange things can happen, such as the patch of blue pizza or pink broccoli. Sara had a contest to see which of her friends could produce the funniest story. Everyone got to make a story but could vote only for someone else's story as the funniest.

Summary

1. Stories can be given a personal touch by asking users to provide words, names, or numbers for the story.
2. After the question-and-answer section of the program, the story is printed.
3. It is a good idea to stop the story every so often and have the readers press RETURN when they are ready to go on.

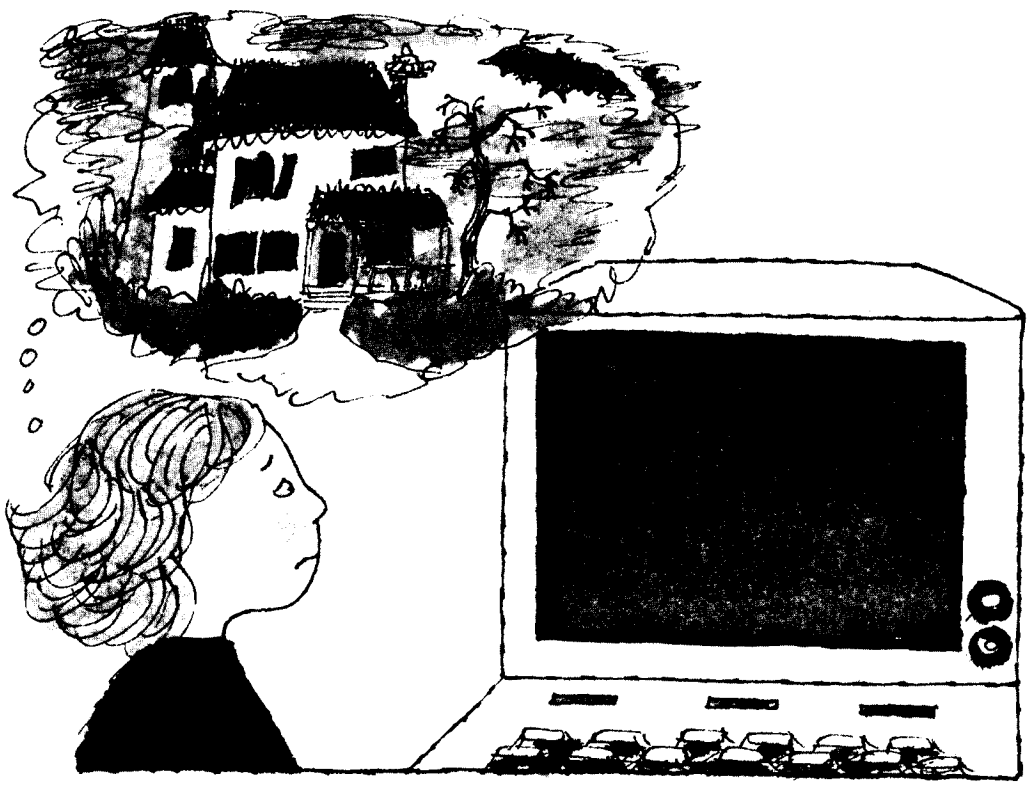
Exercises

1. Write a program that would produce this:

```
WHAT IS YOUR NAME ?LARRY
TYPE AN ARTICLE OF CLOTHING ?PANTS
TYPE AN ANIMAL ?RABBIT
TYPE A NUMBER ?174

LARRY REACHED INTO HIS PANTS
AND PULLED OUT 174 GIANT RABBITS
```

2. (+) Write a program that asks for (1) the names of two people, (2) a game that is played by two people, (3) a number of days that they play the game, and (4) a kind of candy which is used as a prize for the winner. Now write a short story using these ideas, and turn it into a program.
3. (*) Write a storytelling program about a circus, with users giving names for two clowns, some animals, and some refreshments.
4. (*) Write an adventure story about children caught in a storm on a ship out on the ocean. A sea creature comes up to the ship, but the hero takes a weapon and chases the creature away. The sun comes out, and they make it safely home. The users could provide names for the hero and the ship, describe the color of the water or sky, describe the creature, choose the weapon, and name the feelings at the end of the story.

14 The Mystery of the Haunted House



- **PREVIEW**: The Compunicorn Story Writer takes words and includes them in a story. An even more exciting story program is one in which the user is part of the story and can decide what happens next. This chapter starts with a mystery story and then shows you how to write your own.
- **NEW IDEAS**: stories with different endings, complicated paths through programs

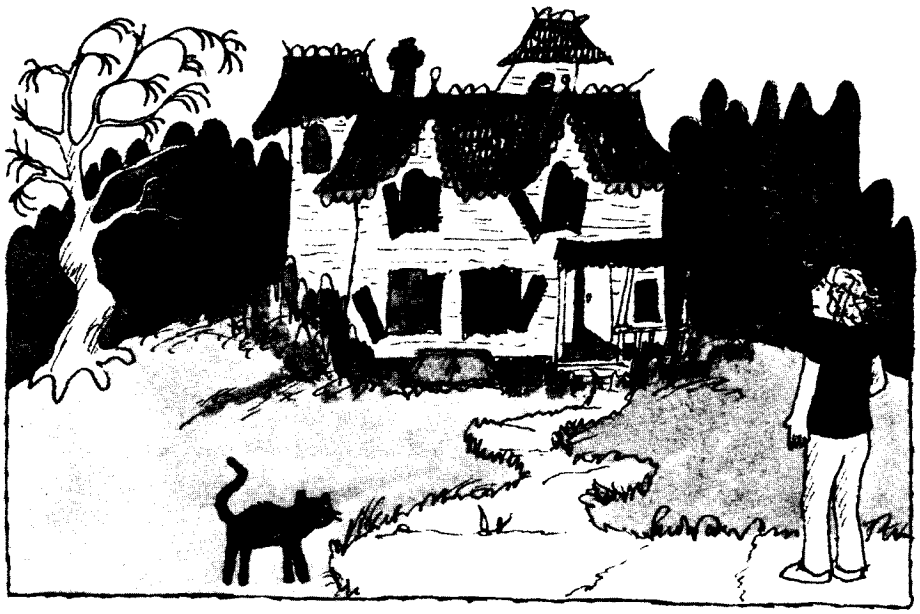
The Mystery Begins

Reading an adventure story or watching a mystery on television is fun. It can be even more exciting if you are in the story and can decide what happens next. There is a series of "Choose your own adventure" books by R. A. Montgomery such as *The Abominable Snowman* or *The Mystery of the Maya* which lets you choose the plot by answering questions and then jumping to some page in the book.

You can write your own adventure programs for your friends to try out. For instance, in this adventure you explore a haunted house. You can get out of the house safely without discovering the mystery, you can solve the mystery and slay the dragon, or you can be killed by the dragon—so watch out.

First, let's see what happens if you leave the house early in the adventure:

```
THE MYSTERY OF THE HAUNTED HOUSE

YOU ARE STANDING IN FRONT OF A
HAUNTED HOUSE WITH BROKEN WINDOWS.
TYPE  F  TO GO IN BY THE FRONT DOOR
      B  TO GO INTO THE BASEMENT
(PRESS RETURN AFTER EVERY CHOICE)
?F
THE DOOR CREAKS OPEN AND THEN SLAMS
SHUT BEHIND YOU.  A LOUD VOICE SAYS
'YOU BETTER RUN AWAY FROM THIS HOUSE.'
TYPE  L  TO LEAVE THE HOUSE
      B  TO GO INTO THE BASEMENT
?B
THE COLD, DARK, AND DAMP BASEMENT SENDS
A SHIVER THROUGH YOUR SPINE.  YOU SEE
TWO DOORS AND HEAR A HISSING SOUND.
TYPE  R  TO OPEN THE RIGHT DOOR
      L  TO OPEN THE LEFT DOOR
?L
```



```
YOU SEE A NARROW WET TUNNEL.
YOUR HEART POUNDS AS YOU CRAWL
THROUGH IT AND
YOU COME OUT SAFELY INTO THE SUNLIGHT.
YOU NEVER RETURN AND NEVER LEARN THE
MYSTERY OF THE HAUNTED HOUSE.
```

The version in which you meet and slay the dragon goes like this:

```
THE MYSTERY OF THE HAUNTED HOUSE

YOU ARE STANDING IN FRONT OF A
HAUNTED HOUSE WITH BROKEN WINDOWS.
TYPE  F  TO GO IN BY THE FRONT DOOR
      B  TO GO INTO THE BASEMENT
(PRESS RETURN AFTER EVERY CHOICE)
?F
THE DOOR CREAKS OPEN AND THEN SLAMS
SHUT BEHIND YOU.  A LOUD VOICE SAYS
'YOU BETTER RUN AWAY FROM THIS HOUSE.'
TYPE  L  TO LEAVE THE HOUSE
```

```
      B  TO GO INTO THE BASEMENT
?B
THE COLD, DARK, AND DAMP BASEMENT SENDS
A SHIVER THROUGH YOUR SPINE.  YOU SEE
TWO DOORS AND HEAR A HISSING SOUND.
TYPE  R  TO OPEN THE RIGHT DOOR
      L  TO OPEN THE LEFT DOOR
?R
A SLIMY SCALE-COVERED DRAGON
BREATHES FIRE AT YOU, BUT ONLY
BURNS YOUR HAIR SLIGHTLY.
YOU LOOK AROUND FOR WEAPONS.
TYPE  S  TO GRAB THE SWORD ON THE FLOOR
      G  TO TAKE THE GUN FROM A SHELF
?S
AS YOU GRAB THE SWORD, IT GLOWS AND
GIVES YOU STRENGTH.  YOU STAB AT THE
DRAGON'S HEART, AND IT TURNS INTO A
GOLD STATUE WITH DIAMOND EYES.  AT THAT
MOMENT, ALL THE EVIL IN THE WORLD TURNS
TO GOOD.  YOU RELAX, WALK PROUDLY INTO
THE SUNLIGHT, AND GET HOME SAFELY TO TELL
YOUR STORY.
```

The unfortunate ending comes if you decide to take the gun:

```
YOU SHOOT AT THE DRAGON'S STOMACH,
BUT THE BULLETS BOUNCE OFF.
THE DRAGON IS ANGERED.  IT GROWLS.
IT BREATHES FIRE AT YOU
AND BURNS YOU UP COMPLETELY.
SORRY.  IT WAS A BAD DAY FOR YOU.
```

You can change the ending if you want it to be gentler. The program is long but uses only PRINT, IF, INPUT, and GOTO commands. To prepare a story like this, you might want to draw a diagram (Figure 2) to show the plan of action. The numbers in the circles are the line numbers in the program. The letters on the arrows are the choices that the user gets to make. The shortest path

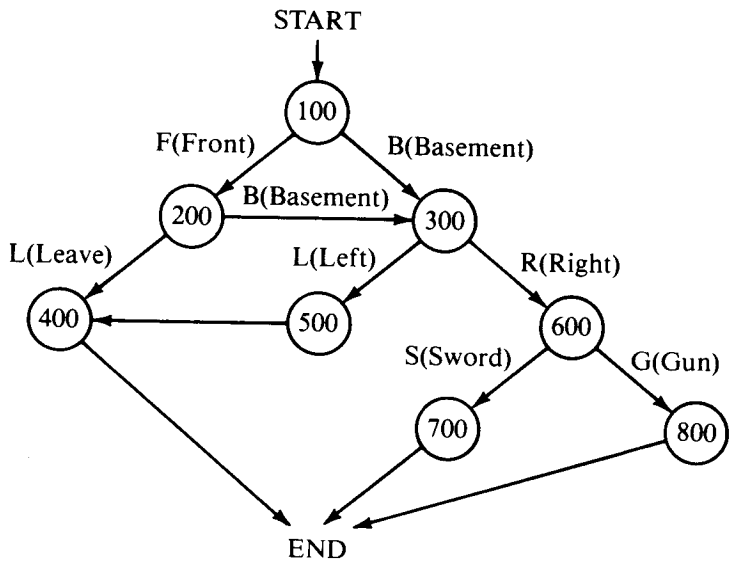


Figure 2. A diagram for the plan of action in "The Mystery of the Haunted House."

has three circles, the longest path has five circles. There are seven possible paths in this simple mystery.

If the user does not type one of the given letters, the choice is repeated. The program goes like this:

```
50 DIM A$(1)
100 PRINT "THE MYSTERY OF THE HAUNTED HOUSE"
110 PRINT
120 PRINT "YOU ARE STANDING IN FRONT OF A"
130 PRINT "HAUNTED HOUSE WITH BROKEN WINDOWS."
140 PRINT "TYPE  F  TO GO IN BY THE FRONT DOOR"
150 PRINT "      B  TO GO INTO THE BASEMENT"
160 PRINT "(PRESS RETURN AFTER EVERY CHOICE)"
170 INPUT A$
180 IF A$ = "F" THEN 200
190 IF A$ = "B" THEN 300
195 GOTO 140
200 PRINT "THE DOOR CREAKS OPEN AND THEN SLAMS"
210 PRINT "SHUT BEHIND YOU.  A LOUD VOICE SAYS,"
```

```
220 PRINT "'YOU BETTER RUN AWAY FROM THIS HOUSE.'"
230 PRINT "TYPE  L  TO LEAVE THE HOUSE"
240 PRINT "      B  TO GO INTO THE BASEMENT"
250 INPUT A$
260 IF A$ = "L" THEN 400
270 IF A$ = "B" THEN 300
280 GOTO 230
300 PRINT "THE COLD, DARK, AND DAMP BASEMENT SENDS
310 PRINT "A SHIVER THROUGH YOUR SPINE.  YOU SEE"
320 PRINT "TWO DOORS AND HEAR A HISSING SOUND."
330 PRINT "TYPE  R  TO OPEN THE RIGHT DOOR"
340 PRINT "      L  TO OPEN THE LEFT DOOR"
350 INPUT A$
360 IF A$ = "R" THEN 500
370 IF A$ = "L" THEN 600
380 GOTO 330
400 PRINT "YOU COME OUT SAFELY INTO THE SUNLIGHT."
410 PRINT "YOU NEVER RETURN AND NEVER LEARN THE"
420 PRINT "MYSTERY OF THE HAUNTED HOUSE."
430 GOTO 1000
500 PRINT "A SLIMY SCALE-COVERED DRAGON"
510 PRINT "BREATHES FIRE AT YOU, BUT ONLY"
520 PRINT "BURNS YOUR HAIR SLIGHTLY."
525 PRINT "YOU LOOK AROUND FOR WEAPONS."
530 PRINT "TYPE S TO GRAB THE SWORD ON THE FLOOR"
540 PRINT "      G TO TAKE THE GUN FROM A SHELF"
550 INPUT A$
560 IF A$ = "S" THEN 700
570 IF A$ = "G" THEN 800
580 GOTO 530
600 PRINT "YOU SEE A NARROW WET TUNNEL."
610 PRINT "YOUR HEART POUNDS AS YOU CRAWL"
620 PRINT "THROUGH IT AND"
630 GOTO 400
700 PRINT "AS YOU GRAB THE SWORD, IT GLOWS"
710 PRINT "AND GIVES YOU STRENGTH.  YOU STAB AT"
720 PRINT "THE DRAGON'S HEART, AND IT TURNS"
```

```
730 PRINT "INTO A GOLD STATUE WITH DIAMOND EYES."
740 PRINT "AT THAT MOMENT, ALL THE EVIL IN THE"
750 PRINT "WORLD TURNS TO GOOD.  YOU RELAX, WALK"
760 PRINT "PROUDLY INTO THE SUNLIGHT, AND GET"
770 PRINT "HOME TO TELL YOUR STORY."
780 GOTO 1000
800 PRINT "YOU SHOOT AT THE DRAGON'S STOMACH,"
810 PRINT "BUT THE BULLETS BOUNCE OFF."
820 PRINT "THE DRAGON IS ANGERED.  IT GROWLS."
830 PRINT "IT BREATHES FIRE AT YOU"
840 PRINT "AND BURNS YOU UP COMPLETELY."
850 PRINT "SORRY.  IT WAS A BAD DAY FOR YOU."
1000 END
```

Take the time to follow this program through for several paths until you feel that you understand it. The seven possible choices are

```
F-L
B-L
F-B-L
B-R-S
B-R-G
F-B-R-S
F-B-R-G
```

You may want to try your hand at writing an adventure in the jungle, a love story, a detective story about a strange murder, a happy story about a family picnic, or a fantasy about unicorns and Pegasus.

The Good Morning Program

This kind of question-asking program can also be useful for planning and organizing your life. You could make sure you got the day started out right by having the computer ask questions like this:

```
GOOD MORNING!

WHAT IS THE OUTSIDE TEMPERATURE
TYPE  A  FOR ABOVE FREEZING
      B  FOR BELOW FREEZING
?B
MAKE SURE THAT YOU WEAR WARM CLOTHES
AND TAKE YOUR GLOVES.
DO YOU HAVE YOUR LUNCH
TYPE  Y  FOR YES
      N  FOR NO
?Y
DO YOU HAVE YOUR HOMEWORK
TYPE  Y  FOR YES
      N  FOR NO
?N
LOOK FOR IT IN YOUR BEDROOM AND
```

```
THE KITCHEN.  DID YOU LEAVE ANY
OF YOUR BOOKS IN SCHOOL.
DO YOU HAVE TO FEED THE PETS
TYPE  C  TO FEED THE CAT
      G  TO FEED THE GOLDFISH
      N  NO FEEDING NEEDED
?C
THE CAT FOOD IS IN THE KITCHEN.
CHECK IF THE CAT NEEDS FRESH LITTER.
DO YOU HAVE TO FEED THE PETS
TYPE  C  TO FEED THE CAT
      G  TO FEED THE GOLDFISH
      N  NO FEEDING NEEDED
?N
HAVE A GOOD DAY AT SCHOOL.
```

In the last part of this run there were three choices. After one was chosen, the user got the chance to make a second choice. A paper checklist might be just as good for a short list, but the computer program might be fun. The computer does get to be useful if there are many tasks to remember.

Long checklists of activities might be useful for packing up for weekend trips, preparing a shopping list, or helping out new users of a computer.

Summary

1. Mystery stories can develop from decisions made by the user. The program offers two or more choices, and the user types one choice.
2. This same approach can be used to prepare useful checklist programs and teaching programs.

Exercises

1. Write a simple program with one question for the user:

```
CHOOSE THE CONTENTS OF MY HANDS
TYPE  R  FOR MY RIGHT HAND
      L  FOR MY LEFT HAND
?R
GOOD CHOICE, YOU GET CANDY
```

   If the user chooses L, he or she gets

```
BAD CHOICE, YOU GET A CANDY WRAPPER
```

2. Write a program about an adventure in a jungle. Give the user the choice of taking a boat or going on foot, then the choice of traveling by day or night. The user can meet crocodiles or snakes and try to capture them. Make sure there is at least one good and one bad ending.
3. (*) Write a checklist program for getting dressed. Choose different kinds of shoes, clothes, hats, and coats. You might keep track by drawing the clothing as they are chosen.
4. (*+) Write a program to help someone choose an activity. Start by asking for indoor or outdoor activities. Then ask if exercise or quiet activities are wanted. Finally, print out one or two activities for each category. Figure 3 has the diagram for this program. The second question will appear twice in your program.

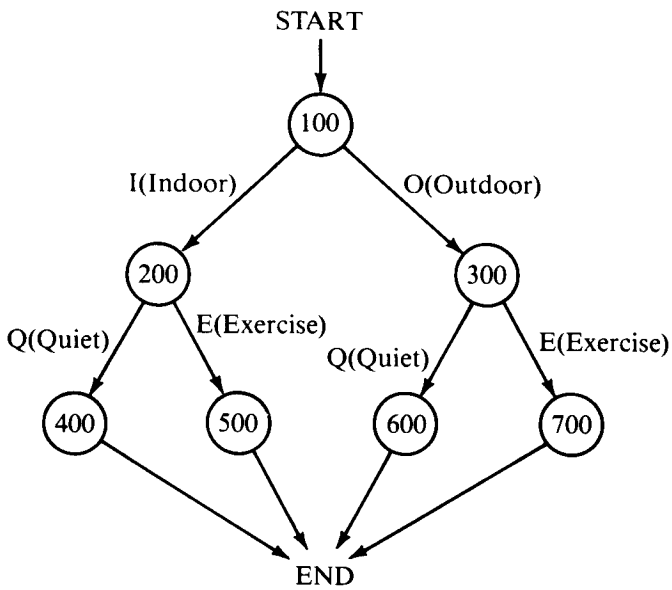
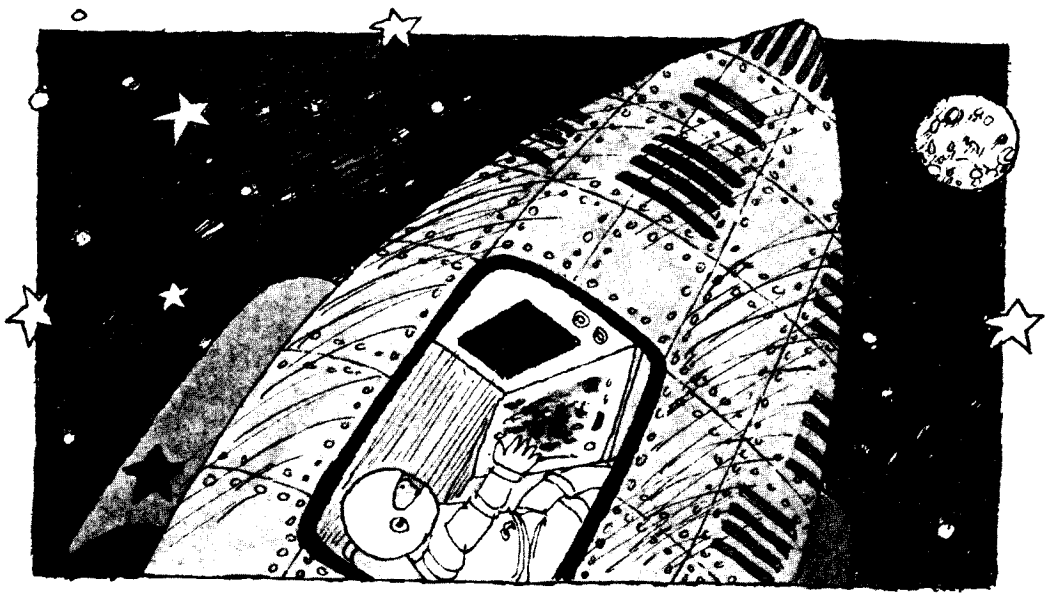


Figure 3. A diagram for Exercise 4.

15 Building Programs from Parts

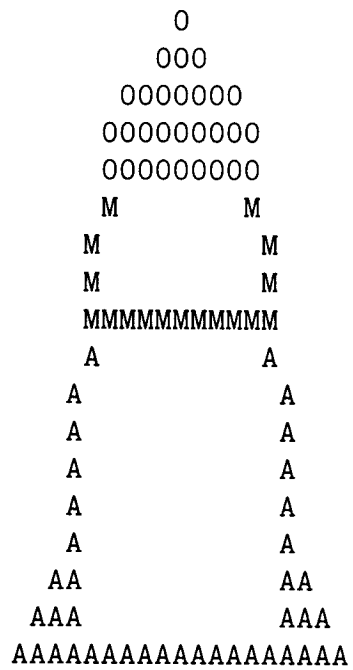


▶ **PREVIEW**: As you start to write longer programs you may find that some parts of the program will get repeated in two or more places. There is a shortcut when you need the same commands in two places: you can write the commands once and then call on them from two places (or more) in your program. These pieces of your program are called subroutines. Writing subroutines also helps to break up a complicated program into a group of smaller pieces that are easier to write.

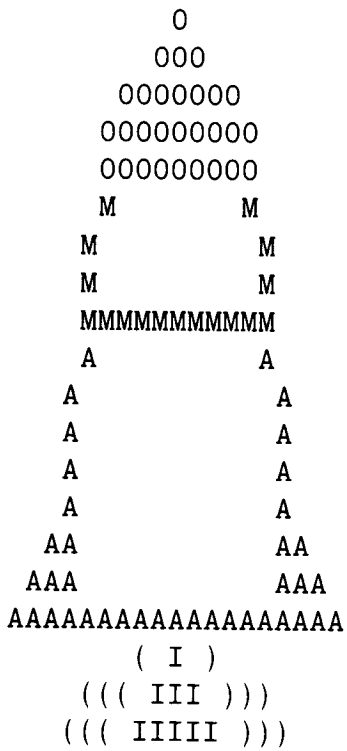
▶ **NEW IDEAS**: subroutines, `GOSUB`, `RETURN`

Building Rockets from Parts

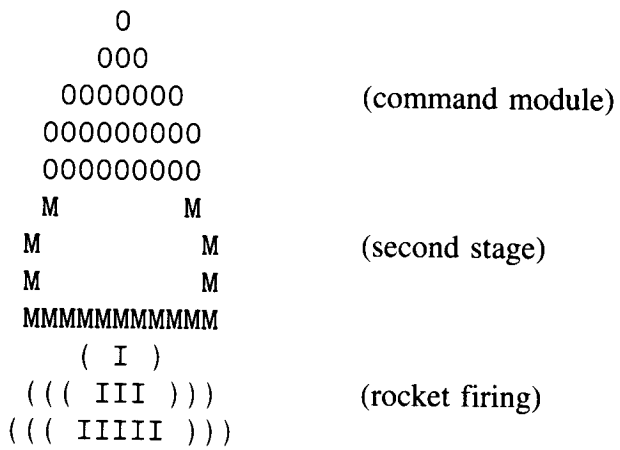
If you have a complicated and long project to do, it helps to break that project into smaller parts. You may not see how to do the big project, but you probably can figure out how to do each small part. You might want to print out a big rocket as part of a game or story program:



This might be what the rocket looks like just before blast-off. You could get this picture printed out easily enough with 17 PRINT commands. But now you might want to print the second picture in the story, which shows the rocket taking off:



To get this picture would take another 20 PRINT commands. The main booster fires for a while until it drops away, and then the second stage rocket begins firing:



This picture would take another 12 PRINT commands. Then the command module would be in orbit by itself:

```
     0
    000
   0000000
  000000000
  000000000
```

This last picture could be done with just 5 PRINT commands. So the whole story with four pictures would take a total of 54 PRINT commands. You could do it this way, but you might make a mistake in one of the parts. That would take quite a while to correct.

There is another way! You could write print commands for each part separately and then use the parts to build each picture. This means you have to plan ahead, but it is well worth it since the program is shorter and easier to write. We'll call each part a *subroutine* and begin it with a REMark to describe what it does. Each subroutine ends with a RETURN command. First, the command module subroutine:

```
100 REM     COMMAND MODULE SUBROUTINE
110 PRINT "            0"
120 PRINT "           000"
130 PRINT "          0000000"
140 PRINT "         000000000"
150 PRINT "         000000000"
160 RETURN
```

Next comes the second stage of the rocket, which we write as a subroutine. This part of the program uses line numbers beginning at 200, to keep it separate from other parts of the program:

```
200 REM    SECOND STAGE SUBROUTINE
210 PRINT "         M        M"
220 PRINT "        M          M"
230 PRINT "        M          M"
240 PRINT "        MMMMMMMMMMM"
250 RETURN
```

The third subroutine takes care of the main booster. It begins at line 300:

```
300 REM    MAIN BOOSTER SUBROUTINE
310 PRINT "         A          A"
320 PRINT "       A             A"
330 PRINT "       A             A"
340 PRINT "       A             A"
350 PRINT "       A             A"
360 PRINT "      AA             AA"
370 PRINT "     AAA             AAA"
380 PRINT "   AAAAAAAAAAAAAAAAAAA"
390 RETURN
```

Finally, the fourth subroutine is the rocket firing. It begins at line 400:

```
400 REM    ROCKET FIRING SUBROUTINE
410 PRINT "           ( I )"
420 PRINT "        ((( III )))"
430 PRINT "       ((( IIIII )))"
440 RETURN
```

Now that the parts of the rocket are ready, there has to be some way to use them in the program. The GOSUB command makes the computer jump to the subroutine and do the printing. Then the RETURN command sends the computer back to the line after the GOSUB. To print the first picture with the command module, second stage, and main booster, you only have to jump to three subroutines:

```
10 PRINT "READY FOR BLAST-OFF"
15 GOSUB 100
20 GOSUB 200
25 GOSUB 300
```

To get the picture of the blast-off, you need to use all four subroutines:

```
30 PRINT "3, 2, 1 BLAST-OFF"
35 GOSUB 100
40 GOSUB 200
45 GOSUB 300
50 GOSUB 400
```

The picture of the second stage rocket firing uses only three subroutines:

```
55 PRINT "SECOND STAGE ROCKET FIRING"
60 GOSUB 100
65 GOSUB 200
70 GOSUB 400
```

Finally, the picture of the command module in orbit uses only one subroutine:

```
75 PRINT "COMMAND MODULE IN ORBIT"
80 GOSUB 100
```

When all four pictures have been drawn, the program is done. To get to the END command without going through the subroutine, you need to include a GOTO command:

```
85 GOTO 500
```

To finish the program, be sure to include the END command

```
500 END
```

after all four subroutines. The complete program follows below. Instead of the 54 lines for the program without subroutines, we have 37 lines to do the printing, 4 lines of printed comments, and 4 REMarks.

```
10 PRINT "READY FOR BLAST-OFF"
15 GOSUB 100
20 GOSUB 200
25 GOSUB 300
30 PRINT "3, 2, 1 BLAST-OFF"
35 GOSUB 100
40 GOSUB 200
45 GOSUB 300
50 GOSUB 400
55 PRINT "SECOND STAGE ROCKET FIRING"
60 GOSUB 100
65 GOSUB 200
70 GOSUB 400
75 PRINT "COMMAND MODULE IN ORBIT"
80 GOSUB 100
85 GOTO 500
100 REM    COMMAND MODULE SUBROUTINE
110 PRINT "            0"
120 PRINT "           000"
130 PRINT "         0000000"
140 PRINT "        000000000"
150 PRINT "        000000000"
160 RETURN
200 REM    SECOND STAGE SUBROUTINE
210 PRINT "         M       M"
220 PRINT "        M         M"
230 PRINT "        M         M"
240 PRINT "        MMMMMMMMMMM"
250 RETURN
```

```
300 REM    MAIN BOOSTER SUBROUTINE
310 PRINT "        A          A"
320 PRINT "       A            A"
330 PRINT "       A            A"
340 PRINT "       A            A"
350 PRINT "       A            A"
360 PRINT "      AA            AA"
370 PRINT "     AAA            AAA"
380 PRINT "   AAAAAAAAAAAAAAAAAAA"
390 RETURN
400 REM    ROCKET FIRING SUBROUTINE
410 PRINT "           ( I )"
420 PRINT "        ((( III )))"
430 PRINT "       ((( IIIII )))"
440 RETURN
500 END
```

Working with Subroutines

Subroutines are a wonderful invention because they make it much easier to build bigger programs and to make changes. For example, let's say you decided to have two second stages in your rocket. It's easy; just add another jump to a subroutine

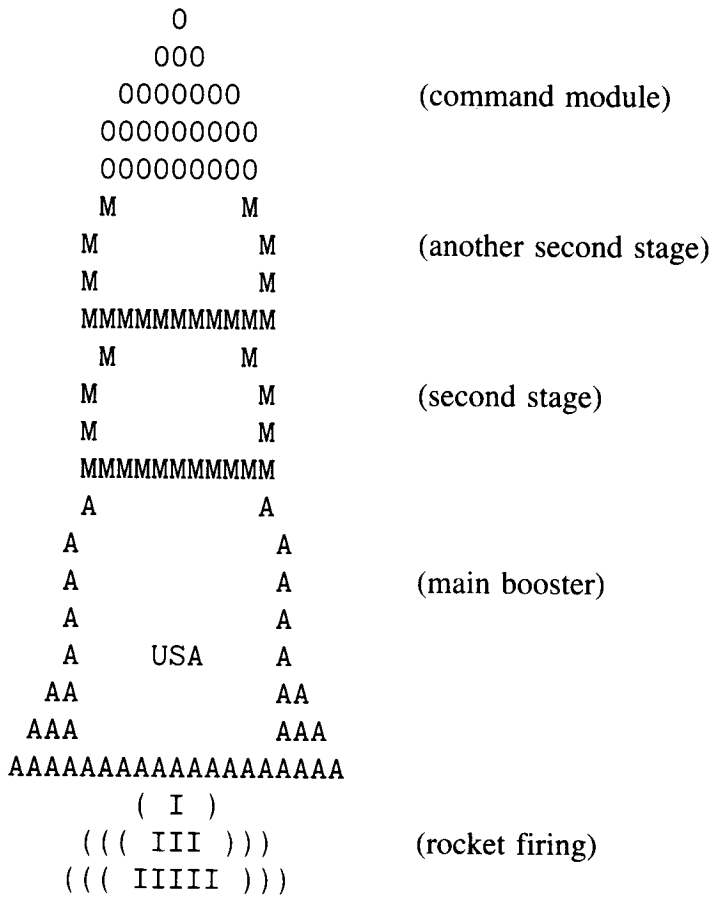
```
22 GOSUB 200
```

and now you've got a taller rocket.

You can make changes inside a subroutine, and then every time it is used the change will show up in the program. You might change the main booster subroutine to have a decoration on the rocket or the letters USA:

```
350 PRINT "       A     USA     A"
```

This change would show up every time the subroutine is used. Here's the taller rocket with the decoration, as it blasts off:



It takes some practice to see how to break programs apart into subroutines. Subroutines become more useful as your programs get longer and more complicated.

Summary

1. When programs become long and parts of the program are repeated, subroutines help organize and shorten the program.
2. You jump to the subroutine with a GOSUB command and go back with the RETURN command.

3. To help organize your program, begin every subroutine with a REMark explaining it.
4. You need a GOTO command to jump over the subroutines and get to the END command.

Exercises

1. Write a program to draw this castle with a roof, three floors, and the entrance:

```
MMM    MMM    MMM    MMM
MMM    MMM    MMM    MMM
MMMMMMMMMMMMMMMMMMMMMMM
I                       I
I  IIIIII   IIIIII  I
I  I    I   I    I  I
I  IIIIII   IIIIII  I
I                       I
I  IIIIII   IIIIII  I
I  I    I   I    I  I
I  IIIIII   IIIIII  I
I                       I
I  IIIIII   IIIIII  I
I  I    I   I    I  I
I  IIIIII   IIIIII  I
I                       I
I                       I
I      IIIIIIIII        I
I      I  0 0  I        I
I      I       I        I
I      I       I        I
I      I       I        I
OOOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOOO
```

Write three subroutines: one for the roof, one for the floor, and one for the entrance. Then use one GOSUB to print the roof, three GOSUBs to print the floors, and one GOSUB to print the entrance.

If you want to get fancy, you can replace the three GOSUBs for the floors with a loop and one GOSUB. Now you can print a 10-story castle if you want

2. (+) First make three flowers lying on their sides as subroutines. One of your flowers might look like this:

```
                                        FF
                                      FFF
                                    FFFFF
                                  FFFFFFFOO
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIFFFFFFFOOOOO
                                  FFFFFFFOO
                                    FFFFF
                                      FFF
                                        FF
```

You can design the other two flowers as you like.

Now print one of the first, two of the second, another one of the first, and three of the third—all right next to each other. If you have a printing terminal, you could make a very long flower garden to decorate your classroom or bedroom. Since the flowers will be printed on their sides, you'll have to turn the paper to the left to make the flowers point up.

3. (*) You can print messages in big letters to make signs for your door or your room. If your name is Anna, you might want to print:

```
AAA
AAAAAA
AAAAAAAAAAAA
AAAAAAAAAAAAAAAA
        AAA     AAAAAA
        AAA      AAAAAAAA
        AAA     AAAAAA
AAAAAAAAAAAAAAAAA
AAAAAAAAAAAA
AAAAAA
AAA

NNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNN
                    NNNNNN
                NNNNNN
            NNNNNN
        NNNNNN
NNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNN

NNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNN
                    NNNNNN
                NNNNNN
            NNNNNN
        NNNNNN
NNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNN

AAA
AAAAAA
AAAAAAAAAAAA
AAAAAAAAAAAAAAAA
        AAA     AAAAAA
        AAA      AAAAAAAA
        AAA     AAAAAA
AAAAAAAAAAAAAAAAA
AAAAAAAAAAAA
AAAAAA
AAA
```

Write the subroutines for the letter A and for the letter N, then use four GOSUBs to get the job done.

You could work with your friends and create more letters of the alphabet so that you could write out whole words and sentences.

Let's Keep Learning

Congratulations! You have finished this introduction to computer programming in BASIC. You deserve to be congratulated because you have learned many important ideas that are used in computer programming:

- ☐ printing strings
- ☐ doing loops
- ☐ taking input from a user
- ☐ calculating with numbers
- ☐ using variables to save values
- ☐ making decisions and jumping over commands
- ☐ building programs from parts

You know enough BASIC to write useful and enjoyable programs to:

- ☐ print poems, shapes, pictures, and stories
- ☐ keep a telephone book
- ☐ add, subtract, multiply, and divide numbers
- ☐ keep track of your allowance
- ☐ make up riddle or quiz programs
- ☐ write storytelling programs
- ☐ create mystery and adventure games

Some people are satisfied knowing just this much about BASIC. They can write programs when they need to or just for fun. They are in charge of the computer and can learn more when they want to. Now they can explore other interests such as basketball, reading, or dancing.

Let's Learn More BASIC

Other people get very excited about learning more BASIC. They want to know all the commands and all the ways to use the commands. They want to write longer programs and use some of the fancy parts of their computer—the color graphics, sound, disks, joysticks, game paddles, or printers.

This chapter is a quick tour of some other BASIC commands, programming ideas, and computer uses. Read this only if you want to learn more about BASIC.

Fancy Calculations

In Chapter 8 you learned to do calculations using arithmetic variables. Changing Fahrenheit to Celsius took three LET commands:

```
50 LET G = F - 32
60 LET H = G * 5
70 LET C = H / 9
```

If you study more BASIC, you'll learn that there is a shortcut which lets you do it all in one LET command:

```
50 LET C = 5/9 * (F - 32)
```

Parentheses ( ) help you write fancy calculations in one LET command. But you have to learn the rules of where to put the parentheses.

Computers are especially useful for doing math problems. If you are into math, you can write programs to do percentages, fractions, decimals, trigonometry, exponents, or square roots. In BASIC there is a function—SQR—that lets you take square roots. You can find out the square root of 9 by running this program:

```
10 PRINT SQR (9)
```

The answer is 3. There are more functions in BASIC that help you do calculations. Check your computer manual.

Commanding an Army of Numbers

The programs in this book never had more than 10 variables. Using one letter of the alphabet for each variable was easy. You never ran out of letters. As you get to write longer programs, you may need many variables. For example, you may want to keep track of the number of hits for each player during a baseball game. If there are 29 players, you would need 29 variables.

The following program uses the DIM—Dimension—command to set up 29 integer variables named H(1), H(2) . . . H(29). This is called an *array* of integers. Lines 20 – 40 set all 29 variables to 0. Then lines 50 – 80 ask which player got a hit. If player number 0 is used, the program goes on to print out the total number of hits for each player (lines 120 – 150). If the INPUT value N was 27, that means that player 27 got a hit. In line 100, one gets added to H(27) to keep track of the number of hits. The program keeps repeating the PRINT and INPUT commands until N = 0.

```
10 DIM H(29)
20 FOR I = 1 TO 29
30 LET H(I) = 0
40 NEXT I
50 PRINT "TYPE THE NUMBER OF THE PLAYER"
60 PRINT "WHO GOT A HIT, OR 0 AT THE"
70 PRINT "END OF THE GAME.  PRESS RETURN."
80 INPUT N
90 IF N = 0 THEN 120
100 LET H(N) = H(N) + 1
110 GOTO 50
120 PRINT "PLAYER", "HITS"
130 FOR I = 1 TO 29
140 PRINT I, H(I)
150 NEXT I
```

This program will give you an idea of how arrays are used. Check your computer manual or a more advanced book to get a full explanation.

Arrays are useful in keeping track of such things as rainfall for each of the 365 days of the year, speed for each of the 22 swimmers in a race, or the amount of money to charge each of the 41 people on a newspaper route.

You can also have arrays of strings such as the names of the top 40 record albums, the names of the 17 people you want to invite to a party, or the 60 strings needed to draw a spaceship on your screen.

Sum and Average

When you start using arrays, you'll learn and think of more programming ideas. An idea for a program is called an *algorithm* or a plan. An important algorithm is adding up a list of numbers. Chapter 9 showed how to add up numbers which were typed in. Now, you can see how to add up the numbers in an array. The idea is the same. Start by setting a total variable T to zero. Then loop through the array and add each value to the total variable.

These commands add up the total number of hits by the baseball team:

```
160 LET T = 0
170 FOR I = 1 TO 29
180 LET T = T + H(I)
190 NEXT I
200 PRINT "TOTAL TEAM HITS = "; T
```

To find the average number of hits per player, simply divide by 29

```
210 LET A = T / 29
220 PRINT "AVERAGE HITS = "; A
```

Since average may be used several times in a program, it makes sense to write it as a subroutine. The average subroutine could then be called from any point in the program.

Biggest and Smallest

Another important algorithm finds the biggest value in an array. The idea is to assume that H(1) is the biggest and copy it into B, for biggest. Then compare H(2), H(3) . . . H(29) to B. If any value is bigger than B, copy it into B.

```
230 LET B = H(1)
240 FOR I = 2 TO 29
250 IF B >= H(I) THEN 270
260 LET B = H(I)
270 NEXT I
280 PRINT "BIGGEST NUMBER OF HITS = "; B
```

Finding the smallest can be done in almost the same way. Just change line 250 to

```
250 IF B <= H(I) THEN 270
```

and change the message in 280, of course.

These algorithms take a while to understand. A lot of the fun in programming is learning new algorithms and inventing your own.

Pluck My Strings

This book showed how to print, input, store, and compare strings. Sometimes it is useful to change one letter in a string, split a string in two, or glue two strings together. There are string functions in BASIC to help you do these jobs.

You could take a string containing a title like THE BLACK STALLION and break it into three strings: THE, BLACK, and

STALLION. Now you can print these on three lines in the center of your screen.

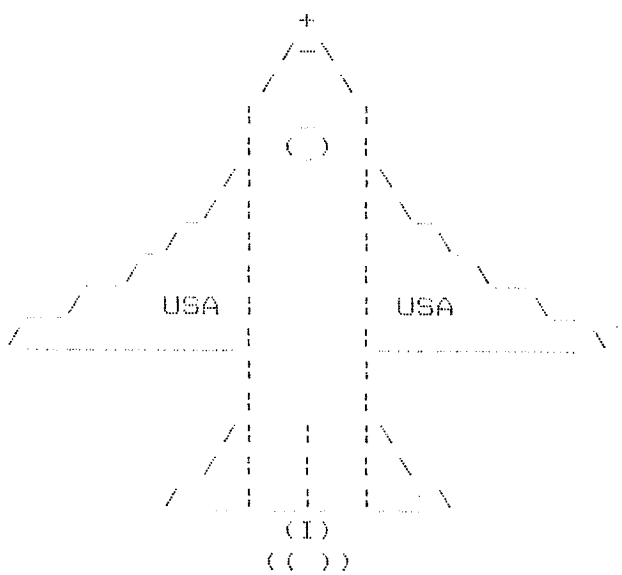
```
THE
BLACK
STALLION
```

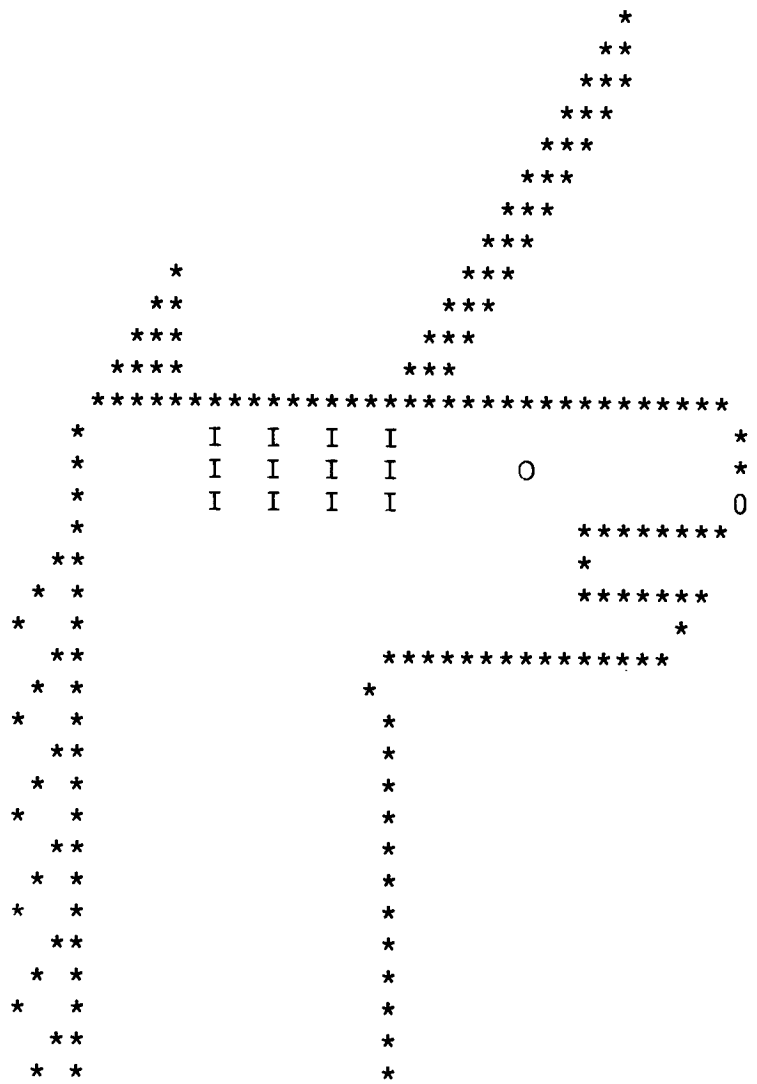
You may also want the computer to search if one string is part of another string. For example, ART is part of DARTH VADER. With just a little bit of study, you should be able to do many things with strings.

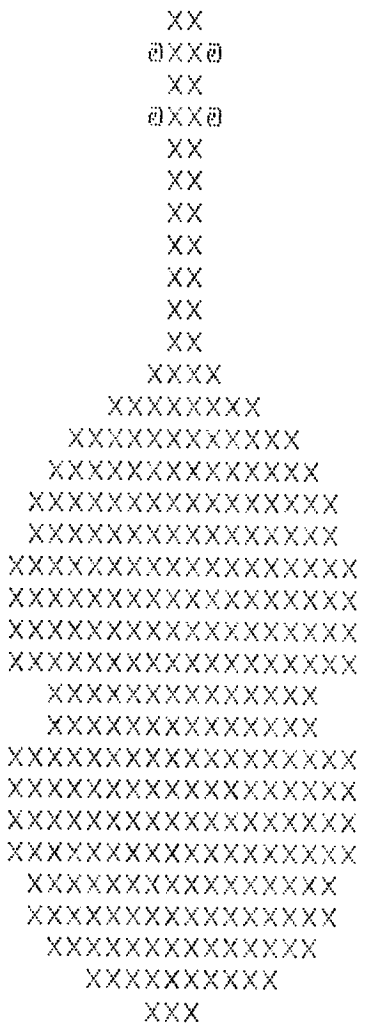
Special Effects

Graphics

This book showed how to make shapes and pictures on your screen or printer. You used the PRINT command and then figured out where each character belonged. You could try to make simple computer graphics like these:







The ATARI and most microcomputers have special graphics characters and color to help you make more detailed pictures or people, spaceships, or flowers. Drawing a picture on the screen is the first step in making fancy graphics.

The second step would be to produce *animation*—moving pictures. This gets you into advanced programming, but the idea is to show one picture and then change part of it quickly. If you put together several changes in a row, you can begin to make movies.

Filmmakers and cartoon artists use computers to make movies. *TRON* and the *Star Wars* series used computer animation for parts of the movies. Video and arcade games are good examples of computer animation. You can find out how to do some graphics by reading the manual that came with your computer. There are program packages to help you make color graphics on your screen. Some of these need a joystick or game paddle. You can find out more about graphics in *Personal Graphics* by Michael P. Barnett and Graham K. Bennett.

Sound

Many microcomputers have a built-in loudspeaker or use the loudspeaker in the display screen. They can be programmed to make sounds that are high or low. Getting single sounds or tones is easy, but putting them together to make a song is more complicated. Some computer scientists and engineers have found ways to get computers to make human voice sounds. The Texas Instruments Speak and Spell machine uses a computer voice to help teach spelling.

Other Programming Languages

BASIC is the most popular language for microcomputers. There are many other languages that are worth learning. If you like drawing pictures and computer graphics, you should look into LOGO. In LOGO you control an electronic turtle on your screen. Simple commands make the turtle go forward, turn left or right, or repeat some steps. The fun part is watching the turtle draw shapes that you have programmed. You can make shapes such as flowers, houses, birds, or circles.

There are hundreds of programming languages. Each one is especially useful for some kinds of work. You may see the names of some of them such as Pascal, FORTRAN, COBOL, or PILOT.

Word Processors

Learning to program is only one use of your computer. Most people who use computers use programs written by other people. They may use a word processor to help them write.

Word processors help you write and change stories, letters, or books. Word processing programs are built up from complicated string operations. I used a word processor in writing this book. It helped me to make changes easily. I could add an exercise, change an example, or correct an error and then get a clean printout. The Bank Street Writer is a word processing program that is very easy to learn and to use.

Using a word processor is great because you can easily fix your spelling mistakes, improve your writing style, and add sentences anywhere. Then when you are satisfied with how your work looks on the screen, you can print it.

Program Packages

Besides programming and word processing, you can use your computer for learning spelling, arithmetic, science, or health. There are thousands of program packages to help you learn different subjects.

There are also thousands of game programs for fun. They can help you develop quick reactions and teach you how to solve some kinds of problems.

If you are choosing a learning or game package, be sure to try it

out first. Make sure it does what you need and that you understand how to use it.

Most program packages are carefully designed and tested. But you may find that some packages have poorly written manuals, confusing screen displays, and nasty or useless error messages. If this happens, you can complain to the store or write a letter to the manufacturer. You can help fight for better quality program packages.

Magazines

One way of learning more about BASIC and computers is to read one of the many computer magazines. *ENTER* and *Family Computing* are meant for children and their families. There are several magazines just for ATARI users such as the *ATARI Connection*. There are many other magazines that will let you know about new ideas in microcomputers: *Creative Computing*, *Byte*, *Personal Computing*, *Electronic Classroom News*, *Popular Computing*, *The Computing Teacher*, and others.

Computers and People

Computers are fun to use. They can also be very valuable and powerful tools for helping people. Computers can be used in medicine, business, education, and in your home. But every powerful tool can be used in harmful ways. Computers have been used to steal money or to trick people. The movie *Wargames* showed some of the dangers of misuse of military computers.

When you see people using computers or when you use computers, think carefully. Think about how you can avoid harming people and how you can be most helpful.

Computers are toys and tools. You can use them to have fun and

to get your work done. You are in charge of the computer and can use it to help a first grader learn spelling, to organize a family business, to keep a mailing list for a community group, or to play games with your friends. Think about how you can share your knowledge about computers with others. People are what really count—friends, teachers, parents, brothers, and sisters. Can you think of creative ways of exploring the computer with these people?

Please let me know what you come up with. Good luck.

Solutions to Selected Exercises

The solutions for some of the exercises are included so that you can compare your solutions to these. These solutions were run on an ATARI 800 computer and printed on the ATARI 820, a 40-column printer. This set of solutions was done as a family project by Joseph, Mildred, and Tracy Johnson.

The programs were saved on the ATARI Cassette 410 Program Recorder. The programs were printed by simply typing the `LIST "P"` command. To get the output printed, the `PRINT` commands were changed to `LPRINT` commands. This approach means that the `?` produced by the `INPUT` command does not appear. In printing the output of the running program, `LPRINT` commands were added for the values entered with the `INPUT` commands.

Exercise 2

```
10 PRINT "ABCDEFG"
20 PRINT "HIJKLMNOP"
30 PRINT "QRSTUVWXYZ"

ABCDEFG
HIJKLMNOP
QRSTUVWXYZ
```

Exercise 4

```
10 PRINT "RRRRRRRRR"
20 PRINT "R       R"
30 PRINT "R       R"
40 PRINT "RRRRRRRRR"

RRRRRRRRR
R       R
R       R
RRRRRRRRR
```

Exercise 7

```
10 PRINT " M     M"
20 PRINT "M M   M M"
30 PRINT "I  ===  I"
40 PRINT "I       I"
50 PRINT "I O   O I"
60 PRINT "I       I"
70 PRINT " I -O- I"
80 PRINT "  I   I"
90 PRINT "   III"
```

```
 M     M
M M   M M
I  ===  I
I       I
I O   O I
I       I
 I -O- I
  I   I
   III
```

2

Exercise 1

```
10 PRINT "KG","KILOGRAM"
20 PRINT "CM","CENTIMETER"
30 PRINT "MM","MILLIMETER"
```

```
KG          KILOGRAM
CM          CENTIMETER
MM          MILLIMETER
```

Exercise 3

```
10 PRINT "+000000","+XXXXXX","+IIIIII"
20 PRINT "+000000","+XXXXXX","+IIIIII"
30 PRINT "+000000","+XXXXXX","+IIIIII"
40 PRINT "+      ","+      ","+      "
50 PRINT "+      ","+      ","+      "
60 PRINT "+      ","+      ","+      "
```

```
+000000   +XXXXXX   +IIIIII
+000000   +XXXXXX   +IIIIII
+000000   +XXXXXX   +IIIIII
+         +         +
+         +         +
+         +         +
```

Exercise 2

```
10 FOR W=1 TO 3
20 PRINT "000000000000000"
30 PRINT "XXXXXXXXXXXXXX"
40 NEXT W
```

```
000000000000000
XXXXXXXXXXXXXX
000000000000000
XXXXXXXXXXXXXX
000000000000000
XXXXXXXXXXXXXX
```

Exercise 4

```
10 FOR G=1 TO 6
20 PRINT "+---------I"
30 NEXT G
40 FOR F=1 TO 8
50 PRINT "+"
60 NEXT F
```

```
+---------I
+---------I
+---------I
+---------I
+---------I
+---------I
+
+
+
+
+
+
+
+
```

4

Exercise 1

```
10 FOR O=1 TO 3
20 FOR N=1 TO 2
30 PRINT "NO"
40 NEXT N
50 PRINT "YES"
60 NEXT O
```

```
NO
NO
YES
NO
NO
YES
NO
NO
YES
```

Exercise 3

```
10 FOR F=1 TO 2
20 FOR L=1 TO 3
30 PRINT "+XXXXXXXXXXXXXXXXX"
40 NEXT L
50 PRINT "+0000000000000000"
60 NEXT F
70 FOR M=1 TO 7
80 PRINT "+"
90 NEXT M
```

```
+XXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXX
+0000000000000000
+XXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXX
+0000000000000000
+
+
+
+
+
+
+
```

Exercise 4

```
10 FOR H=1 TO 10
20 PRINT "TICTOC"
30 FOR S=1 TO 380
40 NEXT S
50 NEXT H
```

```
TICTOC
TICTOC
TICTOC
TICTOC
TICTOC
TICTOC
TICTOC
TICTOC
TICTOC
TICTOC
```

5

Exercise 2

```
10 PRINT "TYPE THE NUMBER OF SECONDS YOU
 WANT THE CLOCK TO RUN"
20 INPUT F
30 FOR N=1 TO F
40 PRINT "TICTOC"
50 FOR S=1 TO 380
60 NEXT S
70 NEXT N
80 PRINT "TIME IS UP"

TYPE THE NUMBER OF SECONDS YOU WANT THE
CLOCK TO RUN
5
TICTOC
TICTOC
TICTOC
TICTOC
TICTOC
TIME IS UP
```

Exercise 5

```
10 DIM A$(100)
20 PRINT "TYPE A SHORT MESSAGE AND PRESS
 RETURN"
30 INPUT A$
40 PRINT "HOW MANY TIMES DO YOU WANT "
41 PRINT "THIS MESSAGE"
50 INPUT J
60 FOR T=1 TO J
70 PRINT A$
80 NEXT T
```

```
TYPE A SHORT MESSAGE AND PRESS RETURN
?   MY NAME IS THEA
HOW MANY TIMES DO YOU WANT
THIS MESSAGE

?  4
 MY NAME IS THEA
 MY NAME IS THEA
 MY NAME IS THEA
 MY NAME IS THEA
```

Exercise 1

```
10 FOR B=1 TO 7
20 PRINT B;"POTATO"
30 NEXT B
40 PRINT "MORE"
```

```
1          POTATO
2          POTATO
3          POTATO
4          POTATO
5          POTATO
6          POTATO
7          POTATO
MORE
```

Exercise 3

```
10 PRINT "I WAS SO HUNGRY THAT"
20 PRINT "I ATE 1 ICE CREAM CONE"
30 FOR R=2 TO 5
40 PRINT "I ATE ";R;" ICE CREAM CONES"
50 NEXT R
60 PRINT "I ATE TOO MANY ICE CREAM CONES
"

I WAS SO HUNGRY THAT
I ATE 1 ICE CREAM CONE
I ATE 2 ICE CREAM CONES
I ATE 3 ICE CREAM CONES
I ATE 4 ICE CREAM CONES
I ATE 5 ICE CREAM CONES
I ATE TOO MANY ICE CREAM CONES
```

7

Exercise 2

```
10 PRINT 24*7;" = HOURS IN A WEEK"

168 = HOURS IN A WEEK
```

Exercise 4

```
10 FOR F=1 TO 6
20 PRINT F,F*12
30 NEXT F

1           12
2           24
3           36
4           48
5           60
6           72
```

Exercise 6

```
10 PRINT "THIS PROGRAM WILL CALCULATE"
11 PRINT "THE NUMBER OF INCHES IF YOU"
12 PRINT "GIVE THE NUMBER.OF FEET"
20 PRINT "HOW MANY FEET DO YOU HAVE"
30 INPUT X
40 PRINT "THERE ARE ";12*X;" INCHES"

THIS PROGRAM WILL CALCULATE
THE NUMBER OF INCHES IF YOU
GIVE THE NUMBER OF FEET
HOW MANY FEET DO YOU HAVE
? 5
THERE ARE 60 INCHES
```

8

Exercise 2

```
10 H=60%60
20 PRINT "THE NUMBER OF SECONDS IN"
21 PRINT "AN HOUR IS ";H
30 D=24%H
40 PRINT "THE NUMBER OF SECONDS IN"
41 PRINT "A DAY IS ";D

THE NUMBER OF SECONDS IN
AN HOUR IS 3600
THE NUMBER OF SECONDS IN
A DAY IS 86400
```

Exercise 6

```
10 PRINT "TYPE THE NUMBER OF STAMPS FOR"
12 PRINT "EACH COUNTRY AND PRESS RETURN"
20 PRINT "FRANCE";
30 INPUT F
40 PRINT "ENGLAND";
50 INPUT E
60 PRINT "ITALY";
70 INPUT I
80 PRINT "GERMANY";
90 INPUT G
100 T=F+E+I+G
110 PRINT "THE TOTAL NUMBER OF STAMPS IS
 ";T

TYPE THE NUMBER OF STAMPS FOR
EACH COUNTRY AND PRESS RETURN
FRANCE
?  5
ENGLAND
?  11
ITALY
?  6
GERMANY
?  9
THE TOTAL NUMBER OF STAMPS IS 31
```

9

Exercise 2

```
10 T=0
20 FOR D=1 TO 5
30 PRINT "TYPE THE NUMBER OF PROBLEMS"
35 PRINT " COMPLETED FOR DAY ";D;
38 PRINT " AND PRESS RETURN"
40 INPUT P
50 T=T+P
60 NEXT D
70 PRINT "PROBLEMS COMPLETED ARE ";T

TYPE THE NUMBER OF PROBLEMS
COMPLETED FOR DAY 1
AND PRESS RETURN
28
TYPE THE NUMBER OF PROBLEMS
COMPLETED FOR DAY 2
AND PRESS RETURN
34
TYPE THE NUMBER OF PROBLEMS
COMPLETED FOR DAY 3
AND PRESS RETURN
67
TYPE THE NUMBER OF PROBLEMS
COMPLETED FOR DAY 4
AND PRESS RETURN
59
TYPE THE NUMBER OF PROBLEMS
COMPLETED FOR DAY 5
AND PRESS RETURN
89
PROBLEMS COMPLETED ARE 277
```

10

Exercise 2

```
10 PRINT "HOW MANY LAPS DID YOU SWIM";
20 INPUT L
30 IF L>=16 THEN 100
40 PRINT "DOES NOT QUALIFY"
50 GOTO 200
100 PRINT "QUALIFIES"
200 END

HOW MANY LAPS DID YOU SWIM
?  12
DOES NOT QUALIFY
HOW MANY LAPS DID YOU SWIM
?  16
QUALIFIES
```

Exercise 5

```
10 PRINT "TYPE IN THE SPEED";
20 INPUT S
30 IF S>55 THEN 100
40 IF S<40 THEN 110
50 PRINT "LEGAL SPEED"
60 GO TO 150
100 PRINT "ILLEGAL SPEED - TOO FAST"
105 GO TO 150
110 PRINT "ILLEGAL SPEED - TOO SLOW"
150 END

TYPE IN THE SPEED
35
ILLEGAL SPEED - TOO SLOW
TYPE IN THE SPEED
65
ILLEGAL SPEED - TOO FAST
TYPE IN THE SPEED
50
LEGAL SPEED
```

11

Exercise 2

```
10 PRINT "WHAT IS THE CAPITAL OF "
15 PRINT "NEW YORK STATE?"
20 PRINT "1.  NEW YORK CITY"
30 PRINT "2.  SYRACUSE"
40 PRINT "3.  BUFFALO"
50 PRINT "4.  ALBANY"
60 PRINT "TYPE THE NUMBER OF THE CORRECT
 CITY";
70 INPUT G
80 IF G=4 THEN 110
90 PRINT "INCORRECT"
100 GOTO 140
110 PRINT "CORRECT"
140 END
```

```
WHAT IS THE CAPITAL OF
NEW YORK STATE?
1.  NEW YORK CITY
2.  SYRACUSE
3.  BUFFALO
4.  ALBANY
TYPE THE NUMBER OF THE CORRECT CITY
2
INCORRECT

WHAT IS THE CAPITAL OF
NEW YORK STATE?
1.  NEW YORK CITY
2.  SYRACUSE
3.  BUFFALO
4.  ALBANY
TYPE THE NUMBER OF THE CORRECT CITY
4
CORRECT
```

Exercise 5

```
10 PRINT "TYPE.THE NUMBER THAT MATCHES"
15 PRINT "THIS SHAPE"
20 PRINT "****"
30 PRINT "   *"
40 PRINT "*******"
50 PRINT
60 PRINT "*******   ****      ********
 ****"
70 PRINT "   *         *            *
    *"
80 PRINT "****      *******   ********
   ****"
90 PRINT " (1)       (2)       (3)
  (4)"
100 PRINT
110 INPUT T
120 IF T=2 THEN 150
130 PRINT "WRONG"
140 GOTO 180
150 PRINT "CORRECT"
180 END

TYPE THE NUMBER THAT MATCHES
THIS SHAPE
****
   *
*******

*******   ****      ********   ****
   *         *            *       *
****      *******   ********      ****
  (1)       (2)       (3)      (4)



2
CORRECT
```

12

Exercise 1

```
10 LET A=INT(2*RND(1)+1)
20 IF A=1 THEN PRINT "HEADS"
30 IF A=2 THEN PRINT "TAILS"
40 GO TO 10

HEADS
HEADS
TAILS
HEADS
HEADS
HEADS
HEADS
TAILS
HEADS
TAILS
TAILS
```

Exercise 3

```
10 PRINT "WHAT IS THE HIGHEST NUMBER"
15 PRINT "YOU WANT TO GUESS?"
20 INPUT N
30 PRINT "THE COMPUTER HAS A NUMBER FROM
 1 TO ";N
40 LET A=INT(N*RND(1)+1)
50 PRINT "GUESS THE NUMBER AND"
55 PRINT "PRESS RETURN"
60 INPUT J
65 IF J=A THEN 110
70 IF J>A THEN PRINT "TOO HIGH"
80 IF J<A THEN PRINT "TOO LOW"
90 IF J=A THEN PRINT "CORRECT, YOU GOT I
T"
100 GOTO 50
110 PRINT "CORRECT!, YOU GOT IT!"
120 GOTO 10
```

```
WHAT IS THE HIGHEST NUMBER
YOU WANT TO GUESS?
100
THE COMPUTER HAS A NUMBER FROM 1 TO 100
GUESS THE NUMBER AND
PRESS RETURN
50
TOO HIGH
GUESS THE NUMBER AND
PRESS RETURN
25
TOO LOW
GUESS THE NUMBER AND
PRESS RETURN
33
TOO LOW
GUESS THE NUMBER AND
PRESS RETURN
40
TOO HIGH
GUESS THE NUMBER AND
PRESS RETURN
36
TOO LOW
GUESS THE NUMBER AND
PRESS RETURN
38
CORRECT!, YOU GOT IT!
WHAT IS THE HIGHEST NUMBER
YOU WANT TO GUESS?
```

Exercise 6

```
10 LET A=INT(10*RND(1)+1)
20 LET B=INT(10*RND(1)+1)
30 IF B>A THEN 10
40 PRINT A;" - ";B;" = ";
50 INPUT Z
60 IF A-B=Z THEN 90
70 PRINT "NO"
80 GOTO 40
90 PRINT "YES"
100 GOTO 10

2 - 2 =
0
YES
9 - 1 =
7
NO
9 - 1 =
8
YES
7 - 1 =
6
YES
5 - 1 =
4
YES
8 - 4 =
4
YES
7 - 3 =
```

13

Exercise 2

```
10 DIM A$(20)
20 DIM B$(20)
30 DIM C$(20)
40 DIM D$(20)
50 PRINT "NAME ONE PERSON WHO WILL PLAY"
;
60 INPUT A$
65 PRINT "NAME THE OTHER PERSON";
70 INPUT B$
80 PRINT "WHAT GAME WILL THEY PLAY";
90 INPUT C$
100 PRINT "HOW MANY DAYS WILL THEY PLAY"
;
110 INPUT N
120 PRINT "WHAT CANDY WILL BE THE PRIZE"
;
130 INPUT D$
140 PRINT A$;" AND ";B$;" AGREED TO PLAY
 ";C$;" ON ";N;" CONSECUTIVE DAYS"
150 PRINT "THE WINNER ON EACH DAY"
151 PRINT "WOULD GET ONE ";D$
152 PRINT "FROM THE OTHER."
160 PRINT "AT THE END OF ";N;" DAYS,"
161 PRINT B$;" HAD 4 ";D$;"S.  ";A$
162 PRINT "HAD NONE; SHE HAD EATEN HERS.
"
```

(continued on next page)

```
NAME ONE PERSON WHO WILL PLAY
ALICE
NAME THE OTHER PERSON
HARRY
WHAT GAME WILL THEY PLAY
BADMINTON
HOW MANY DAYS WILL THEY PLAY
20
WHAT CANDY WILL BE THE PRIZE
MILKYWAY
ALICE AND HARRY AGREED TO PLAY BADMINTON
 ON 20 CONSECUTIVE DAYS
THE WINNER ON EACH DAY
WOULD GET ONE MILKYWAY
FROM THE OTHER.
AT THE END OF 20 DAYS,
HARRY HAD 4 MILKYWAYS.  ALICE
HAD NONE;  SHE HAD EATEN HERS.
```

14

Exercise 4

```
10 REM CHOOSING AN ACTIVITY
20 PRINT "TYPE 1 IF YOU WANT AN OUTDOOR
ACTIVITY"
30 PRINT "TYPE 2 FOR AN INDOOR ACTIVITY"
40 INPUT A
50 IF A=1 THEN 200
60 PRINT "DO YOU WANT QUIET ACTIVITIES"
61 PRINT "OR EXERCISE?"
70 PRINT "TYPE 3 FOR QUIET ACTIVITIES"
80 PRINT "TYPE 4 FOR EXERCISE"
90 INPUT B
100 IF B=3 THEN 150
110 PRINT "YOU MAY CHOOSE RUNNING IN"
111 PRINT "PLACE, JUMPING JACKS, SIT-UPS
,"
112 PRINT "OR LIFTING WEIGHTS"
120 END
150 PRINT "IF YOU HAVE SOMEONE TO PLAY"
151 PRINT "WITH, YOU MAY CHOOSE:"
152 PRINT "CHECKERS, CHESS, OR"
153 PRINT "BACKGAMMON"
160 PRINT "IF YOU ARE ALONE, YOU MAY"
161 PRINT "CHOOSE:  READING, SOLITAIRE,"
162 PRINT "OR SEWING"
170 END
200 PRINT "DO YOU WANT QUIET ACTIVITIES"
201 PRINT "OR EXERCISE?"
210 PRINT "TYPE 6 FOR QUIET"
220 PRINT "TYPE 7 FOR EXERCISE"
230 INPUT T
240 IF T=6 THEN 300
250 PRINT "YOU MIGHT RIDE A BIKE, JOG,"
251 PRINT "CROSS-COUNTRY SKI, ICE SKATE,
"
252 PRINT "OR SWIM, DEPENDING ON"
253 PRINT "THE WEATHER"
260 END
300 PRINT "YOU MIGHT TAKE A WALK, WEED"
301 PRINT "THE GARDEN, OR TRY"
302 PRINT "BIRDWATCHING"
```

(continued on next page)

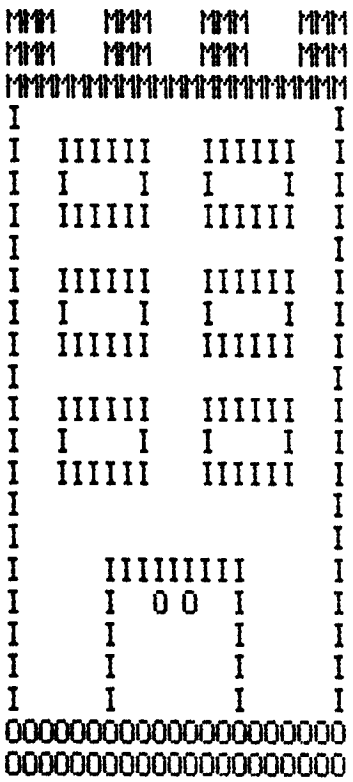
```
TYPE 1 IF YOU WANT AN OUTDOOR ACTIVITY
TYPE 2 FOR AN INDOOR ACTIVITY


1
DO YOU WANT QUIET ACTIVITIES
OR EXERCISE?
TYPE 6 FOR QUIET
TYPE 7 FOR EXERCISE


7
YOU MIGHT RIDE A BIKE, OR JOG,
CROSS-COUNTRY SKI, ICE SKATE,
OR SWIM, DEPENDING ON
THE WEATHER
```

15

Exercise 1

```
5 REM CH 15 EXER 1
10 REM PROGRAM TO DRAW CASTLE
20 GOSUB 100
30 FOR F=1 TO 3
40 GOSUB 200
50 NEXT F
60 GOSUB 300
80 END
100 REM - ROOF SUBROUTINE
110 PRINT "MMM   MMM   MMM   MMM"
120 PRINT "MMM   MMM   MMM   MMM"
130 PRINT "MMMMMMMMMMMMMMMMMMMMM"
140 RETURN
200 REM - ONE FLOOR OF BUILDING
210 PRINT "I                   I"
220 PRINT "I  IIIIII   IIIIII  I"
230 PRINT "I  I    I   I    I  I"
240 PRINT "I  IIIIII   IIIIII  I"
250 RETURN
300 REM ENTRANCE
310 PRINT "I                   I"
320 PRINT "I                   I"
330 PRINT "I     IIIIIIIII     I"
350 PRINT "I     I  O O  I     I"
360 FOR G=1 TO 3
370 PRINT "I     I       I     I"
380 NEXT G
390 PRINT "OOOOOOOOOOOOOOOOOOOO"
395 PRINT "OOOOOOOOOOOOOOOOOOOO"
```

(continued on next page)



Index

FPT

$8.95

UNLOCK THE CREATIVITY IN YOUR KIDS!

Packaged programs are fine, but they can get boring fast. Ask any kid. But when kids write their own programs, *they're* in charge. The number of things they can do at the computer is limited only by their imaginations. And, as any kid will tell you, taking command of a powerful computer is *fun!*

INTRODUCING THE CHALLENGES AND REWARDS OF COMPUTING . . .

This book teaches kids how to write commands for the ATARI® Home Computers by using a programming language called BASIC. It is written for 8- to 14-year-olds to read and use on their own. No knowledge of programming or math beyond third-grade arithmetic is necessary.

Let's Learn BASIC is activity-oriented. Kids are introduced to the step-by-step process of programming right from the start by "Getting the Computer to Do Printing." Repeating PRINT commands using program loops gives kids a feel for the power of the computer. Writing programs that do arithmetic, make decisions, and create stories comes after that.

Along the way, kids will learn how to

- print poems, shapes, pictures, and stories
- keep an address book
- add, subtract, multiply, and divide numbers
- keep track of their allowance
- make up riddle and quiz programs
- write storytelling programs
- create mystery and adventure games
- and much, much more!

Throughout, lots of examples and exercises use games, stories, graphics, and riddles to maintain a high level of interest. *Let's Learn BASIC* has been extensively "kid-tested" to make sure that it's *easy* to read, *fun* to use, and *effective* in teaching BASIC programming concepts.

ABOUT THE AUTHOR

Ben Shneiderman is Associate Professor of Computer Science and Head, Human-Computer Interaction Laboratory at the University of Maryland. He is also the father of two children. He wrote this book for his 8-year-old daughter, Sara, when he found that other computer books were either too shallow (picture books) or too technical (reference guides) to be of much use or interest to the average pre-teen or young teenager. *Let's Learn BASIC* is the solution that worked for Dr. Shneiderman's youngster—and it will do the same for yours!

Little, Brown and Company • Boston • Toronto

0884870

ISBN 0-316-78722-1