Cat. No. 62-2067

Ten Dollars and Ninety-Five Cents

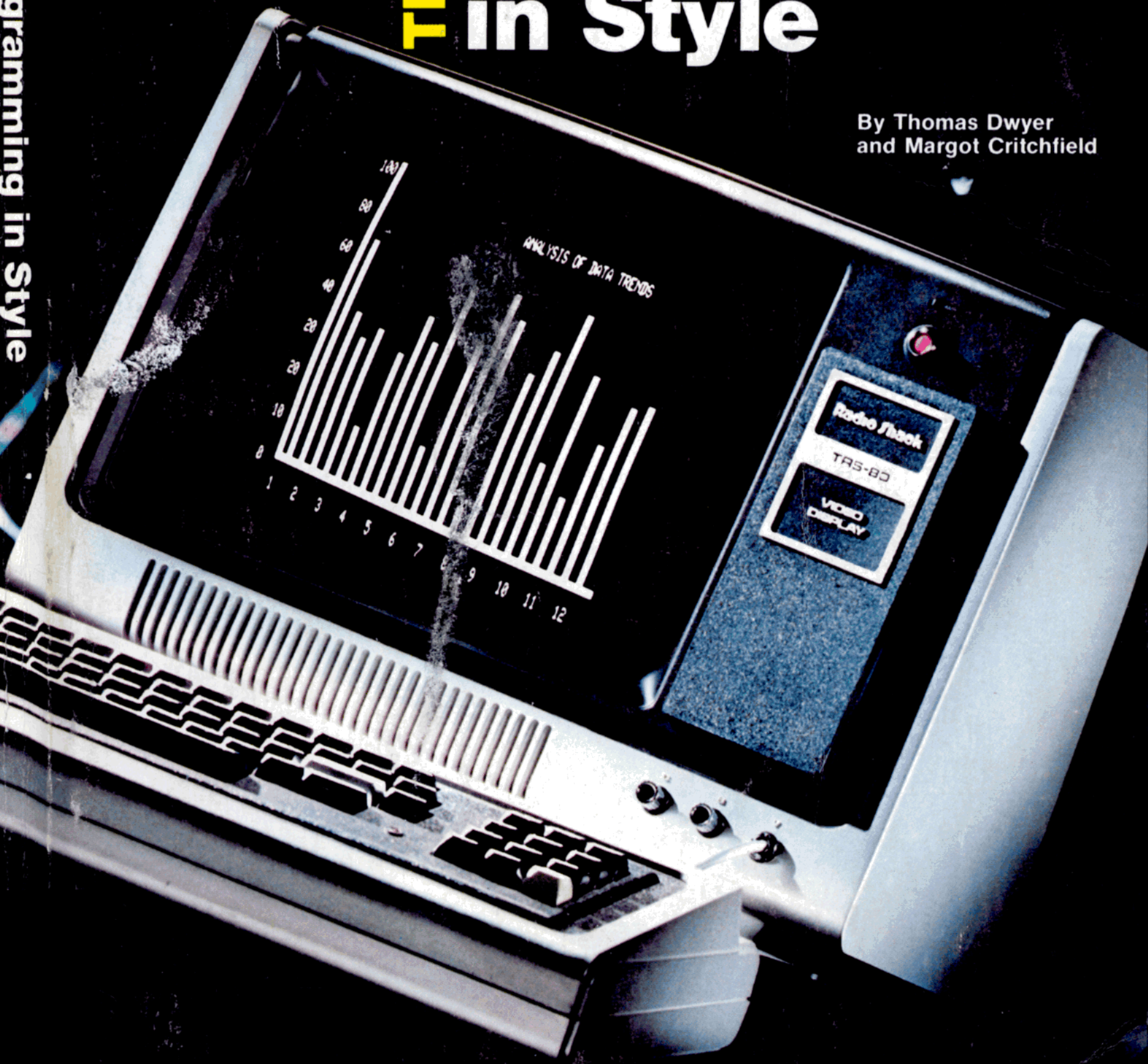
TRS-80 Programming in Style

Radio Shack®

TRS-80™

# Programming in Style

By Thomas Dwyer
and Margot Critchfield




An inventive, idea-oriented approach to creative programming for those who have mastered the fundamentals of Level II BASIC . . . Chock full of programs and programming ideas!

# PROGRAMMING IN SYTLE


**Thomas Dwyer**

**Margot Critchfield**

*University of Pittsburgh*


*Illustrations by*

**Margot Critchfield**


RADIO SHACK
A Division of Tandy Corp.
Ft. Worth, Texas 76102

# PROGRAMMING IN STYLE

The authors of the programs provided with this book have carefully reviewed them to ensure their performance in accordance with the specifications described in the book. Neither the authors nor BYTE Publications Inc, however, make any warranties whatever concerning the programs, and assume no responsibility or liability of any kind for errors in the programs or for the consequences of any such errors. The programs are the sole property of the authors and have been registered with the United States Copyright Office.

Dwyer, Thomas A 1923-
You just bought a personal what?

Includes index.
1. Minicomputers. 2. Microcomputers. 3. Minicomputers--Programming. 4. Microcomputers--Programming.
I. Critchfield, Margot, joint author. II. Title.
QA76.5.D96 001.64'04 80-14410
ISBN 0-07-018492-5

Copyright © 1980 by Thomas Dwyer. All Rights Reserved. No part of this book may be translated or reproduced in any form without the prior written consent of Thomas Dwyer.

ISBN 0-07-018492-5

# CONTENTS

*Preface vii*

**1** LEARN COMPUTER PROGRAMMING AT HOME

1.0 Introduction 3 1.1 Computer Hardware Plus Software: The New Cybernetic Clay. BONJOUR 5 1.2 HYACINTH: The World's Shortest Impressive BASIC program. *GUTENTAG, GOTCHA* 12 1.3 More about BASIC. PRINTPI, FIELDS, SURVEY, *RICHQUIK 19* 1.4 How to Feed Data to a Computer. INPUT, HEIGHT, *CARPET1, CARPET2,* READDATA, GROCERY1, *GROCERY2, (ENERGY)* 23 1.5 Decision Making in Programs; Flowcharts. VOTER, *RETIRE,* ANSWER 29 1.6 Computer Loops and Arrays. DRILL, *SQCUBE,* GOWRONG1, *GOWRONG2, GOWRONG3,* SALEDATA, *ICECREAM,* DOGSALES, *DOGDAYS* 34 1.7 Processing Words. FAMOUS, *SAM, POEM, APRIL1* 43 1.8 Built-in Functions in BASIC. DICE, *CRAPS, TABGRAPH 47* 1.9 Other Features of BASIC. STARS, FANCY, *THROW A PARTY* 51 Exercises 55 Sources of Further Information 57

**1.5** STRUCTURED PROGRAMMING FOR BEGINNERS: INTRODUCING THE FORMAT FOR THE REST OF THE BOOK 59

**2** PUT AN EDUCATIONAL GAMEROOM IN YOUR BASEMENT 65

2.0 Introduction: Computers, Games, and Learning 65 2.1 Simple Number Games. ARITH, *DATARITH, DATARIT2, DATARIT3, DATARIT4, RNDARITH, MULT, VARIETY1, VARIETY2, SQROOT* 67 2.2 Time Out to Learn Some Extended BASIC. LRDEMO, VALDEMO, ASCDEMO, PRUDEMO, BUGPROG, *OKPROG* 76 2.3 Finding Your Way around Number Land. GUESS1, *GUESS2, SRCH1, SRCH2* SRCH3 86 2.4 Word Games. SCRAM1, *SCRAM2, SCRAM3, CIPHER* 96 2.5 Space Games. BABYQ, *BABYZAP* 122 2.6 Games and Graphics. SETPLOT, TRIGPIX, *ARROW2, (ELBLASTO)* 137 Exercises 142 Sources of Further Information 143

3 GETTING SERIOUS: HOME FINANCE AND BUSINESS PROGRAMS 145

3.0 Introduction 145 3.1 What's Possible? The World of Data Processing 147 3.2 Simple Finance Programs. LOAN1, SAVE, LOAN2, AMORT, *FINANCE, FINAN2* 149 3.3 Data Based Business Applications 163 3.4 Small Data Based Programs. ESTIMATE, *(CATERER)* 164 3.5 Screen-Oriented Transactions. SALESLIP, *HARDSALE* 169 3.6 Data Based Financial Reports, MONEY, *MONEY2 (CHECKBAL)* 176 Exercises 189 Sources of Further Information 190

4 HOW TO TAKE THE PLUNGE: UPGRADING TO A BIGGER COMPUTER SYSTEM 193

4.0 Introduction 193 4.1 Why More Equipment? Examples of What's Possible 195 4.2 Adding Hard Copy. MONYLIST, *TAXRPT*, SNAPSHOT 199 4.3 Adding Disk Storage. FILEBOX, FILEDEMO, *FILEBOX2, FILEBOX3, HASHDEMO, (HASHFILE), (MONEYDISK)* 210 4.4 Word Pro cessing and Linked Lists. EDIT1000, *EDIT2000* 230 4.5 Machine Language Anyone? MLDEMO 242 4.6 Should You Invest in a Business Microcomputer? *EDIT5000, (EDIT7000), (TEXTFORM)* Sources of Further Information 250

5 SOFTWARE CITY: SOLUTIONS TO THE PROJECTS 253

APPENDIX A. SUMMARY OF EXTENDED BASIC 319

APPENDIX B. ASCII CODES 337

INDEX 341

# PREFACE

The new low-cost microcomputers for homes, schools, and businesses are shaking up our ideas about computing. They are also a lot of fun.

This is a book for anyone who would like to know more about what these machines can do, and how to make them do it. It's a collection of practical ideas for using a personal computer at home or at work, together with information on how to develop innovative uses of your own.

The key word here is *innovative*. The real magic of computing is that even a beginner can improve on the work of others. Our goal is to show you how, using a structured approach to developing creative applications. "Structured" means based on a discipline that will steer you around most of the pitfalls of do-it-yourself programming. "Creative" means going beyond what's been done, flavoring all your ideas with a sprinkling of imagination.

We'd also like to convince you that while the road to imaginative computing can be a challenging one, it's less difficult to navigate than many imagine. People of all ages and backgrounds are proving this every day. In different ways and at different levels, of course, but that's what makes the whole thing so interesting.

In particular, there are two flexible options available to any beginning computer user. The first (we'll call it plan A) is for the person who wants to put a computer to work right away, building on application ideas and techniques developed by someone else. Plan A means you not only buy a computer, but you also buy the "intelligence" it needs to operate. You do this by purchasing pre-written computer programs. These are sets of instructions for controlling the actions of the machine. They come in printed form (like this book), or on magnetic tapes and disks. Computer programs are often referred to collectively as "computer software." Using computer software prepared by someone else is something like using prerecorded music on your hi-fi: easy, pleasurable, and educational, but not directly connected with the creative side of things.

The second option (plan B) is to "learn computer programming at home." The idea is to find out how to write your own software. This takes more patience and perseverance, and (like learning to make your own

music) the results may not be very professional at first. Yet those who choose this option and succeed even mildly claim that it's "like nothing I've ever experienced—totally fascinating."

This book is for personal computer users interested in either plan. To make it a plan A book, you can copy any of the programs we show in the text (or answer section) into your computer "as is," and have fun using it. Most of the programs have been written in Microsoft extended BASIC, using a Radio Shack TRS-80 computer. However, we've also included information showing how to translate some of the extended features of this BASIC into "minimal" BASIC.

To follow plan B, you'll want to use the book's programs, text, and self-study projects as stepping stones to writing your own programs. This approach will take more time, and only you will know if it was worth the effort. But one thing is certain: Even if you decide to go back to plan A, buying most of your programs from others, your experience with programming will have made you a more discerning software consumer.

The real payoff to mastering a personal computer, whether done with your own or someone else's programs, is the sense of intellectual satisfaction you'll eventually experience. This isn't easy to describe, but when it finally all comes together, you'll know. It will probably happen on the same day you decide that investing in a personal computer was a pretty good idea after all.

**NOTE ON THE PROGRAMS IN THIS BOOK**

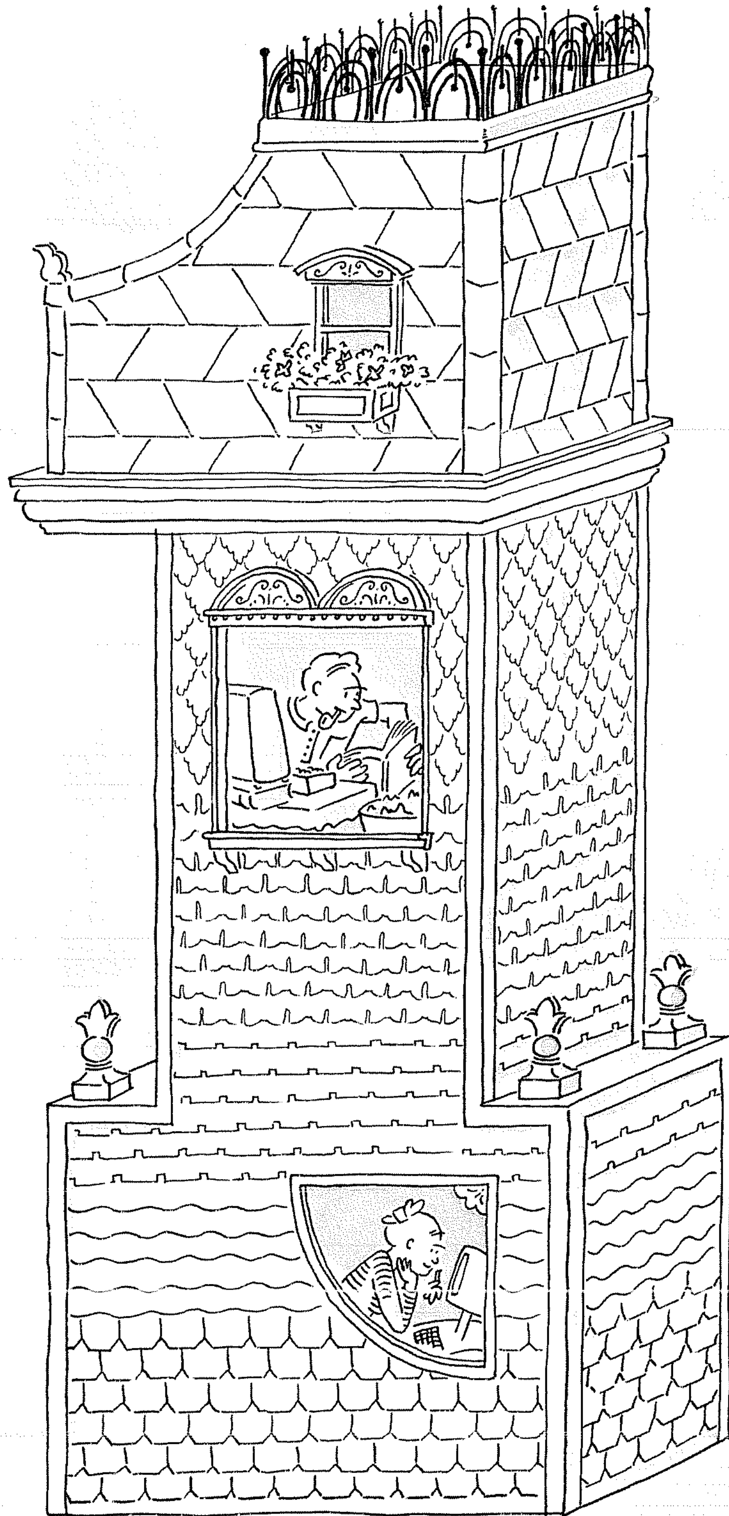
Each of the working programs we'll present has been given a unique "file name." File names printed in capital letters (for example DICE, GUESS1, or SALESLIP) refer to computer programs that appear in the explanatory part of the book (Chapters 1-4). The techniques used to design as well as write these programs are also explained. Readers are encouraged to try their hands at using similar techniques to design their own programs.

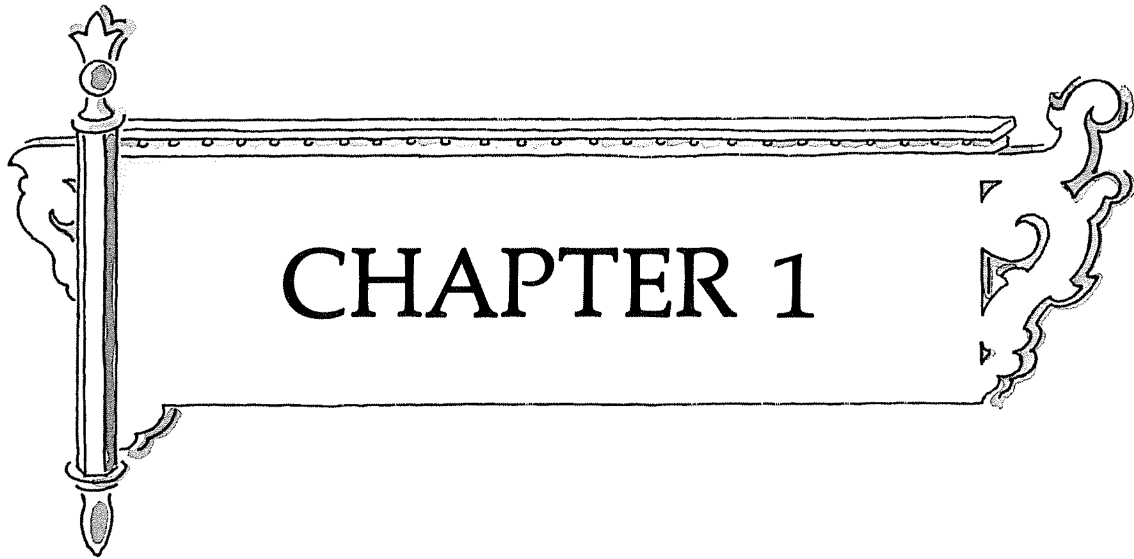
In addition, there are a number of program ideas first presented as self-study projects. These are indicated by names in italicized capital letters (for example *RNDARITH*, *SCRAM3*, or *MONEY2*). These programs are left incomplete in the front of the book so you can practice filling in the details. However, sample solutions in the form of complete listings are given in Chapter 5 (Software City).

For readers who want to go beyond the projects, we've included a number of superprojects. Solutions for these are not given in the book, so they would make good course projects in schools. Superprojects are indicated by italicized capital letters in parentheses—for example *(CATERER)*, or *(EDIT7000)*.

The programs and project solutions for the first three chapters were written in TRS-80 (Microsoft) Level II BASIC. Chapter 4 programs and project solutions were written in the disk-extended form of this BASIC. Information on converting some of the features of Level II to minimal BASIC can be found in section 2.2.

# PROGRAMMING
# IN STYLE






# LEARN COMPUTER PROGRAMMING AT HOME

## 1.0 INTRODUCTION

The start of the computer age is usually given as 1946, the year the pioneering ENIAC digital computer produced its first useful results. If the scientists of that day had predicted that within one generation there would be hundreds of thousands of far more powerful machines in the homes of ordinary people, they would have been labeled as impossible dreamers. If only ten years ago it had been claimed that several million copies of books on the technical aspects of computing would find their way into these same homes, most publishers would have dismissed the number as totally unrealistic.

That such predictions would have been right on target is now a matter of record. The wide, wild and woolly world of personal computing is expanding rapidly, and where it will stop nobody knows.

If you have just joined the festivities, or even if you are only thinking about it, one of the questions you are undoubtedly asking is "What's personal computing *really* like?"

Part of the answer is that it's lots of fun. But there are also overtones of mystery and magic, with dozens of unknowns to be fearlessly faced.

Not the least of these is an avalanche of new jargon that sounds like dialogue from the space-saloon scene in a science fiction movie.

The obvious follow-up question to ask is "How much of this technical double-talk do I have to unravel before I can buy and use a computer intelligently?" The answer is "Surprisingly little." Let's see why.

The mysteries of computing can be lumped into two big categories called hardware and software. In this chapter we'll start by briefly describing the hardware part. We'll then switch over to the software area, and spend most of our time there. Our game plan will be something like that of the flight instructor who tells you a little about the mechanical side of your airplane (the hardware), but who then spends most of the days ahead showing you how to fly it. Later on, after you have a good feel for what your machine can do, you can return to a more profitable study of the hardware and how it works.

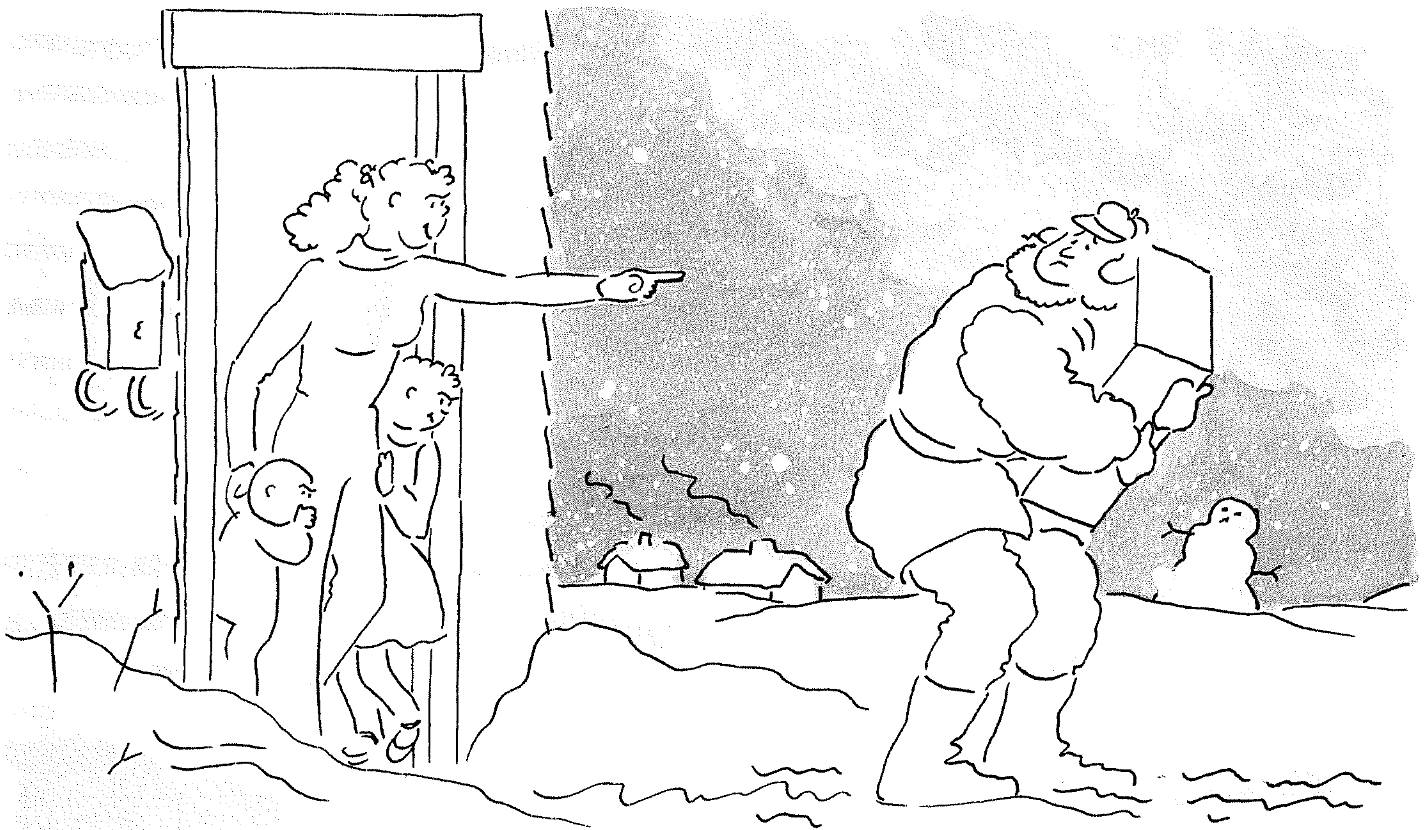
Learning "how to fly" in the world of computing translates into learning how to write good programs. This can be an expensive skill to acquire, as the thousands of people who enroll in computer programming schools each year can testify. A personal computer makes an excellent tool for those who'd rather learn about programming on their own. The bonus is that the hardware you buy to learn programming has many other applications. Chapters 2, 3 and 4 will look at some of these, while continuing to teach the principles of good programming by example.

The best way to learn computer programming is to write programs—lots of them. The programs in the first half of this chapter are short enough to be tried in one sitting. Even if you've written programs before, it will be worth trying them as warm-up exercises. Things get more challenging after section 1.5 and that's when you can expect to burn more midnight oil than usual. When you get to the end of this chapter, you will have covered about as much material as is given in a first college course. The last project *(Throw A Party)* will probably seem like a good idea by then.

## 1.1 COMPUTER HARDWARE PLUS SOFTWARE: THE NEW CYBERNETIC CLAY

Let's assume that with the help of friends, a knowledgeable salesperson, and lots of reading, you've finally selected a computer. The big question now is what will it do after you bring it home and set it up.

The answer is "Nothing special." Of course you knew this, and before the rest of the family has a chance to send your computer, (and possibly you) back for a refund, you intend adding the software needed to make things start happening.



*Software* is a word that refers in a general way to sets of computer instructions called *computer programs.* Without programs, computer hardware can be thought of as a kind of "cybernetic clay," waiting to be molded into any one of countless shapes. It's the programmer who makes this happen, and it's with good reason that the process is often called "the art of computer programming."

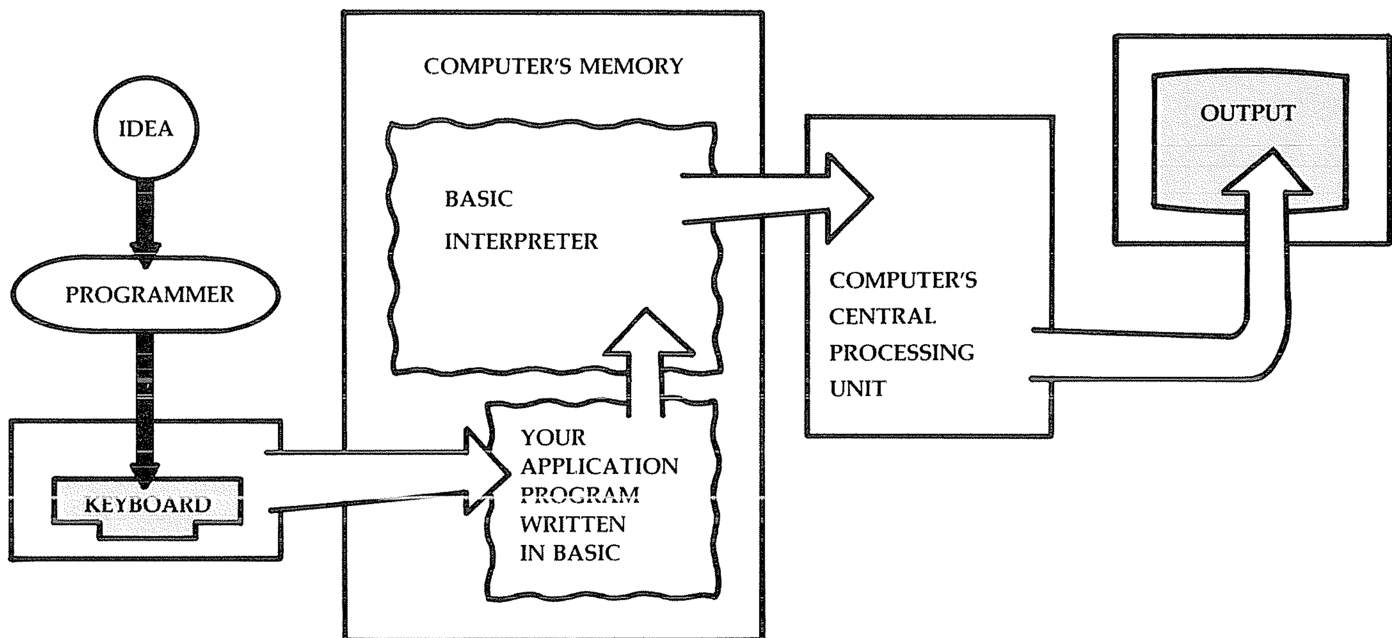
Software is a coined word, so you won't find any Latin or Greek roots for it in the dictionary. It was invented by computer people to mean "the complement of hardware." This explains why the spelling "softwear" — which has appeared in print — is incorrect; there's no connection between computer programs and cashmere sweaters.

Software is an intriguing thing. You can't touch it, or bolt it to the computer mainframe the way you can electronic circuit cards. Engineers can electronically store software in silicon chips, at which time it's often called *firmware*. But such chips are just particular realizations of a much more general concept. The essence of software is that it represents ideas of far greater complexity than found in any one physical device. It's a product of our minds much more than our hands.

Now for a remarkable surprise. Despite the fact that software is a somewhat abstract, intangible thing, the majority of beginners can learn to create and use it. This is not true of hardware. The reason is that one can learn to write programs by trying things. Nothing will be hurt by experimentation. If you make a wrong connection in hardware, however, there's a good chance that some expensive parts will go up in smoke.

## Kinds of Software

Computer programs fall into one of two general categories: system software and application software. *System software* is usually supplied with the computer, and it consists of computer programs that help you use the machine itself. The most valuable system program supplied with microcomputers today is what's called a *BASIC interpreter.* This is a program that helps you write other programs called application software.




The term *application software* refers to those programs that help you use the computer for specific tasks. For example, business application programs are really sets of instructions that show the computer how to keep inventory records, update accounts receivable, print payroll checks, and

so on. For schools, application programs might focus on helping students handle problems in science and math, or on exploring ideas that go beyond textbook problems. For a scientist, application programs include simulations, gathering of lab data, analysis of experiments, modeling, and complex computations. Actually, all the programs in this book are application programs. Some are fun, some are serious. As you'll see, the range is almost limitless.

## An Example

Let's do a run-through of what it's like to actually use a personal computer. The hardware we'll use is shown in the photo. There are many other makes of computers that work in a similar manner. If you're trying any of these ideas on a different machine, you'll want to use the reference manual that comes with it to clarify any differences in operating procedures.

*The Radio Shack TRS-80 model I computer consists of a keyboard unit that also contains the computer's memory and processing circuits, a video display for showing output, and a cassette tape recorder for saving programs. An extended BASIC interpreter is stored in "read only memory" (also inside the keyboard case), and it's ready for use as soon as the machine is turned on.*

As the photo shows, our computer has a keyboard for input, a TV-like screen for output, and a cassette tape machine for storage. What you

can't see in the photo are the electronic memory and processing circuits which are on compact circuit boards inside the computer case.

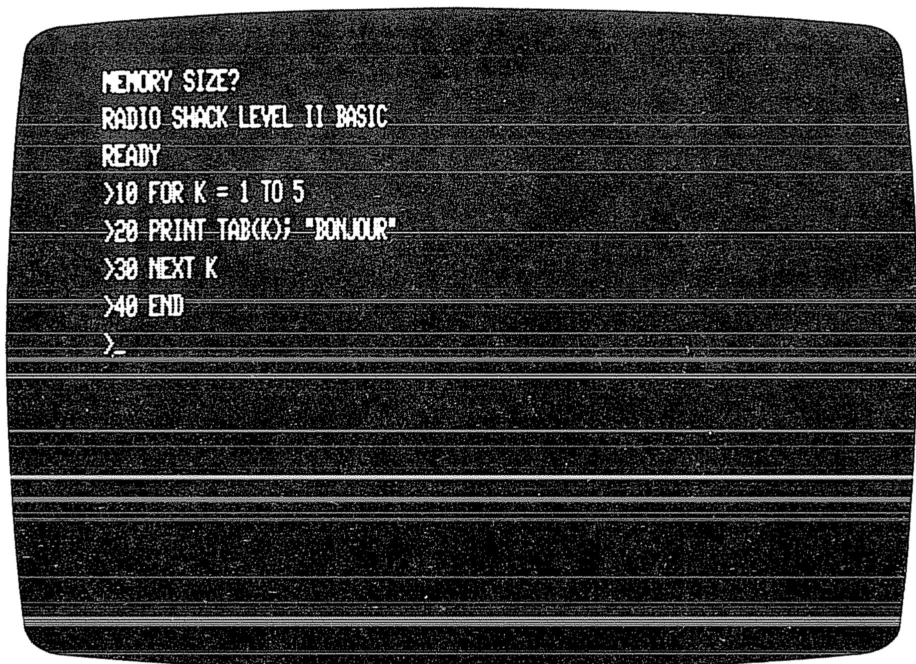
To use a personal computer for a given purpose—say, to play a game or balance a checking account—there are four steps to follow:

1. Get the machine ready.
2. Feed in the instructions that explain what you want done. This is called *entering* or *loading* the program.
3. Command the computer to carry out these instructions and give you whatever output (answers) are expected.
4. Shut the machine down.

For most machines, step 1 is really easy. All you do is put the plug in the wall and push a few buttons to "on." After a few seconds warm-up the word READY, or a similar prompting message, will appear on the screen. This means the machine is ready to accept instructions, so go on to step 2.

You can enter instructions into the computer in one of two ways. You can either *load* them from a tape cassette (or disk) on which they were previously recorded, or you can type them in at the keyboard. Let's look at an example of step 2 where the instructions are typed in from the keyboard.

We'll type in a program that has four lines of instructions. Each line is called a *statement*, and it's written with a kind of code called a computer language. The most common computer language used on personal computers is called BASIC (short for Beginner's All-purpose Symbolic Instruction Code).




*Here's what the user sees after typing in the four BASIC statements labeled 10, 20, 30, and 40. These make up a BASIC program which has been stored in the computer's memory, but which has not yet been executed (run).*

Notice in the above photo that the machine prompts you for each new line with the symbol >. You type everything after this symbol. You also press the ENTER key (on other machines this may be called something else, such as RETURN) at the end of each line to get a new prompting symbol on the next line.

We won't explain exactly what these four lines of BASIC mean (that's coming). However, you can get a pretty good idea of what this program tells the computer to do by going to the next step and running it.

Step 3 is easy. To run, or execute, a program, you simply type RUN, and press the ENTER key. Watch what happens:

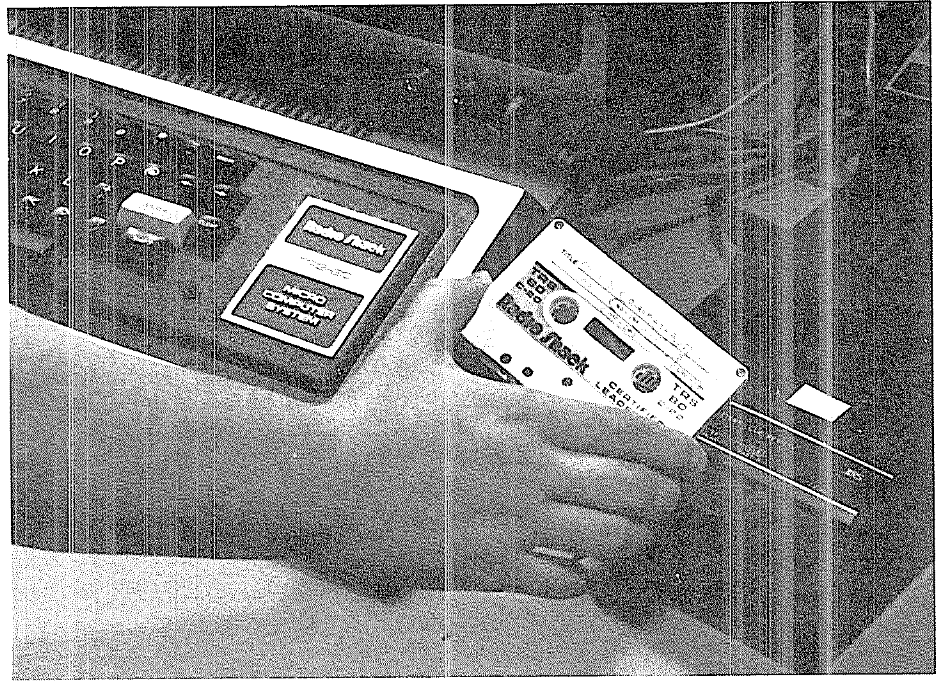
```
MEMORY SIZE?
RADIO SHACK LEVEL II BASIC
READY
>10 FOR K = 1 TO 5
>20 PRINT TAB(K); "BONJOUR"
>30 NEXT K
>40 END
>RUN
 BONJOUR
  BONJOUR
   BONJOUR
    BONJOUR
     BONJOUR
READY
>_
```

*After typing in the BASIC program, the programmer next typed RUN, and pressed the ENTER key. The machine immediately executed (carried out) the BASIC instructions. In this example, the instructions were to print the word "BONJOUR" five times, moving it over ("tabbing" it) 1 space the first time, 2 spaces the second time, and so on up to 5 spaces for the fifth time.*

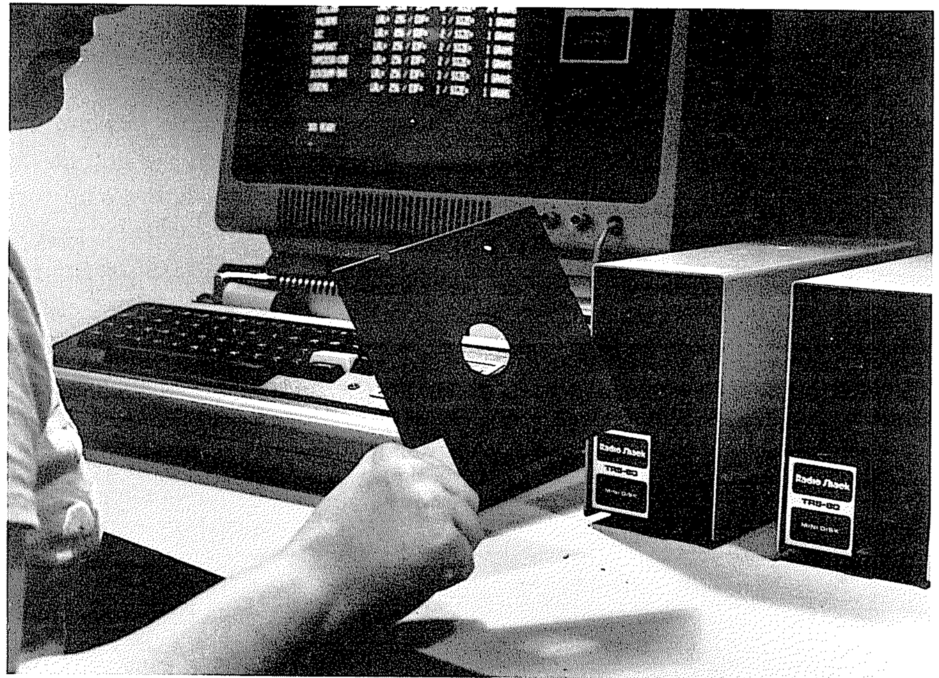
As you can see, our program told the computer to print the phrase "BONJOUR" five times, but to TAB (move horizontally) over a little bit each time. It's a silly program, but it illustrates the technique for entering and running all BASIC programs, no matter how complicated.

The fourth step is also easy, at least if you don't want to save your program. You simply turn all the units off. However, this will cause your program to disappear. If you want to save it for future use you must first copy it onto tape or disk. To do this on most machines, you first put a blank tape in the cassette machine. Then you type CSAVE, meaning

cassette save, start the tape machine recording, and press ENTER. When the computer prints READY, or an equivalent statement, you'll know your program is saved, and you can go ahead with turning everything off. The next time you want to use this program, it can be read from the tape by typing the command CLOAD, starting the tape machine, and then pressing ENTER.




*BASIC programs can be saved by recording them on tape cassettes. The programs can then be "loaded" directly from the tape at a later date with no need to re-type the BASIC statements. Tapes should be kept away from magnetic fields (including those around power supplies, TV sets, power cords, and motors).*



*Magnetic disks can also be used to save programs and other information. The minidisk shown here is actually a circular piece of flexible (floppy) plastic inside the 5¼ inch square jacket you see. After it's placed in the disk drive it will rotate at 300 revolutions per minute, allowing for rapid saving and loading of programs and data. Chapter 4 discusses other uses of these disks, including "data-based" business applications.*

Since various makes of computers use slightly different procedures, you'll want to read the instruction manual for the machine you're using before doing any of the above. However, you'll find that the procedures on most machines are similar, and that after a few experiments the whole process will become second nature.

## 1.2 HYACINTH: THE WORLD'S SHORTEST IMPRESSIVE BASIC PROGRAM. *GUTENTAG, GOTCHA*

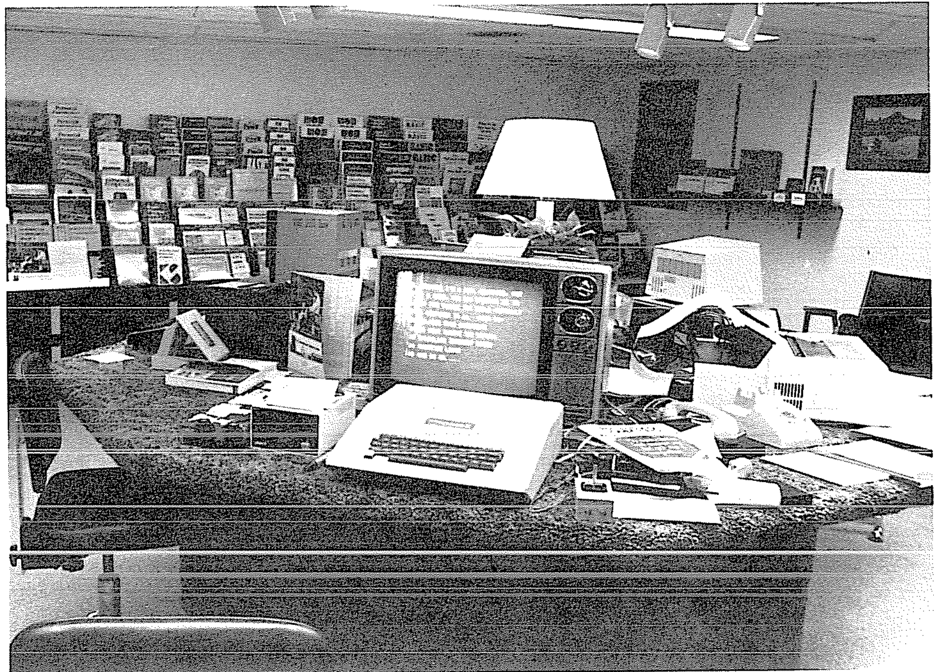
Let's suppose you're visiting a computer store with a friend named Hyacinth. You find a machine that's guaranteed to understand programs written in BASIC, and it's all turned on and ready to go. You ask if you can try a BASIC program, and the friendly salesperson says, "Sure."

The first thing you want to do is type NEW, followed by pressing ENTER. This erases any old program that may have been in the computer's memory. You should then see something like the word READY or OK, and a readiness symbol like >. This means it's your turn and you can type in some BASIC statements. For example, you might type:

```
>10 PRINT "HELLO"
>20 PRINT "FRIEND"
```

Don't forget to push ENTER at the end of each line. You have just entered a BASIC program with two statements. To see it work, you next type the command RUN, and press ENTER. Here's what should happen:

```
> RUN
HELLO
FRIEND
```



*Personal computers can be purchased through retail stores where technical assistance is available. This photo was taken at the Computer House, one of the many independent computer stores around the country.*

O.K. so far? Now let's impress your friend Hyacinth with a trickier program. First type NEW (this is going to be a new program), and then type the following (don't forget to press ENTER at the end of each line):

```
>NEW
>10 PRINT "HYACINTH!!!";
>20 GO TO 10
>30 END
```

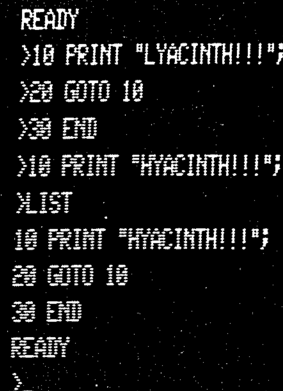
Hold it! Suppose you made a mistake and typed

```
10 PRINT "LYACINTH!!!";
```

No problem. Just retype the bad line.

```
10 PRINT "HYACINTH!!!";
```

The rest of the lines are still in the computer. If you want to make sure you have it all O.K., just type the command LIST. Here's what you'll see:




*Everything after each symbol > was typed by the programmer. Notice that line 10 was re-typed (after line 30) to correct a mistake. Then LIST was typed (followed by pressing ENTER) to verify which statements were currently in the computer's memory.*

The command LIST told the computer to print (or list) out all the BASIC statements it currently has in memory. Notice that commands do not have a line number.

Both commands and statements must always be followed by the pressing of the ENTER key. This does not show on the screen, but nothing will be entered into the computer until you press this key. Also, if an error message shows up, this means that the offending line was not entered. Any time you're not sure of what has been entered, just type the command LIST.

*This is a standard computer keyboard as found on a professional machine. The key at the right labelled CAR RET means "carriage return," and it should be pressed at the end of every line the user types.*

*This is the keyboard on the TRS-80 model I computer. The key marked ENTER should be pressed at the end of every line the user types.*

So far you've entered a three-line program, but the computer has not *executed* (carried out) your instructions. To make it do this, you simply type the command RUN:

```
>RUN
HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!
!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINT
H!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACI
NTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYA
CINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!H
YACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!
!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH
!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACIN
TH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYAC
INTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HY
ACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!
HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!
!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINT
H!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACI
NTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HY
```

*This is what the screen will look like when the program HYACINTH is run. Actually, the pattern you see will "scroll" upward since the program continues to print new lines at the bottom of the screen until you press BREAK.*

No you haven't broken the computer. This is a silly program deliberately chosen to give you lots of action from only two statements. The first statement (line 10) says to print the message HYACINTH!!! on the screen. The computer does this, and then goes on to execute the next statement in line 20. But this says go back to line 10. So the computer obediently does this, and prints HYACINTH!!! again. It's printed right after the first message because line 10 ended with a semicolon (;). If the semicolon had not been there, the message would have printed on a new line.

Do you see what's happening? Our program contains what's called an infinite loop, and it will go on forever printing HYACINTH!!!, branching back (GO TO) to line 10 and so on. The program will never get to line 30 (END).

To stop such a program, a special key must be pressed. On some computers it's labeled BREAK. On others, you must press both the control key (CTRL) and the C key simultaneously. This is called a "control-C".

```
ACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!
HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!
!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINT
H!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACI
NTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYA
CINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!H
YACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!
!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH
!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACIN
TH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYAC
INTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HY
ACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!HYACINTH!!!
HYACINTH!!!
BREAK IN 20
READY
>_
```

*When you press BREAK the execution of the program is interrupted, and a message telling you which statement was being executed at that time is printed.*

## Review

Here's what we've learned so far:

1. You write BASIC programs by typing in BASIC statements. These must have line numbers, and you must press ENTER after you type each line.
2. To make things happen, you type BASIC commands, again followed by pressing ENTER. These don't have line numbers. The commands we've seen so far are NEW, RUN, LIST, CSAVE, and CLOAD. The last two commands are written as SAVE and LOAD on some machines.
3. To make corrections in BASIC statements, retype the entire line followed by pressing ENTER.
4. Most programmers number BASIC statements 10, 20, 30, and so on. This way, if they forget a line it can be added at the bottom as in the following:

```
> 10 PRINT "STAND BY"
> 20 PRINT "BLAST OFF"
> 15 PRINT "FOR"
> RUN
STAND BY
FOR
BLASTOFF
```

   The computer inserted statement 15 in its proper place between statements 10 and 20. If you don't believe it, type LIST and see.
5. BASIC statements have to use certain keywords, put together in just the right way. Some keywords we've seen so far are PRINT, GOTO, and END. We've also used the function TAB (see section 1.8).

## Project *GUTENTAG* (Note: Chapter 5 contains solutions to all projects.)

1. Type the program from section 1.1 into a computer and run it. Then retype line 20 with BONJOUR changed to GUTEN TAG and run it.
2. Now see if you can use TAB(X) to make the message GUTEN TAG move over two spaces each line. (*Hint:* In BASIC, 2*K means multiply K by 2. So for K = 1, 2, 3, ..., etc., 2*K = 2, 4, 6, ..., etc.)

## Project *GOTCHA*

The program to print HYACINTH looks best on a screen that uses 64 columns to display output. Some computers can use a different number of columns, and on these machines names with different numbers of characters create better patterns. For example, the TRS-80 computer uses

a double-sized character set when you push the shift and right arrow (→)keys, giving the effect of a 32-column screen. In a program you can get the same effect by using the statement PRINT CHR$(23). To go back to normal characters, use the statement CLS, meaning clear screen, or press the CLEAR key.

Write a program that creates an interesting pattern of names using double-sized characters. The photograph shows one possibility. You should ask why we used 11 characters for the message "* GOTCHA! *". What other numbers or patterns would look good?

*This is output of the type suggested for project GOTCHA. It's similar to HYACINTH, but modified to look best in a 32-column format.*

## 1.3 MORE ABOUT BASIC: PRINTPI, FIELDS, SURVEY, *RICHQUIK*

Let's now go on to a more serious (but not quite as flashy) program. We'll use PRINT again, but this time for printing both messages and the results of calculations. We'll also use the new keyword LET.

First, let's examine some of the things that can be included in PRINT statements with a program that does a simple calculation:

*This is a (REMARK) statement. Remark statements are used to explain what programs do. The computer ignores remark statements during a run.*

```
5  REM---PRINTPI---
10 PRINT "APPROXIMATION TO PI"
20 PRINT "355/113 = ";355/113,"FINISHED"
30 END
```

*These two statements produce these two lines of output*

```
>RUN
APPROXIMATION TO PI
355/113 =  3.14159                 FINISHED
```

1. The first thing you can see from this example is that anything inside quotation marks is printed exactly as you gave it.
2. When you don't use quotation marks, the computer will first do the arithmetic called for and then print the answer. In our example "355/113 = " was printed as is, but 355/133 was first evaluated (using division), and then the result was printed as 3.14159.
3. When you want several items of output printed on the same line, you use either a semicolon or a comma between them in the PRINT statement. The semicolon forces answers to print close together, while the comma spreads the answers out. The amount of spreading depends on the computer. Semicolons usually force things together as closely as possible, leaving one space on either side of numbers (but none for letters). Commas usually force the output to fall into fields 14 to 16 spaces wide. The following program can be used to see what the spacing is on your computer.

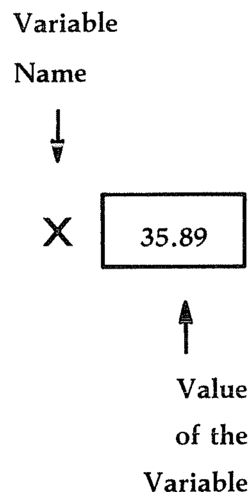
```
7 REM---FIELDS---
8 PRINT"SHOWS THE SPACING CAUSED BY ',' AND ';'"
9 PRINT"    5    10    15    20    25    30    35    40    45    50"
10 PRINT"----I----I----I----I----I----I----I----I----I----I"
20 PRINT "1";"2";"3";"4"
30 PRINT 1; 2; 3; 4
40 PRINT "1","2","3","4"
50 PRINT 1, 2, 3, 4
60 END
```

*Lines 20 and 40 print characters but lines 30 and 50 print numbers, leaving room for + or − signs*

```
>RUN
SHOWS THE SPACING CAUSED BY ',' AND ';'
    5    10    15    20    25    30    35    40    45    50
----I----I----I----I----I----I----I----I----I----I
1234
 1  2  3  4
1               2               3               4
 1               2               3               4
```

So far we've only printed the results of calculations that use fixed numbers (such as 113 and 355). These are called *constants.* Things get more interesting in BASIC when both constants and *variables* are used. As the name says, variables represent data that can change.

Variables are possible because microcomputers store the data (numbers and/or characters) used by programs in erasable locations in their memory. When you use BASIC, you can refer to each of these locations with what's called a *variable name.* This consists of a single letter, like A, B, or X, or a single letter followed by a single digit, like A6, B8, or X3, as shown below:



The LET statement

10 LET X = 35.89

does two things. It associates the name X with one of the computer's memory locations, and then stores the number 35.89 in that location. When you want to find out what number is stored in a location, you use a statement like this:

20 PRINT X

Notice that this doesn't mean print the letter X; it means print the number that is stored in the location named X. To erase the number stored in X and replace it, for example with 2.5, you use a statement like this:

90 LET X = 2.5

Here's a program to type in and run which uses the LET statement. This program calculates the percent of people in a survey who were Republicans and the percent who were Democrats. Recall that if 5 out of 20 people do something, the percent will be (5/20)*100, or 25%. Before typing in this program, don't forget to erase the previous program by typing NEW.

Notice that arithmetic can be done in LET statements

And in PRINT statements

```
5 REM---SURVEY  (USES LET STATEMENT)---
10 LET D = 4832
20 LET R = 3741
30 LET T = R + D
40 PRINT "TOTAL VOTES = "; T
50 PRINT "PERCENT DEMOCRATS"; (D/T)*100; "%"
60 PRINT "PERCENT REPUBLICANS"; (R/T)*100; "%"
70 END

>RUN
TOTAL VOTES =  8573
PERCENT DEMOCRATS 56.363 %
PERCENT REPUBLICANS 43.637 %
```

We used parentheses in lines 50 and 60 to make sure that the computer grouped the proper numbers together. For example:

10/(2 + 3) means 10/5 = 2 but

10/2 + 3 means 5 + 3 = 8

Note that the symbols for addition, subtraction, multiplication, and division in BASIC are +, −, *, and /, respectively. Another symbol used is the up-arrow to mean exponentiation. Writing 2↑5 means raise 2 to the 5th power, which is equivalent to calculating 2 * 2 * 2 * 2 * 2.

### Project *RICHQUIK*

Write a program which when run shows how much money would be paid on days 1, 2, 3, 4, 5, 10, 20, and 30 at a job where the contract specifies $2 for the first day, with the salary doubled each succeeding day. Here's what the output could look like:

```
>RUN
DAY 1 =  2
DAY 2 =  4
DAY 3 =  8
DAY 4 =  16
DAY 5 =  32
DAY 10 =  1024
DAY 20 =  1.04858E+06
DAY 30 =  1.07374E+09
```

*This means 1,048,580*

*This means 1,073,740,000*

*Hint:* For day 3, the following statement works

50 PRINT "DAY 3 = $";2 * 2 * 2

but so does

50 PRINT "DAY 3 = $";2↑3

So, for day 5 you can use

60 PRINT "DAY 5 = $";2↑5

and so on.

If you haven't seen the notation E+06 before, don't be surprised. It's a computer shorthand which means move the decimal point 6 places to the right, while E-06 is shorthand for move the decimal point 6 places to the left. The shorthand was devised to save space when printing very large or very small numbers.

In our sample output, the numbers 1.04858E+06 and 1.07374E+09 are therefore shorthand for 1,048,580 and 1,073,740,000, respectively. Mathematically, E+06 means times 10 to the 6th power, and E+09 means times 10 to the 9th power. If you get a number like 1.04858E-06 it means 1.04858 times 10 to the −6th power (which is .00000104858).

## 1.4 HOW TO FEED DATA TO A COMPUTER: INPUT, HEIGHT, *CARPET1*, *CARPET2*, READDATA, GROCERY1, GROCERY2, (*ENERGY*)

Suppose you want to write a program that calculates the area of a square. You'll have to tell it how big one side is. This is called *inputting data* to the program. There are several ways to do this. The one we'll examine first uses what is called the INPUT statement. Here's how it works:

```
10 REM---INPUT  (USES INPUT STATEMENT)---
20 PRINT "WHAT IS ONE SIDE OF THE SQUARE IN INCHES"
30 INPUT S
40 PRINT "AREA OF SQUARE = ";S*S;" SQ. IN."
50 GOTO 20
60 END
```

*This statement makes the program print a ? and then wait until you type in a value for S, followed by pressing* return *or* enter

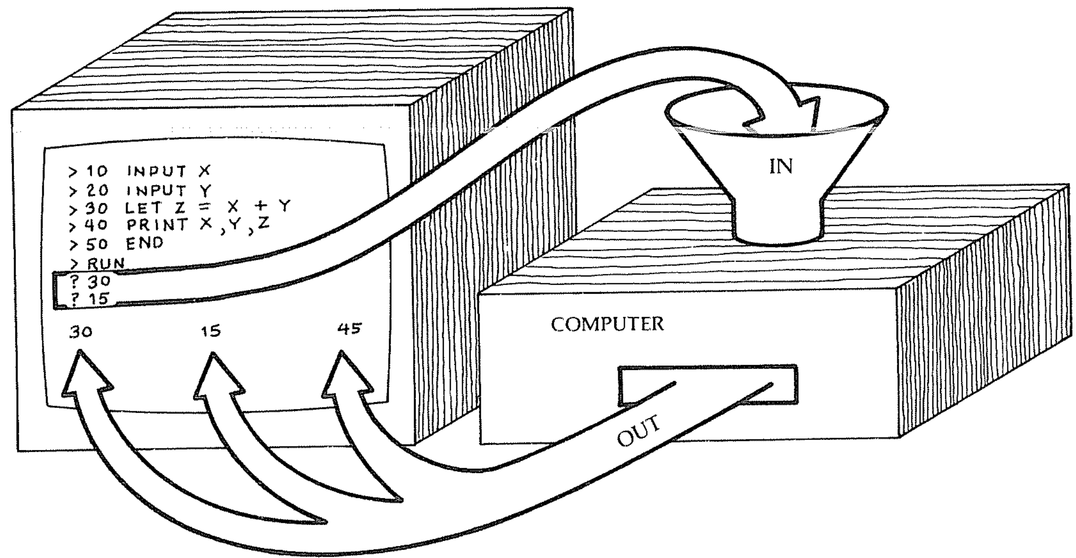
```
>RUN
WHAT IS ONE SIDE OF THE SQUARE IN INCHES
? 1
AREA OF SQUARE =  1  SQ. IN.
WHAT IS ONE SIDE OF THE SQUARE IN INCHES
? 25
AREA OF SQUARE =  625  SQ. IN.
WHAT IS ONE SIDE OF THE SQUARE IN INCHES
? 36456
AREA OF SQUARE =  1.32904E+09  SQ. IN.
WHAT IS ONE SIDE OF THE SQUARE IN INCHES
?
BREAK IN 30
READY
```

*This is shorthand for 1,329,040,000 sq inches*

*The program was stopped by pushing the break key after the last?*

Line 10 is called a REMARK statement. It's ignored by the computer. It's used to help explain programs to people. So the first line actually executed by the computer is line 20.

Executing line 20 causes a prompting message to be printed. It asks you to type in the length of one side of your square. When line 30 is executed by the computer, the program prints a question mark, and then stops. It is now waiting for you to type in a number after the question mark, and press the ENTER key. As soon as you do this, the number you typed is stored in the variable S, and the program continues to line 40 where it prints the correct area. Line 50 makes the program repeat, asking for another S, and so on. To stop this program, press BREAK (or CONTROL-C).




Suppose now you want to write a program that gives you the height of someone in centimeters if you supply the height in feet and inches. This time you'll have to input two numbers (one for feet, one for inches). Here's how it's done:

```
10 REM---HEIGHT  (USES 2 INPUT VARIABLES)---
20 PRINT "TYPE YOUR HEIGHT AS FT., IN."
30 INPUT F, I
40 PRINT "YOU ARE";(12*F+I)*2.54;" CM. TALL"
50 END

>RUN <----------------
TYPE YOUR HEIGHT AS FT., IN.
? 4,11
YOU ARE 149.86  CM. TALL
READY
>RUN <----------------
TYPE YOUR HEIGHT AS FT., IN.
? 6,1.5
YOU ARE 186.69  CM. TALL
READY
```

*Notice that once a program is in the computer's memory you can run it as often as you wish*

The INPUT statement causes the computer to print a question mark (?) and then wait for you to type two numbers. It expects these two numbers to be separated by a comma. When you type these numbers, followed by pressing the ENTER key, your numbers are stored in the variables we called F and I in the program. The program then proceeds in the sequence of the remaining line numbers, giving your height in both inches and centimeters.

### Project *CARPET1*

Write a program that asks for the dimensions (in feet) of a bedroom, living room, and den, and then prints the total number of square feet of carpet needed. A run might look like this:

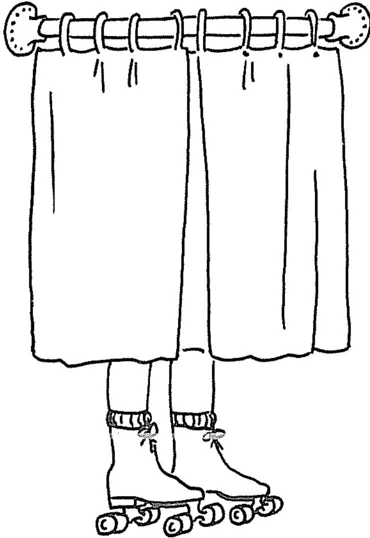
```
>RUN
TYPE LENGTH, WIDTH (IN FT.) OF
BEDROOM:  ? 20,15
LIVING ROOM:  ? 25,18
DEN:  ? 15,21
TOTAL CARPET NEEDED =  1065  SQ. FT.
```

### Project *CARPET2*

Write a program that does the same as CARPET1, and also prints the number of square yards of carpet needed, as well as the total cost. (*Note:* You'll have to add an INPUT statement that requests cost per square yard.) Here's a sample run:

```
>RUN
TYPE LENGTH, WIDTH (IN FT.) OF
BEDROOM:  ? 20,15
LIVING ROOM:  ? 25,18
DEN:  ? 15,21
TOTAL CARPET NEEDED =  1065  SQ. FT.
WHAT IS COST PER SQ. YD. ? 13.95
TOTAL CARPET NEEDED =  118.333  SQ. YDS.
TOTAL COST =  $ 1650.75
```

### Using READ and DATA Statements



So far we've seen that a BASIC program can store data in a computer's memory in two ways. For example, to put the numbers of Republican, Democratic, and other voters in the variables R, D, and X, one method is to use LET statements as follows:

```
10 LET R = 4568
20 LET D = 5132
30 LET X = 489
```

The second method is to use INPUT statement(s) which request the data during each RUN of the program:

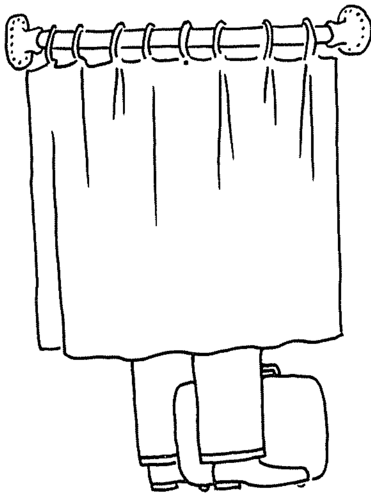
```
10 PRINT "TYPE # OF REPUBLICANS, DEMOCRATS, AND 'OTHER' "
20 INPUT R, D, X
30 PRINT "TOTAL VOTERS = "; R + D + X
RUN
TYPE # OF REPUBLICANS, DEMOCRATS, AND 'OTHER'
?4568, 5132, 489
TOTAL VOTERS = 10189
```

A third approach is to use the READ and DATA statements as illustrated in the following program:

```
5 REM---READDATA  (USES READ, DATA STS.)---
7 PRINT "** COMPARISON OF SURVEYS **"
10 READ R, D, X
20 LET T = R + D + X
30 PRINT "TOTAL VOTERS = ";T
40 PRINT "PERCENT REPUBLICANS = ";R/T * 100;"%"
50 PRINT "PERCENT DEMOCRATS   = ";D/T * 100;"%"
60 PRINT "PERCENT OTHERS      = ";X/T * 100;"%"
70 PRINT "..............................."
80 GOTO 10
89 REM---DATA FOR TWO SURVEYS---
90 DATA 4568, 5132, 489,   4532, 5134, 523
100 END
```

*The first three pieces of data are read by line 10. Then when line 80 says go (back) to 10, the second three pieces of data are read.*

```
>RUN
** COMPARISON OF SURVEYS **
TOTAL VOTERS =  10189
PERCENT REPUBLICANS =  44.8327 %
PERCENT DEMOCRATS   =  50.368 %
PERCENT OTHERS      =  4.79929 %
...............................
TOTAL VOTERS =  10189
PERCENT REPUBLICANS =  44.4793 %
PERCENT DEMOCRATS   =  50.3877 %
PERCENT OTHERS      =  5.13299 %
...............................
OUT OF DATA IN 10
```



When line 10 is executed, the computer looks for a DATA statement (which it finds in line 90), and then stores the first three numbers it finds at line 90 in the variables R, D, and X. One advantage to this method is that the DATA statements can have lots of data. If you then execute the READ statement several times, it will go along the DATA list picking up new values. Which data is used is determined by an invisible pointer that moves along the list of data each time a variable is used. In our example, the pointer moves three positions for each READ.

Here's another example showing how this feature can be used to print unit price labels for the shelves in a grocery. The variables used are:

- N for product number
- Q for package quantity, in ounces
- P for price in dollars

```
4 REM---GROCERY1  (MAKES UNIT PRICE LABELS)---
5 PRINT"-----------------------------------------------------"
10 READ N, Q, P
20 PRINT"PRODUCT QTY (OZ.) PRICE"," UNIT PRICE"
30 PRINT N; TAB(7);Q; TAB(17);P, 100*P/Q; "CENTS PER OZ."
40 GOTO 5
50 DATA 1, 16, .89,    2, 8, 1.49
60 DATA 3, 24, .99,    4, 2.5, 3.50
70 END


>RUN
-----------------------------------------------------
PRODUCT QTY (OZ.) PRICE          UNIT PRICE
 1      16        .89            5.5625 CENTS PER OZ.
-----------------------------------------------------
PRODUCT QTY (OZ.) PRICE          UNIT PRICE
 2      8         1.49           18.625 CENTS PER OZ.
-----------------------------------------------------
PRODUCT QTY (OZ.) PRICE          UNIT PRICE
 3      24        .99            4.125 CENTS PER OZ.
-----------------------------------------------------
PRODUCT QTY (OZ.) PRICE          UNIT PRICE
 4      2.5       3.5            140 CENTS PER OZ.
-----------------------------------------------------
OUT OF DATA IN 10
```

Our program contains data for 4 products, organized in groups of three data items. We put these values in two DATA statements, but we could just as well have used one long DATA statement, four short ones, or any other combination. The only important consideration is that the data always appear in the sequence N, Q, P, N, Q, P... etc.

When this program starts, the data pointer is at the 1. At the first READ, three pieces of data (1, 16, .89) are used, so when the second READ occurs as a result of the GOTO in line 40 the pointer will be at the 2, and so on. When the last three pieces of data are finally read, the

pointer is at the end. When the program goes back to line 10 to read more data, the message OUT OF DATA IN 10 is printed and the program stops.

**Project** *GROCERY2*

Modify the GROCERY1 program so it produces labels for ten products that have weights marked in both pounds and ounces.

**Superproject** *(ENERGY)*

Write a program that stores data about the quantity, price, and calories per ounce for two food products. The program should print out the price per ounce, calories per ounce, and price per calorie for each product, and for a combination of products. For example, suppose milk costs 60 cents per quart, and a breakfast cereal costs 69 cents for a 6-ounce box. If the nutrition information printed on the side of the box states that a 1-ounce serving of dry cereal has 90 calories, while ½ cup (4 oz.) of milk has 80 calories, here's what the output might look like:

```
PRODUCT  PRICE($) QTY.(OZ.)   ¢/OZ.     CAL/OZ.  ¢/CALORIE
CEREAL       .69        6      11.5         90       .12777
MILK         .60       32       1.875       20       .09375
COMBINATION OF 1 OZ. CEREAL WITH 4 OZ. MILK
                      TOTAL COST(¢)     =  19.0
                      TOTAL CALORIES    = 170
                      ¢/CALORIE         =   0.11176
```

## 1.5 DECISION MAKING IN PROGRAMS: VOTER, *RETIRE,* ANSWER

The most interesting programs are those that make decisions by using what are called *conditonal branching* statements. The IF ... THEN statement in BASIC makes this possible. It tells the computer to do one thing if a certain *condition* is true, but to do another thing if that condition is false. The ... represents the condition.

Here's an example which we have run twice to show the two possibilities:

```
10 REM---VOTER  (USES CONDITIONAL ST.)---
15 PRINT "WHAT IS YOUR AGE";
25 INPUT A
35 IF A < 18 THEN 65
45   PRINT "YOU ARE ELIGIBLE TO VOTE."
55   STOP
65 PRINT"YOU WILL BE ABLE TO VOTE IN";18-A;"YEARS."
75 END
```

*The indentations in lines 45 and 55 help you see which statements are executed when A < 18 is false*

```
>RUN
WHAT IS YOUR AGE? 12
YOU WILL BE ABLE TO VOTE IN 6 YEARS.
READY
>RUN
WHAT IS YOUR AGE? 18
YOU ARE ELIGIBLE TO VOTE.
BREAK IN 55
READY
```

Statement 35 is the IF ... THEN statement, The part written A < 18 is the condition being tested. You read it: "A is less than 18." The last part written as THEN 65 gives the line number that the computer will execute next if the condition is true. If the condition is false, the rule is that the computer will execute the statement right after the IF ... THEN statement. In our example, this is statement 45.

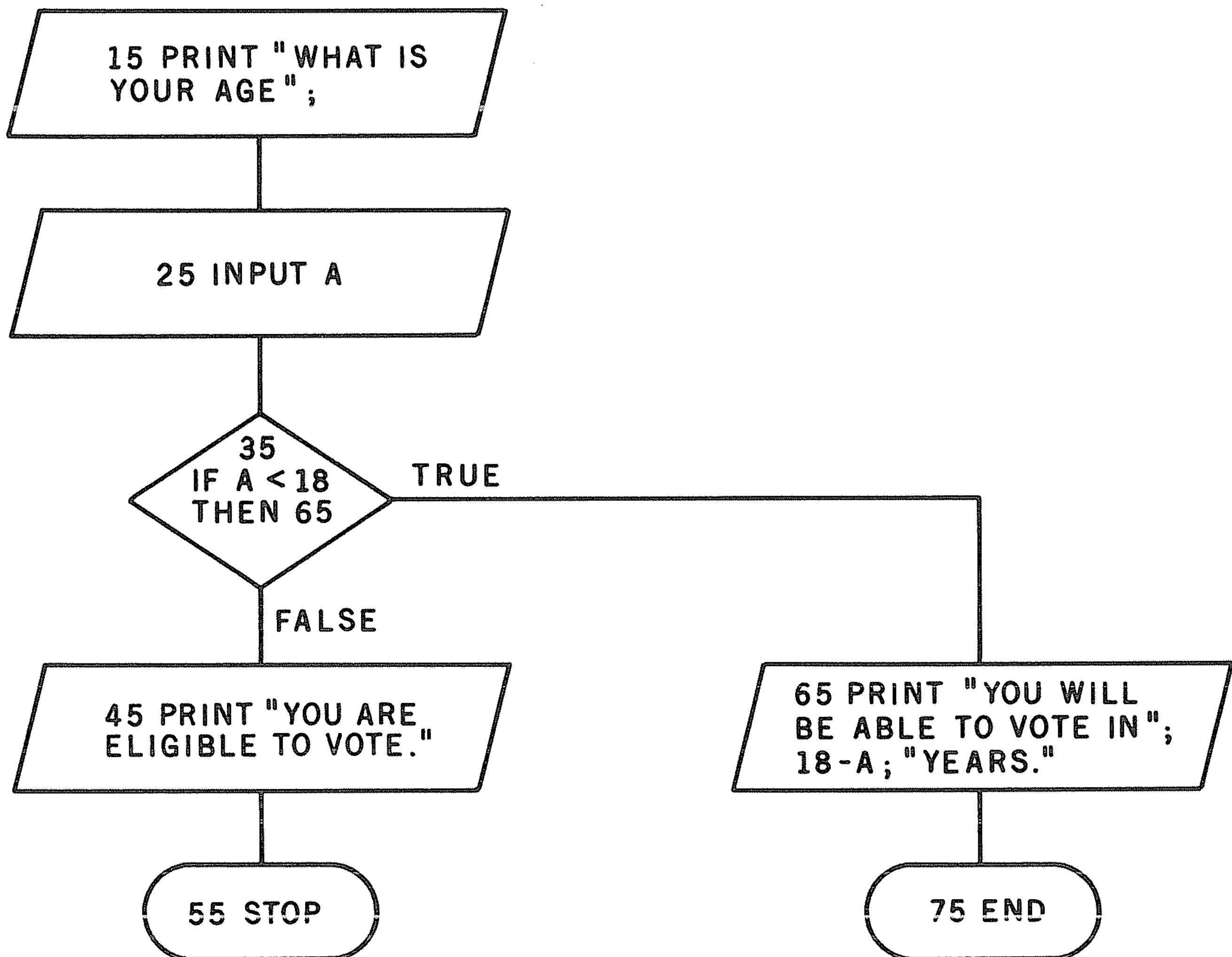
The statement

55 STOP

means that the computer is to stop executing the program at line 55; it should not go on to the end, but stop right where it is. You can have several STOP statements in a program, but only one END, which must be the last statement. Incidentally, the semicolon at the end of line 15 is what causes the ? from the INPUT statement to appear on the same line.

### Using Flow Charts to Picture Programs

Programs with conditional statements can be tricky to follow. A diagram showing all the paths possible through the program helps a lot. This a called a *flowchart*. Here's how our previous program would look in flowchart form.




A flowchart uses boxes with different shapes for different purposes. The conditional statement is shown as a diamond-shaped box, and it indicates the two possible branches (paths) the computer can take. It represents the IF ... THEN statement.

### A NOTE ON FLOWCHARTS

Flowcharts were developed in the early days of computing to help programmers keep track of the logic behind the program, especially when it involved complex "branching" or "jumping." The arrows on the lines made jump statements (like GOTO), and conditional branching statements (like IF ... THEN) easier to follow. Unfortunately, the final result often ended up looking like what some writers called a "bowl of spaghetti." It soon became clear that additional techniques were needed.

With the development of high-level languages (like BASIC), there has been a movement away from detailed flowcharts. The goal today is to make the logic of the program self-documenting. This is done partly by breaking a large program into small program *modules*, partly by using lots of REM (remark) statements that explain each module, partly by insisting that programmers discipline themselves to keep most of the branching within the modules, and partly by extending higher-level languages to include *structured branching* statements. An example of a structured branching statement is IF ... THEN ... ELSE ... (this will be explained in section 2.2). The combined use of all these techniques is usually called *structured programming.*

Structured programming doesn't work very well for complicated applications unless it's used along with something called *structured program design.* We'll discuss some of the techniques used in structured program design in Chapter One and a Half, and use a structured design approach to all the programs from Chapter 2 on. Our approach will use several informal, common sense techniques that work very well in practice. There have also been more formal techniques developed, including one called Warnier-Orr diagrams. Several good articles on this technique can be found in the book *Program Design,* volume 1, *Programming Techniques,* Blaise W. Liffick, ed., BYTE Publications, Peterborough, N.H., 1978.

Other shapes of boxes are used as shown.

There are six types of conditions that can be used in BASIC:

- A < B means "A is less than B"
- A > B means "A is greater than B"
- A = B means "A is equal to B"
- A <= B means "either A is less than B *or* A is equal to B"
- A >= B means "either A is greater than B *or* A is equal to B"
- A <> B means "A is not equal to B"

### Project *RETIRE*

Modify the VOTER program so that it also prints the message "P.S. YOU ARE ELIGIBLE FOR 'OVER 65' BENEFITS" where appropriate.

### More About LET Statements

In addition to being able to store data in a variable, the LET statement can do computations on the right side of the = sign, before storing the results. For example,

10 LET A = 3.1416*18*18/4

will first calculate the quantity shown on the right side (the area of an 18-inch pizza), and then store the answer in A.

An even trickier calculation is one where the contents of a variable are changed on the right side, and the result stored back in the same variable. For example, if X = 12, then LET X = X + 1 will first calculate the expression on the right side as 12 + 1, and then store the result, 13, back in X. So X has been changed from 12 to 13. This is very handy for counting purposes. Here's a program that illustrates how counting is used to control how often a *loop* (from lines 30 to 70) is executed.

```
10 REM---ANSWER  (USES LET TO INCREASE COUNTER)---
20 LET C = 1
30 IF C > 5 THEN 80
40   PRINT "ANSWER #";C
50   PRINT "-------------------------"
60   LET C = C + 1
70   GOTO 30
80 END

>RUN
ANSWER # 1
-------------------------
ANSWER # 2
-------------------------
ANSWER # 3
-------------------------
ANSWER # 4
-------------------------
ANSWER # 5
-------------------------
```

*Another way to interpret line 30 is to read it as meaning "While C < = 5 do everything up to line 80 over and over"*

## How to Interrupt the Printing of Output

The amount of output you can display on a video screen is limited to a fixed number of lines—for example 16 or 24. If your program produces more than this number as output, you won't see all of it on the screen. The beginning lines will *scroll up* out of view. For example, if you try the program ANSWER we've just shown, but with the number 5 in line 30 changed to a 15, the first lines of output will flash by so fast you won't see them. Of course, if you had a printer or typewriter-like terminal for output, all of the output results would be preserved on paper.

To get around this problem, there is a way you can freeze the video output on many computers. On the Radio Shack TRS-80, you press the SHIFT and @ keys at the same time. On other systems you press the control and S keys (called control-S). To resume printing on the Radio Shack screen you press any key. On other systems, you might press control-S or control-Q.

Another trick for handling this problem is to write your program so that it prints only a limited number of lines at a time and then asks you if you are ready to continue. This technique is illustrated in the next section in the program DOGSALES.

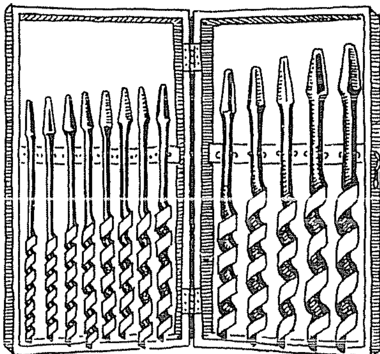
## 1.6 COMPUTER LOOPS AND ARRAYS: DRILL, *SQCUBE*, GOWRONG1, *GOWRONG2*, *GOWRONG3*, SALEDATA, *ICECREAM*, DOGSALES, *DOGDAYS*.

One of the strong points of computers is that they can do lots of repetitious work quickly. BASIC allows you to ask for repetitive computations by using the FOR and NEXT statements. These specify what is called a loop in the program. For example, suppose you want to generate a list of the decimal equivalents of drill bit sizes given as fractions of an inch. Here's how you can do this with three lines of BASIC:

```
5 REM---DRILL---
10 PRINT"FRACTION","DECIMAL EQUIVALENT"
20 FOR N=1 TO 8
30   PRINT N; "/ 8", N/8
40 NEXT N
50 END
```

*Line 30 is the body of the loop. It's executed 8 times.*

```
>RUN
FRACTION       DECIMAL EQUIVALENT
 1 / 8          .125
 2 / 8          .25
 3 / 8          .375
 4 / 8          .5
 5 / 8          .625
 6 / 8          .75
 7 / 8          .875
 8 / 8          1
```



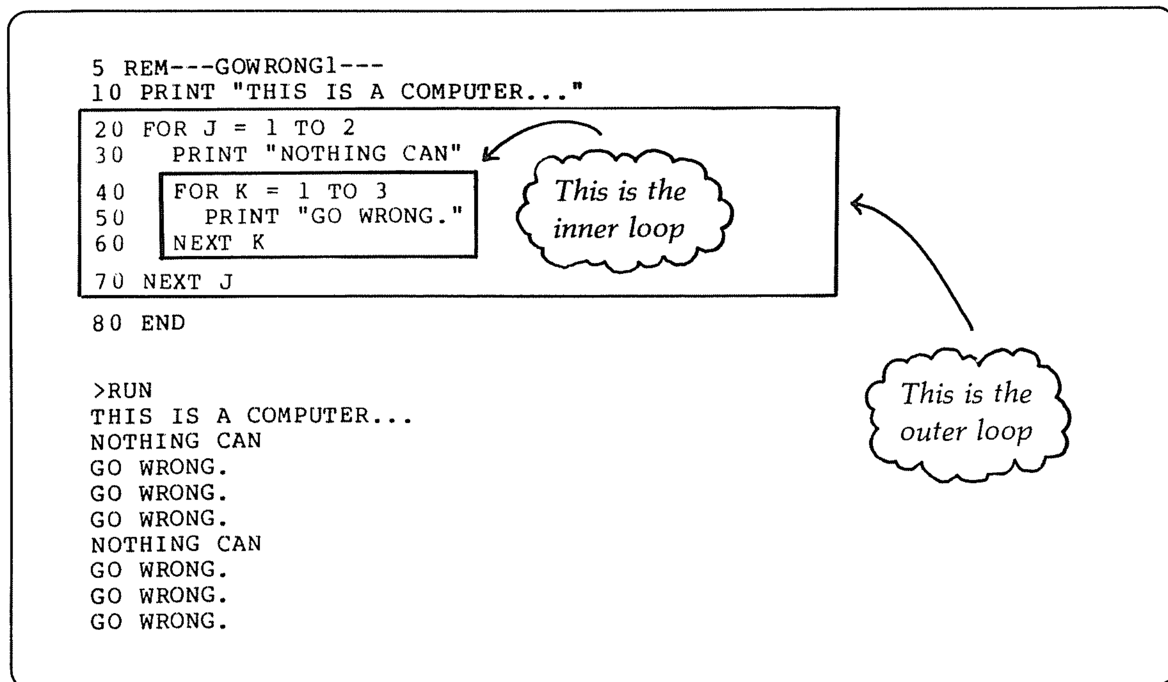
Lines 20, 30, and 40 form our loop. Line 30 is called the body of the loop, and the computer will execute this statement over and over. How often line 30 is repeated, and which values of N are used in its calculations are determined by the FOR statement. In our case, line 30 will be executed 8 times, with N going through the sequence 1, 2, 3, 4, 5, 6, 7, and 8.

### Project *SQCUBE*

Write a program that calculates and prints the squares and cubes of the numbers from 1 to 12.

## Nested FOR Loops

Now for some fun. Let's try making the body of one FOR loop be another FOR loop. This is called *nesting* one loop inside the other. Watch what happens:




You can pretty well figure out what nested loops do by examining the output of this program. To get an even better insight into what's happening, try the following projects.

### Project *GOWRONG2*

Change line 50 in the previous program to the following:

50 PRINT "GO WRONG", "J = ";J;"K = ";K

Now pretend you're a computer, and simulate the running of this program. Write down the output you think the computer will produce on a piece of paper. Then check your results by actually running the program.

### Project *GOWRONG3*

Write a silly message program of the type just shown that uses *three* nested FOR loops.

### Subscripted Variables

Suppose you want to write a program to study the sales in your new ice cream business for the 30 days of the month. You'd like to use the variable D1 to mean sales on day 1, D2 to mean sales on day 2, and so on. Unfortunately you'll soon run out of variable names since D10, D11, D12, etc. are illegal in BASIC. Even worse, you'd need thirty INPUT statements to enter the sales data into your program!

To get around this dilemma, BASIC allows you to use *subscripted* variables. These start with a letter, but are followed by any integer in parentheses. Examples are A(4), B(56), and Z(357). The neat part about subscripted variables is that the number in parentheses can be represented by a variable. So we can use D(I) to mean the sales on day I, and let the computer select I from 1 to 30 (or any other number) in our program.

Here is an example showing how this works. It uses a FOR loop to input as many sales figures as you specify with N. It then calculates the largest sale L, the smallest sale S, and the average sale value A. The daily sales are stored in the subscripted variables D(1), D(2), D(3), and so on. The collection of all these variables is called a one-dimensional *array*. (Some books call an array a *table*, while others call it a *vector*.)

COMPUTER'S MEMORY

D(1) = 47

D(2) = 24

D(3) = 78

.
.
.

D(9) = 32

```
10 REM---SALEDATA  (CALCULATES MIN,MAX,TOTAL,AVG)---
15 DIM D(30)
20 PRINT"HOW MANY DAYS (MAX.= 30)";
25 INPUT N
30 IF N > 30 THEN 20
35 IF N < 1 THEN 998
40 FOR I = 1 TO N
50   PRINT"SALES FOR DAY #";I;
60   INPUT D(I)
70 NEXT I
75 PRINT".................END OF DATA..............."
80 LET L = D(1)
90 LET S = D(1)
100 LET T = D(1)
110 FOR I = 2 TO N
120   IF D(I) > L THEN LET L = D(I)
130   IF D(I) < S THEN LET S = D(I)
140   T = T + D(I)
150 NEXT I
160 PRINT"SMALLEST SALE =";S;TAB(25);"TOTAL SALES  =";T
170 PRINT"LARGEST SALE  =";L;TAB(25);"AVERAGE SALE =";T/N
998 PRINT".................FINISHED................."
999 END
```



```
>RUN
HOW MANY DAYS (MAX.= 30)? 9
SALES FOR DAY # 1 ? 47
SALES FOR DAY # 2 ? 24
SALES FOR DAY # 3 ? 78
SALES FOR DAY # 4 ? 0
SALES FOR DAY # 5 ? 12
SALES FOR DAY # 6 ? 33
SALES FOR DAY # 7 ? 79
SALES FOR DAY # 8 ? 80
SALES FOR DAY # 9 ? 32
.................END OF DATA...............
SMALLEST SALE = 0        TOTAL SALES  = 385
LARGEST SALE  = 80       AVERAGE SALE = 42.7778
..................FINISHED.................
```

Our program starts with the statement DIM D(30) which means "dimension (set aside) 30 variable names D(1), D(2), etc. up to D(30)." Some computers also give you an extra variable name D(0) at the beginning of the list. The sales data is then INPUT with the FOR loop in lines 40 to 70. Then a second FOR loop is used to search through all these figures to find the largest, L, the smallest, S, and the total, T (in lines 110 to 150).

The trick to finding L and S is to start by guessing that the first sale is both the largest (line 80) and the smallest (line 90). When a better value is found, it is used instead (lines 120 and 130). The total sales are accumulated in line 140 where each new sale is added to the previous total T. The average sale is calculated as T/N and printed with the other results in lines 160 and 170.

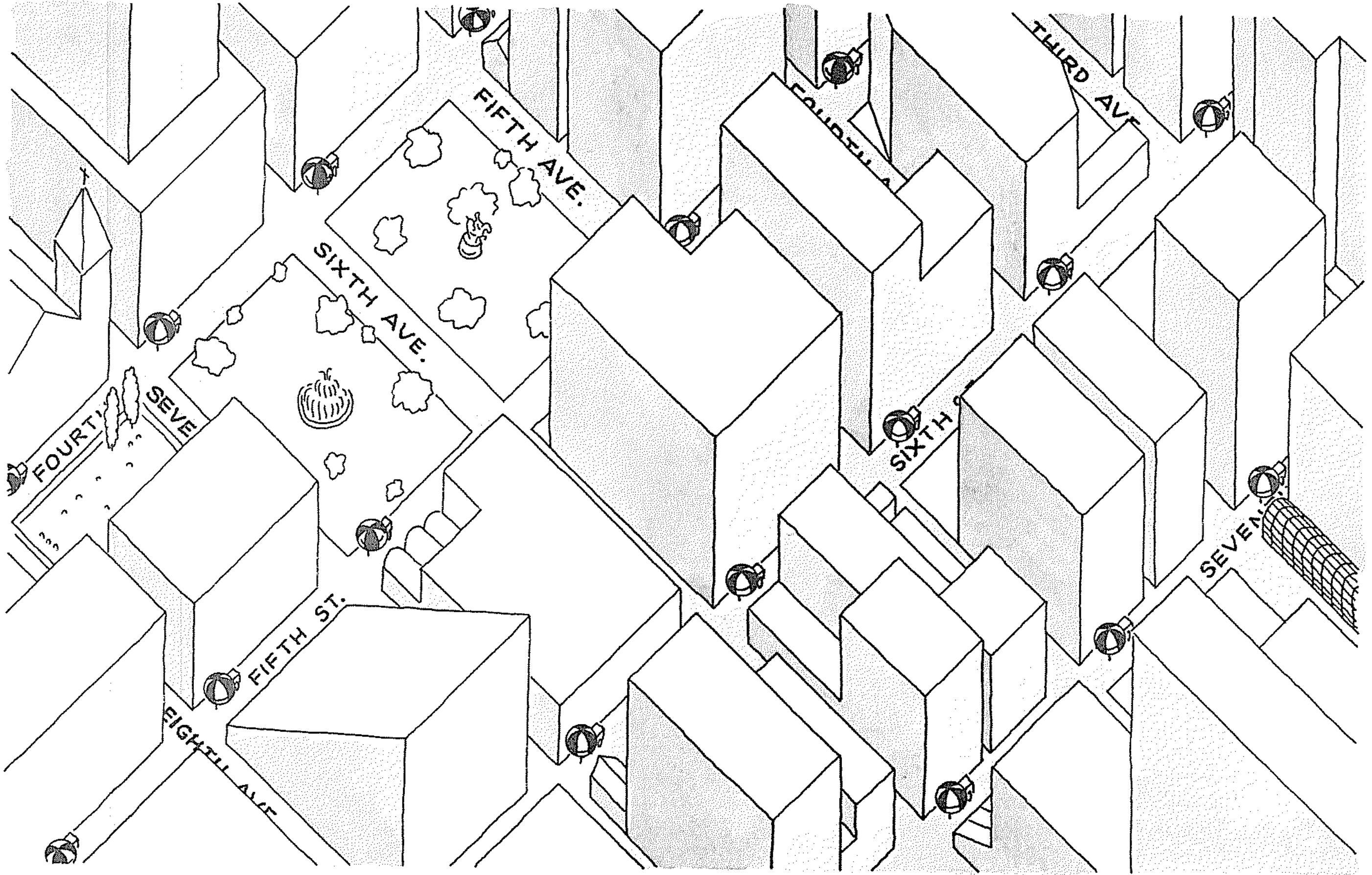
Since this is the most complicated program we've shown so far, don't be surprised if you have to restudy it a few times to fully understand how it works. *Note:* Lines 120 and 130 of SALEDATA use an *extended* IF statement. This allows the keyword THEN to be followed by certain other BASIC words (like LET, PRINT, or GOTO). Chapter 2 discusses extended BASIC statements in greater detail.

**Project** *ICECREAM*

See if you can write the preceding program using only one FOR loop. *Hint:* Use the INPUT loop to also calculate S, L, and T. Before going into this loop, set T = 0, L = 0, and S = 999999. (Why?)

## Two-Dimensional Arrays

Suppose you manage a chain of pushcart hot dog vendors. A map of the city showing where they operate looks like this:




You decide to store the sales figures in a computer program, and would like to use variables that correspond to these locations. That's easy in BASIC, since you can use variables with two subscripts. So S(3,4) could be used to mean sales at the corner of Third Avenue and Fourth Street, S(10,2) for 10th Avenue and Second Street, and so on.

Here's a program to collect the sales data and give back a report on sales along each street, sales along each avenue, and total sales for the entire city:

```
100 REM---DOGSALES  (USES 2-D ARRAYS)---------------------
104 CLS
105 PRINT"****** REPORT OF HOTDOG SALES ******"
106 PRINT
110 DIM S(11,9), R(11), C(9)
120 PRINT"HOW MANY AVENUES";
130 INPUT M
140 PRINT"HOW MANY STREETS";
150 INPUT N
155 REM---AN IMPROVED PROG. WOULD CHECK IF M,N ARE LEGAL---
160 FOR I = 1 TO M
170   PRINT"TYPE";N;"SALES FOR AVENUE";I
180   FOR J = 1 TO N
185     PRINT "AT STREET #"; J;
190     INPUT S(I,J)
200   NEXT J
205   PRINT"PRESS ENTER (OR RETURN) TO CONTINUE--READY";
206   INPUT D$
210 NEXT I
220 REM---SUMMARY BLOCK-----------------------------------
240 PRINT"SUMMARY OF SALES (STS. SHOWN ACROSS, AVES. DOWN)"
245 FOR J = 1 TO N
250   PRINT TAB(6*J);"S";J;
260 NEXT J
270 PRINT
280 FOR J=1 TO N+1
285   PRINT"------";
290 NEXT J
295 PRINT
300 FOR I = 1 TO M
305   PRINT"A";I;":";
310   FOR J = 1 TO N
320     LET R(I) = R(I) + S(I,J)
330     LET C(J) = C(J) + S(I,J)
340     PRINT TAB(6*J);S(I,J);
350   NEXT J
360   PRINT
370 NEXT I
380 PRINT
390 PRINT"PRESS ENTER (OR RETURN) TO CONTINUE--READY";
400 INPUT D$
405 REM---SUM OF AVE. SALES BLOCK-------------------------
410 PRINT"TOTAL SALES BY AVENUES"
420 FOR I = 1 TO M
430  PRINT "FOR AVE #";I;"TOTAL =";R(I)
440 NEXT I
450 PRINT"PRESS ENTER (OR RETURN) TO CONTINUE--READY";
460 INPUT D$
470 PRINT
475 REM---SUM OF ST. SALES BLOCK--------------------------
480 PRINT "TOTAL SALES BY STREETS"
490 FOR J = 1 TO N
500   PRINT"FOR ST. #";J;"TOTAL =";C(J)
510 NEXT J
515 PRINT
520 PRINT"****** END OF REPORT ******"
999 END
```

*Here's where the number of sales of each street corner are fed into the computer*

*And here's where all the information about various sales totals is generated and printed*

```
****** REPORT OF HOTDOG SALES ******

HOW MANY AVENUES? 3
HOW MANY STREETS? 4
TYPE 4 SALES FOR AVENUE 1
AT STREET # 1 ? 34
AT STREET # 2 ? 46
AT STREET # 3 ? 58
AT STREET # 4 ? 98
PRESS ENTER (OR RETURN) TO CONTINUE--READY?

TYPE 4 SALES FOR AVENUE 2
AT STREET # 1 ? 44
AT STREET # 2 ? 85
AT STREET # 3 ? 57
AT STREET # 4 ? 62
PRESS ENTER (OR RETURN) TO CONTINUE--READY?

TYPE 4 SALES FOR AVENUE 3
AT STREET # 1 ? 95
AT STREET # 2 ? 75
AT STREET # 3 ? 36
AT STREET # 4 ? 88
PRESS ENTER (OR RETURN) TO CONTINUE--READY?

SUMMARY OF SALES (STS. SHOWN ACROSS, AVES. DOWN)
       S 1    S 2    S 3    S 4
------------------------------
A 1 :  34     46     58     98
A 2 :  44     85     57     62
A 3 :  95     75     36     88

PRESS ENTER (OR RETURN) TO CONTINUE--READY?

TOTAL SALES BY AVENUES
FOR AVE # 1 TOTAL = 236
FOR AVE # 2 TOTAL = 248
FOR AVE # 3 TOTAL = 294
PRESS ENTER (OR RETURN) TO CONTINUE--READY?


TOTAL SALES BY STREETS
FOR ST. # 1 TOTAL = 173
FOR ST. # 2 TOTAL = 206
FOR ST. # 3 TOTAL = 151
FOR ST. # 4 TOTAL = 248

****** END OF REPORT ******
```

The first part of the statement

110 DIM S(11,9), R(11), C(9)

means "dimension (reserve) a two-dimensional array called S with at most 11 rows for avenues, and 9 columns for streets." Later, when you use INPUT S(I,J) or PRINT S(I,J) you then automatically refer to a computer location that corresponds to sales at the corner of Avenue I and Street J. The same DIM statement is used to reserve eleven locations, R(11), for holding the sums of sales by avenues (rows), and nine locations, C(9), for

the sums of sales by streets (columns). A good way to study this program is to picture the computer's memory as being structured in the form of a rectangular table of variables for S, plus two one dimensional arrays for R and C.

## COMPUTER'S MEMORY

| S(1,1) = 34 | S(1,2) = 46 | S(1,3) = 58 | S(1,4) = 98 | S(1,5) = 0 | S(1,6) = 0 | S(1,7) = 0 | S(1,8) = 0 | S(1,9) = 0 |
|---|---|---|---|---|---|---|---|---|
| S(2,1) = 44 | S(2,2) = 85 | S(2,3) = 57 | S(2,4) = 62 | S(2,5) = 0 | S(2,6) = 0 | S(2,7) = 0 | S(2,8) = 0 | S(2,9) = 0 |
| S(3,1) = 95 | S(3,2) = 75 | S(3,3) = 36 | S(3,4) = 88 | S(3,5) = 0 | S(3,6) = 0 | S(3,7) = 0 | S(3,8) = 0 | S(3,9) = 0 |
| S(4,1) = 0 | S(4,2) = 0 | S(4,3) = 0 | S(4,4) = 0 | S(4,5) = 0 | S(4,6) = 0 | S(4,7) = 0 | S(4,8) = 0 | S(4,9) = 0 |
| S(5,1) = 0 | S(5,2) = 0 | S(5,3) = 0 | S(5,4) = 0 | S(5,5) = 0 | S(5,6) = 0 | S(5,7) = 0 | S(5,8) = 0 | S(5,9) = 0 |
| S(6,1) = 0 | S(6,2) = 0 | S(6,3) = 0 | S(6,4) = 0 | S(6,5) = 0 | S(6,6) = 0 | S(6,7) = 0 | S(6,8) = 0 | S(6,9) = 0 |
| S(7,1) = 0 | S(7,2) = 0 | S(7,3) = 0 | S(7,4) = 0 | S(7,5) = 0 | S(7,6) = 0 | S(7,7) = 0 | S(7,8) = 0 | S(7,9) = 0 |
| S(8,1) = 0 | S(8,2) = 0 | S(8,3) = 0 | S(8,4) = 0 | S(8,5) = 0 | S(8,6) = 0 | S(8,7) = 0 | S(8,8) = 0 | S(8,9) = 0 |
| S(9,1) = 0 | S(9,2) = 0 | S(9,3) = 0 | S(9,4) = 0 | S(9,5) = 0 | S(9,6) = 0 | S(9,7) = 0 | S(9,8) = 0 | S(9,9) = 0 |
| S(10,1) = 0 | S(10,2) = 0 | S(10,3) = 0 | S(10,4) = 0 | S(10,5) = 0 | S(10,6) = 0 | S(10,7) = 0 | S(10,8) = 0 | S(10,9) = 0 |
| S(11,1) = 0 | S(11,2) = 0 | S(11,3) = 0 | S(11,4) = 0 | S(11,5) = 0 | S(11,6) = 0 | S(11,7) = 0 | S(11,8) = 0 | S(11,9) = 0 |

| R(1) = 236 |
|---|
| R(2) = 248 |
| R(3) = 294 |
| R(4) = 0 |
| R(5) = 0 |
| R(6) = 0 |
| R(7) = 0 |
| R(8) = 0 |
| R(9) = 0 |
| R(10) = 0 |
| R(11) = 0 |

| C(1) = 173 |
|---|
| C(2) = 206 |
| C(3) = 151 |
| C(4) = 248 |
| C(5) = 0 |
| C(6) = 0 |
| C(7) = 0 |
| C(8) = 0 |
| C(9) = 0 |

**Note:** Zeros are shown in the unused parts of the arrays since most versions of BASIC automatically initialize all variables to zero when a program is run.

### Project *DOGDAYS*

Enter the program DOGSALES in your computer, get it to work, and then save it on tape (or disk). This is a long program and it will be easy to make typing mistakes, producing what are called bugs (errors in the program). If this happens, get a friend to do the typing while you proofread over his or her shoulder. We've called the project DOGDAYS because that's what the first few days of debugging will seem like.

After you have it working, see if you can improve the program so that it checks for illegal input. For example, in the question HOW MANY AVENUES?, the inputs −3, 0, 4.5, or 12 should be rejected as illegal. Another improvement would be to distinguish between a street corner where there is no hot dog cart, and a corner where there is a cart but no sales. In this second situation the number zero should be entered for sales and it should go into calculating totals. In the first case, a −1 should be entered to mean no cart and it should not go into calculating totals.

## 1.7 PROCESSING WORDS: FAMOUS, *SAM, POEM, APRIL1*

Computer programs written in BASIC can handle letters as well as numbers, using variables that have a dollar sign ($) added to a letter (for example A$, B$, etc). These are called *string* variables, where string means any sequence of characters like X, 31, JOHN, BAH!!, or even GUSUMPH#@*!---. Below is a simple example you can use on visitors. When the INPUT statement causes the ? to print, you then type a name (string) which gets stored in A$. You can see that the INPUT statement can use string variables as well as numeric variables.

```
10 REM---FAMOUS---
20 PRINT"WHAT IS YOUR NAME";
30 INPUT A$
40 PRINT"***   GOOD GRIEF    ***"
50 FOR K = 0 TO 20 STEP 5
60   PRINT TAB(K);"IT'S THE FAMOUS GENIUS ";A$
70 NEXT K
80 PRINT"??????????--WHO'S ";A$;"--??????????"
90 END


>RUN
WHAT IS YOUR NAME? ISABELLE T. SMITH
***   GOOD GRIEF   ***
IT'S THE FAMOUS GENIUS ISABELLE T. SMITH
     IT'S THE FAMOUS GENIUS ISABELLE T. SMITH
          IT'S THE FAMOUS GENIUS ISABELLE T. SMITH
               IT'S THE FAMOUS GENIUS ISABELLE T. SMITH
                    IT'S THE FAMOUS GENIUS ISABELLE T. SMITH
??????????--WHO'S ISABELLE T. SMITH--??????????
READY
```

*This makes K = 0,5,10,15,20. Another way to write this program would be to change line 50 to 50 for K = 0 to 5 and change tab (K) to tab (4*K)*

This program should work on most computers. However, with some of the simpler BASIC interpreters, you may have to first dimension (reserve space) for the string variable by starting with a statement like this:

15 DIM A$(25)

The key word TAB(K) in this program causes each line of output to be printed K spaces over to the right. This feature is useful for producing graphical output. We'll show some examples of computer graphics in the

next chapter. We'll also have more to say about string variables there, including subscripted string variables like A$(I), or P$(I,J).

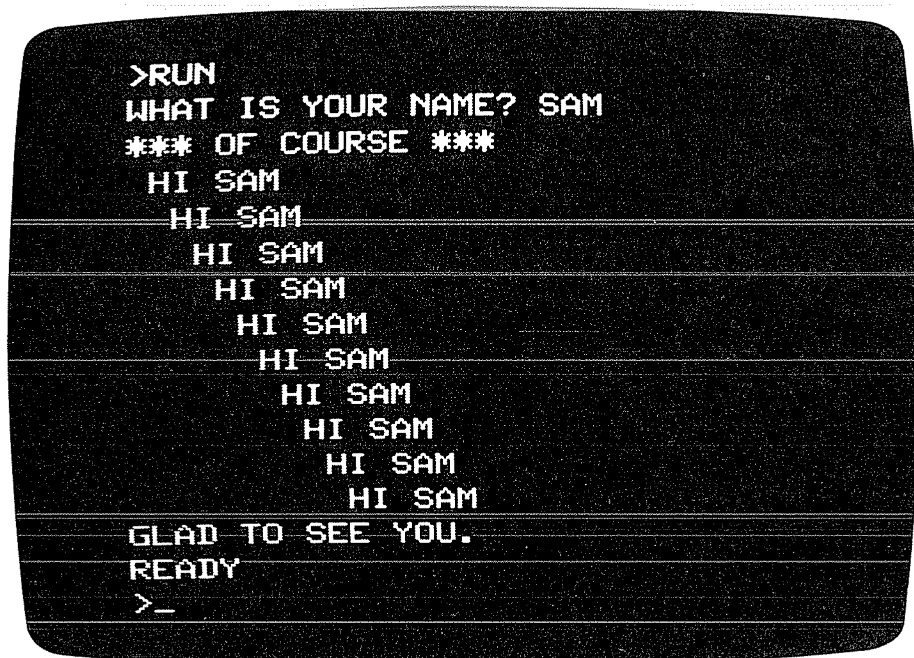
### Project *SAM*

If your computer allows only 40 characters on a line, the previous program will spill over from line to line. Rewrite FAMOUS so it stays within 40 columns. *Hint:* Change line 50 to

```
50 FOR K = 0 TO 10 STEP 2
```

You'll also want to shorten line 80, and input a shorter name, like SAM. On some machines you may also have to add the line,

```
15 DIM A$(25)
```




*This is an example of output from a program similar to FAMOUS that uses double-sized characters, with 32 characters printed on each line.*

### Project *POEM*

Find a standard poem that can be varied by changing some of its words. Then write a program that allows the user to input those words, storing them in string variables like A$, B$, C$, etc. It should then print back the poem with the user's words inserted in the proper place. Here's an example of what a run might look like:

```
>RUN
WITH YOUR HELP, THIS PROGRAM WILL
WRITE AN 'ORIGINAL' POEM.

WHAT'S A GOOD WORD FOR A PLACE--
LIKE DESERT, MOON, FOREST, ALLEY, ETC.    ? ARCTIC
THANKS.
GIVE ME A PLURAL NOUN FOR SOME
THINGS THAT BELONG ON LAND.               ? HOT CAKES
THANKS.
NOW I NEED A PLURAL NOUN FOR SOME
THINGS THAT LIVE IN THE SEA.              ? OYSTERS
THANKS.


          A RIDDLE

THE MAN IN THE ARCTIC ASKED OF ME,
HOW MANY HOT CAKES GROW IN THE SEA?
I ANSWERED HIM AS I THOUGHT GOOD,
AS MANY AS OYSTERS GROW IN THE WOOD.


WANT TO PRINT IT AGAIN (Y = YES)? NO
WANT TO WRITE ANOTHER (Y = YES)? NO
```

### Project *APRIL1*

Write a program that creates filler paragraphs for the April 1st issue of a newspaper. First find a paragraph on some general subject, using the inside pages of the paper. It should be fairly short, 50 to 150 words. Then choose words that can be replaced without completely losing the meaning of the sentences they are in. Have the program ask the user for an appropriate part of speech for each blank—without revealing the paragraph, of course. Then have the program print the paragraph with the user's chosen words inserted. You can help the humor along by narrowing the choice at times; for example, ask for a geographical place, a number, an exclamation, the name of an animal, and so on. (*Note:* This program can make a good vocabulary and spelling exercise for youngsters, especially when they try to outdo each successive run.)

A sample of the output for APRIL1 is given on the next page.

```
***  APRIL 1ST NEWSPAPER ARTICLE   ***

PLEASE TYPE IN THE FOLLOWING:
A NOUN (A MESSY SUBSTANCE)           ? GLUE
A NOUN (PERSONAL PROPERTY)           ? HAT
  OH, HOW DO YOU SPELL THE PLURAL OF THAT? HATS
THE NAME OF A CRAFT OR INDUSTRY      ? PLUMBING
A TOOL                               ? CROWBAR
A COMMON HOUSEHOLD ARTICLE           ? BED
  AND HOW DO YOU SPELL THE PLURAL OF THAT? BEDS
ANOTHER HOUSEHOLD ARTICLE            ? SALAMI
A LIQUID                             ? SODA POP
     THANK YOU.


               AUNT HOMEBODY

     WE'VE TOLD YOU HOW TO REMOVE GLUE SPOTS FROM
WALLS AND FURNITURE--NOW FOR HATS!
     THE PLUMBING ASSOCIATION TELLS US THAT IF THE
HAT IS WASHABLE AND HAS ONLY A FEW SPOTS, DO THIS:
     REMOVE AS MUCH OF THE GLUE AS POSSIBLE WITH A
DULL CROWBAR (A LEFT-HANDED CROWBAR IS PERFECT).  NOW PLACE
THE STAINED AREA BETWEEN BEDS AND PRESS WITH A
WARM SALAMI.  AFTER YOU HAVE DONE THIS, PLACE THE
HAT, STAIN SIDE DOWN, ON A BED AND RUB ANY
REMAINING STAIN ON THE BACK WITH SODA POP.
     LET DRY AND LAUNDER AS USUAL.
```

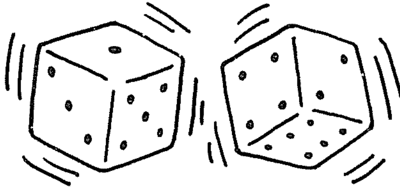
### A NOTE ABOUT THE INPUT STATEMENT

The program FAMOUS shows that INPUT can be used with a string variable like A$. You can also use several string variables after INPUT, separating them with commas. The user must then supply several strings, also separating them with commas. Example:

```
> 10 PRINT "TYPE LAST NAME, FIRST NAME"
> 20 INPUT L$, F$
> 30 PRINT "YOUR NAME IS ";F$;" ";L$
> RUN
> TYPE LAST NAME, FIRST NAME
? SMITH, HORATIO
  YOUR NAME IS HORATIO SMITH
```

One problem can arise: There's no way to input a single string that contains a comma. The computer will think you were trying to input two strings, and complain. To get around this, a special statement called LINE INPUT has been invented. It will be explained in chapter 4, section 4.2

## 1.8 USING BUILT-IN FUNCTIONS IN BASIC. DICE, *CRAPS, TABGRAPH*



If you've done much mathematical calculation, you know that you must occasionally obtain values like sines, cosines, logarithms, square roots, and so on from a handbook. TRS-80 Level II BASIC does this for you automatically for a number of such functions. For example, SQR(X) gives you the square root of X, SIN(A) gives you the sine of A, INT(N) gives the integer part of N, ABS(Y) gives the absolute value of Y, and so on. Other BASIC dialects may use slightly different names for these functions.

Absolute value is handy in finding the distance between two numbers, regardless of sign (ABS(5-3) = 2, but also ABS(3-5) = 2). This is applied in checking for correct answers in *Project VARIETY* which is discussed in the next chapter. This example also shows you how to use the SQR and INT functions.

Games, quiz programs, and simulations all become more interesting when you are able to use "surprise" or *random* numbers. In BASIC, RND(0) (or on some computers RND(1)) gives you a random number between 0 and 1 which seems to be selected by chance each time RND(0) is used. You can get larger random numbers by multiplying RND(0) by a constant, or adding something to it. For example, INT(6 * RND(0) + 1) will give random numbers that are integers between 1 and 6.

Here's a simple dice throwing program that uses this formula:

```
100 REM---DICE  (USES RANDOM NUMBER FUNCTION)---
105 PRINT"*** DICE ROLL ***"
110 LET C = 0
120 PRINT"WHAT IS YOUR POINT (2 TO 12)";
130 INPUT P
140 IF P > 12 THEN PRINT"TOO LARGE": GOTO 120
150 IF P < 2 THEN PRINT"TOO SMALL": GOTO 120
160 REM---THIS IS START OF DICE THROWING LOOP---
170 LET D1 = INT(6*RND(0)+1)
180 LET D2 = INT(6*RND(0)+1)
190 LET S = D1 + D2
200 PRINT"THE DICE ARE THROWN...THE SUM IS";S
210 LET C = C + 1
220 IF S = P THEN 300
230   GOTO 170
300 PRINT"***  YOU MADE YOUR POINT IN";C;"THROWS."
310 PRINT"WANT TO TRY TO DO BETTER ";
320 INPUT A$
330 IF LEFT$(A$,1) = "Y" THEN 110
340 END
```

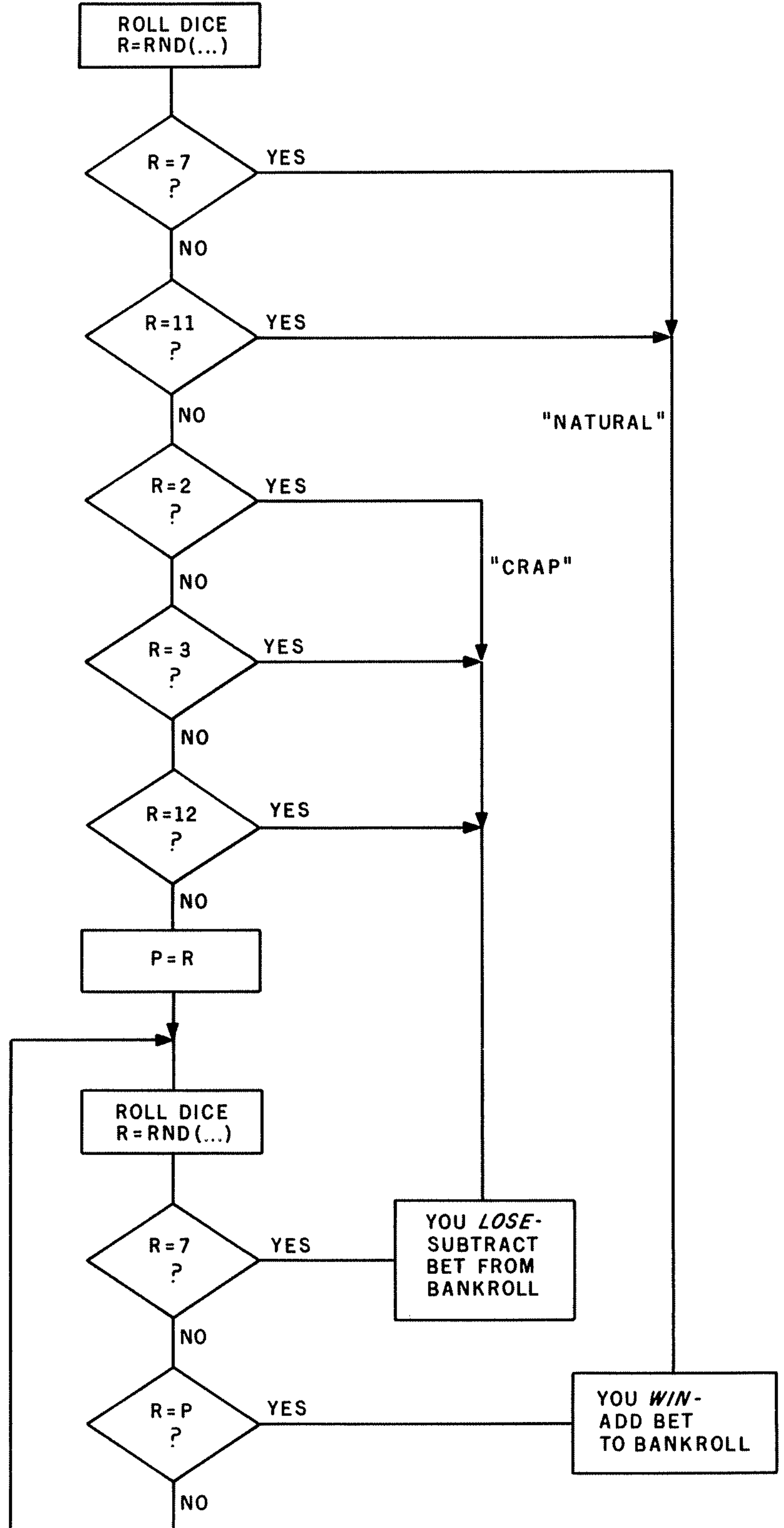
*Each time line 170 or 180 is executed, a new random number from 1 to 6 is produced. On the TRS-80 you could change these lines to D1 = RND(6) D2 = RND(6)*

```
>RUN
*** DICE ROLL ***
WHAT IS YOUR POINT (2 TO 12)? 7
THE DICE ARE THROWN...THE SUM IS 10
THE DICE ARE THROWN...THE SUM IS 8
THE DICE ARE THROWN...THE SUM IS 8
THE DICE ARE THROWN...THE SUM IS 9
THE DICE ARE THROWN...THE SUM IS 12
THE DICE ARE THROWN...THE SUM IS 9
THE DICE ARE THROWN...THE SUM IS 5
THE DICE ARE THROWN...THE SUM IS 11
THE DICE ARE THROWN...THE SUM IS 5
THE DICE ARE THROWN...THE SUM IS 11
THE DICE ARE THROWN...THE SUM IS 7
***  YOU MADE YOUR POINT IN 11 THROWS.
WANT TO TRY TO DO BETTER ? YES

WHAT IS YOUR POINT (2 TO 12)? 11
THE DICE ARE THROWN...THE SUM IS 4
THE DICE ARE THROWN...THE SUM IS 7
THE DICE ARE THROWN...THE SUM IS 7
THE DICE ARE THROWN...THE SUM IS 9
THE DICE ARE THROWN...THE SUM IS 9
THE DICE ARE THROWN...THE SUM IS 8
THE DICE ARE THROWN...THE SUM IS 6
THE DICE ARE THROWN...THE SUM IS 5
THE DICE ARE THROWN...THE SUM IS 10
THE DICE ARE THROWN...THE SUM IS 11
***  YOU MADE YOUR POINT IN 10 THROWS.
WANT TO TRY TO DO BETTER ? YUP

WHAT IS YOUR POINT (2 TO 12)? 7
THE DICE ARE THROWN...THE SUM IS 6
THE DICE ARE THROWN...THE SUM IS 5
THE DICE ARE THROWN...THE SUM IS 6
THE DICE ARE THROWN...THE SUM IS 8
THE DICE ARE THROWN...THE SUM IS 7
***  YOU MADE YOUR POINT IN 5 THROWS.
WANT TO TRY TO DO BETTER ? NOPE
```

### Project *CRAPS*

Write a program that simulates playing the game of craps. This will require you to look up the rules of the game, think them through carefully and write BASIC statements that (1) ask the person what his "bankroll" is (so when he wins or loses it can be changed), (2) ask the person how much he is betting, (3) roll each die (using statements like those in DICE), (4) use conditional statements to determine if the player won or lost, (5) add or subtract from the bankroll, and (6) ask if the person wants to play again. A simplified flowchart based on the usual rules of Craps is shown on the facing page.

*THE RULES OF CRAPS*

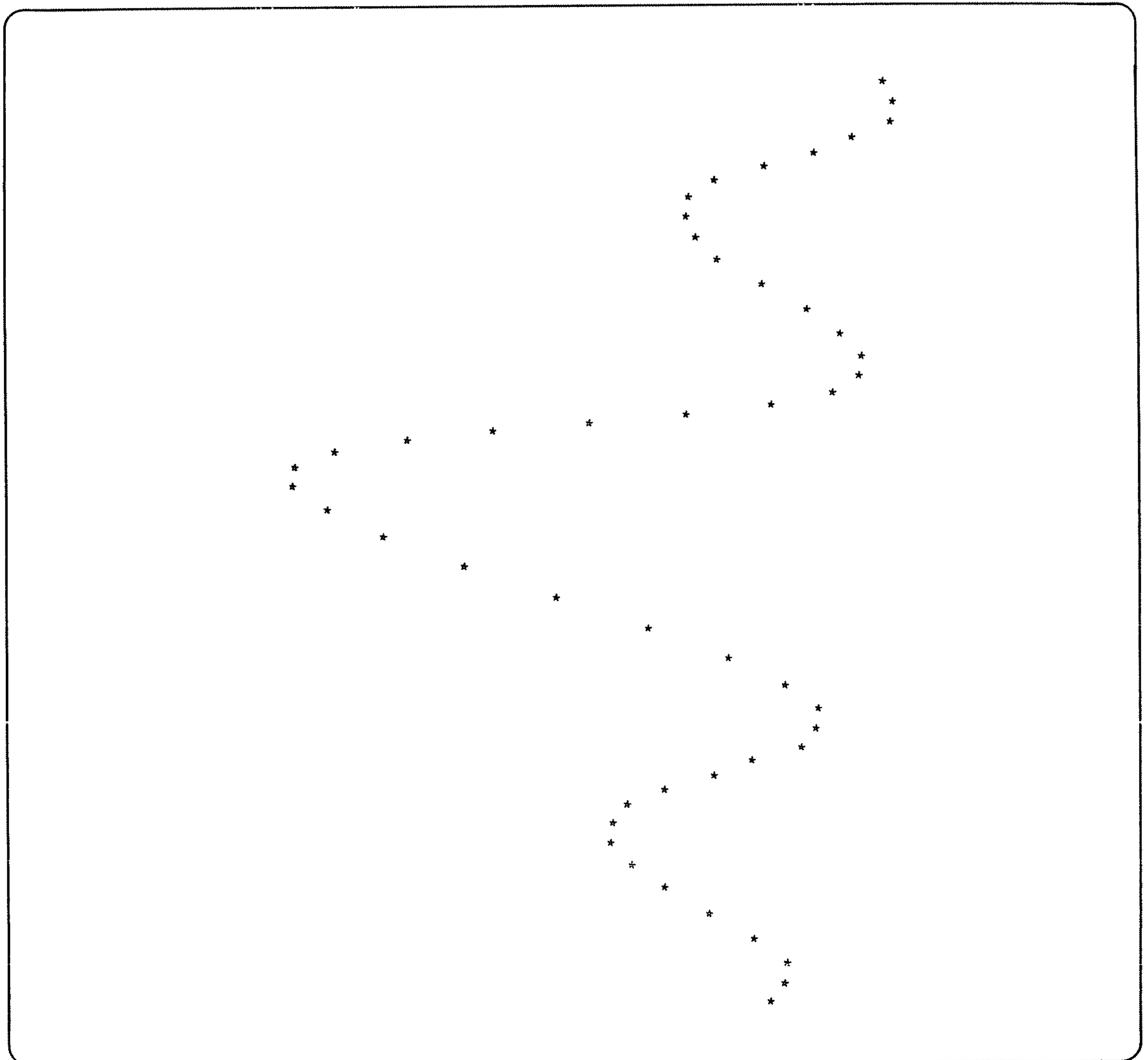


This flowchart only details the rules which govern the playing of Craps. Additional steps are necessary to provide input from the players, output messages to the players, multiple games, etc. For some help see page 75 of *BASIC and the Personal Computer* (Addison Wesley, Reading MA 01867).

**Project** *TABGRAPH*

Write a program that uses the SIN(A) and COS(A) functions together with TAB(X) to plot a wiggling curve. If this subject is new to you, Chapter 3 of *Basic and the Personal Computer* discusses several versions of SIN and COS graphs. Chapter 7 of the same book shows the techniques that can be used for plotting more complex mathematical functions.

Here's what a sample run would look like if the amount tabbed over was controlled by COS(2*A) + SIN(A), where A can be thought of as an angle going from 0 to 9.5 radians.



## 1.9 OTHER FEATURES OF BASIC. STARS, FANCY, *THROW A PARTY*

We've now covered the most important features of standard BASIC (sometimes called *minimal* BASIC). The vocabulary that's been developed is sufficient for handling a large variety of programs. However, experience has shown that programmers are a restless and imaginative lot, and they keep on asking for more and more capabilities. This has resulted over the years in what is usually called *extended* BASIC, a language that includes minimal BASIC as a subset, but adds many other powerful features.

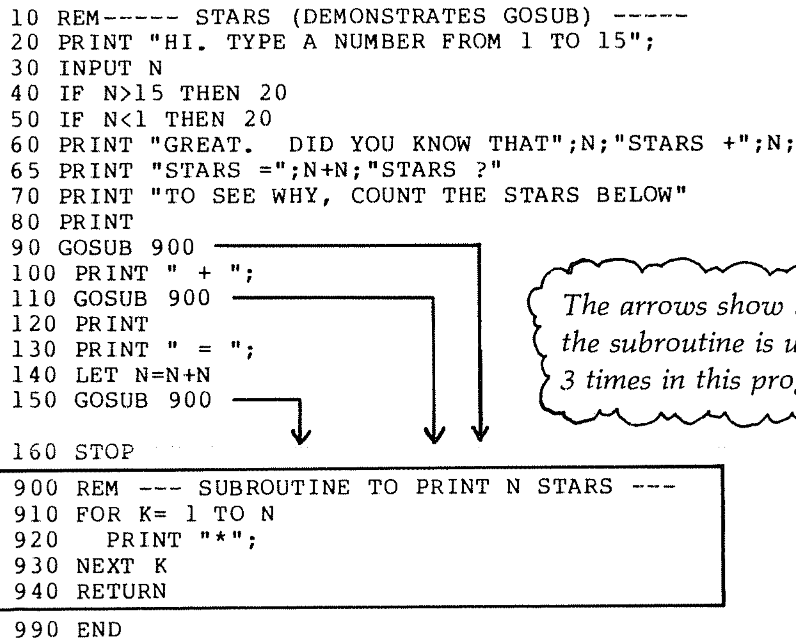
The best known extended BASIC is the version produced by the Microsoft Company, marketed under such names as TRS-80 Level II BASIC, Altair BASIC, PET BASIC, Exidy Sorcerer BASIC, APPLESOFT BASIC, OSI BASIC, and several others. We'll take a look at most of the extra features of TRS-80 Level II BASIC in the remaining chapters of the book (particularly in Chapter 2, section 2.2).

In this section we'll conclude our discussion of minimal BASIC by introducing three additional statements that will set the scene for some of the projects coming up in Chapter 2.

### GOSUB 900

This statement is a way of saying to the computer, "Go do the subroutine that starts at line 900. When you find the word RETURN, come back to this statement and continue with the next statement."

*A subroutine* is a small subprogram imbedded inside a main program. One reason for using a subroutine is that it can be called upon from different parts of the main program. For example, a subroutine to print a line of N stars could be called upon three times in the elementary math program shown on the next page.



```
10 REM----- STARS (DEMONSTRATES GOSUB) -----
20 PRINT "HI. TYPE A NUMBER FROM 1 TO 15";
30 INPUT N
40 IF N>15 THEN 20
50 IF N<1 THEN 20
60 PRINT "GREAT.  DID YOU KNOW THAT";N;"STARS +";N;
65 PRINT "STARS =";N+N;"STARS ?"
70 PRINT "TO SEE WHY, COUNT THE STARS BELOW"
80 PRINT
90 GOSUB 900
100 PRINT " + ";
110 GOSUB 900
120 PRINT
130 PRINT " = ";
140 LET N=N+N
150 GOSUB 900
160 STOP
900 REM --- SUBROUTINE TO PRINT N STARS ---
910 FOR K= 1 TO N
920   PRINT "*";
930 NEXT K
940 RETURN
990 END

>RUN
HI. TYPE A NUMBER FROM 1 TO 15? 5
GREAT.  DID YOU KNOW THAT 5 STARS + 5 STARS = 10 STARS ?
TO SEE WHY, COUNT THE STARS BELOW

***** + *****
 = **********
```



## ON K GOTO 80, 110, 140, 170

This statement allows branching to one of several statements (in our example these would be the statements with line numbers 80, 110, 140, and 170). There can be a reasonably long list of these line numbers, corresponding to as many *cases* K as you wish to consider. When K = 1, the program branches to the first line number, when K = 2, to the second line number, and so on. Here's an example using four line numbers:

```
10 REM --- FANCY  (DEMONSTRATES ON K GOTO) ---
20 PRINT "HOW FANCY DO YOU WISH TO GET";
30 INPUT K
40 IF K > 4 THEN 20
50 IF K < 1 THEN 20
60 ON K GOTO 80, 110, 140, 170
70 REM --- FIRST CASE ---
80 PRINT "YOU TYPED A 1.  THAT'S FANCY."
90 GOTO 20
100 REM --- SECOND CASE ---
110 PRINT "YOU TYPED A 2.  THAT'S VERY FANCY."
120 GOTO 20
130 REM --- THIRD CASE ---
140 PRINT "YOU TYPED A 3.  THAT'S SUPER FANCY."
150 GOTO 20
160 REM--- FOURTH CASE ---
170 PRINT "YOU TYPED A 4.  THAT'S ULTRA FANCY."
180 GOTO 20
190 END

>RUN
HOW FANCY DO YOU WISH TO GET? 2
YOU TYPED A 2.  THAT'S VERY FANCY.
HOW FANCY DO YOU WISH TO GET? 1
YOU TYPED A 1.  THAT'S FANCY.
HOW FANCY DO YOU WISH TO GET? 3
YOU TYPED A 3.  THAT'S SUPER FANCY.
HOW FANCY DO YOU WISH TO GET? 4
YOU TYPED A 4.  THAT'S ULTRA FANCY.
HOW FANCY DO YOU WISH TO GET?
BREAK IN 30
READY
>
```

*Note:* This program can only be stopped by pressing BREAK.

## ON K GOSUB 900, 1000, 1100, 1200

This works in a manner similar to ON K GOTO .... The difference is that each line number must correspond to a subroutine. When K = 1, the program branches to the first of these subroutines, when K = 2, it goes to the second, and so on. In each case, when the RETURN at the end of the subroutine is encountered, the program then automatically goes back to the line right after the ON K GOSUB statement. An example of a situation in which this statement is useful will be discussed in Chapter 2, section 2.1 under the heading **Project** *VARIETY1.* A more advanced example of its use will be given in Chapter 3, section 3.6 in the program MONEY.

## Project *THROW A PARTY*

As mentioned in the introduction, the first chapter of this book covers about as much material as found in some introductory computer programming courses. This means that you shouldn't expect to understand everything on a first reading, and that it will be helpful to go over the material several times, at least if you're new at programming.

A good trick for reviewing new ideas and really mastering them is to try to explain them to others. The problem is finding victims for such an experiment. One way might be to throw a party where you just happen to have your computer up and running. Then when someone expresses interest in the gadget, you'll have an excuse for showing off some of the programs you've written and explaining how they work. If possible, let the other person do the typing at the keyboard. Also don't be too anxious to correct his or her mistakes—remember, your goal is to learn as much as your "student," and seeing what bugs others find in your work can be really educational.

## Exercises for Chapter 1

Here are some exercises to try before moving on to the next chapter. Some of them ask you to *simulate* a run. To do this, divide your work paper into two parts. On the left, draw a picture of the computer's output screen (or paper). On the right, draw a box for each program variable, and put a zero (or space) in each box for a start. Then trace through the program statements just as the computer would, placing the output you'd get in the output space, while changing variable values as called for. Example:

```
10 REM %%% EXERCISE #0.  SIMULATE A RUN OF THIS PROGRAM %%%
20 FOR K=1 TO 3
30 LET A=2*K+10
40 PRINT K,A
50 NEXT K
60 END
```

To simulate a run, you would trace through this program in the order 10, 20, 30, 40, 50, 20, 30, 40, 50, 20, 30, 40, 50, 60. When finished, here's what you'd have on your paper:

```
OUTPUT                    VARIABLES

> RUN
1   12                    K    1    2    3
2   14
3   16                    A    12   14   16
```

To check your work, you can then try the program on an actual computer. If the results are different be sure to figure out what went wrong before going ahead.

```
10 REM %%% EXERCISE 1. SIMULATE A RUN OF THIS PROGRAM %%%
20 LET A=4
30 LET B=3
40 LET S=A+B
50 PRINT S; "COME"; S+A
60 END
```

In exercises 2 and 3, first fill in the missing parts, then simulate a RUN of the program.

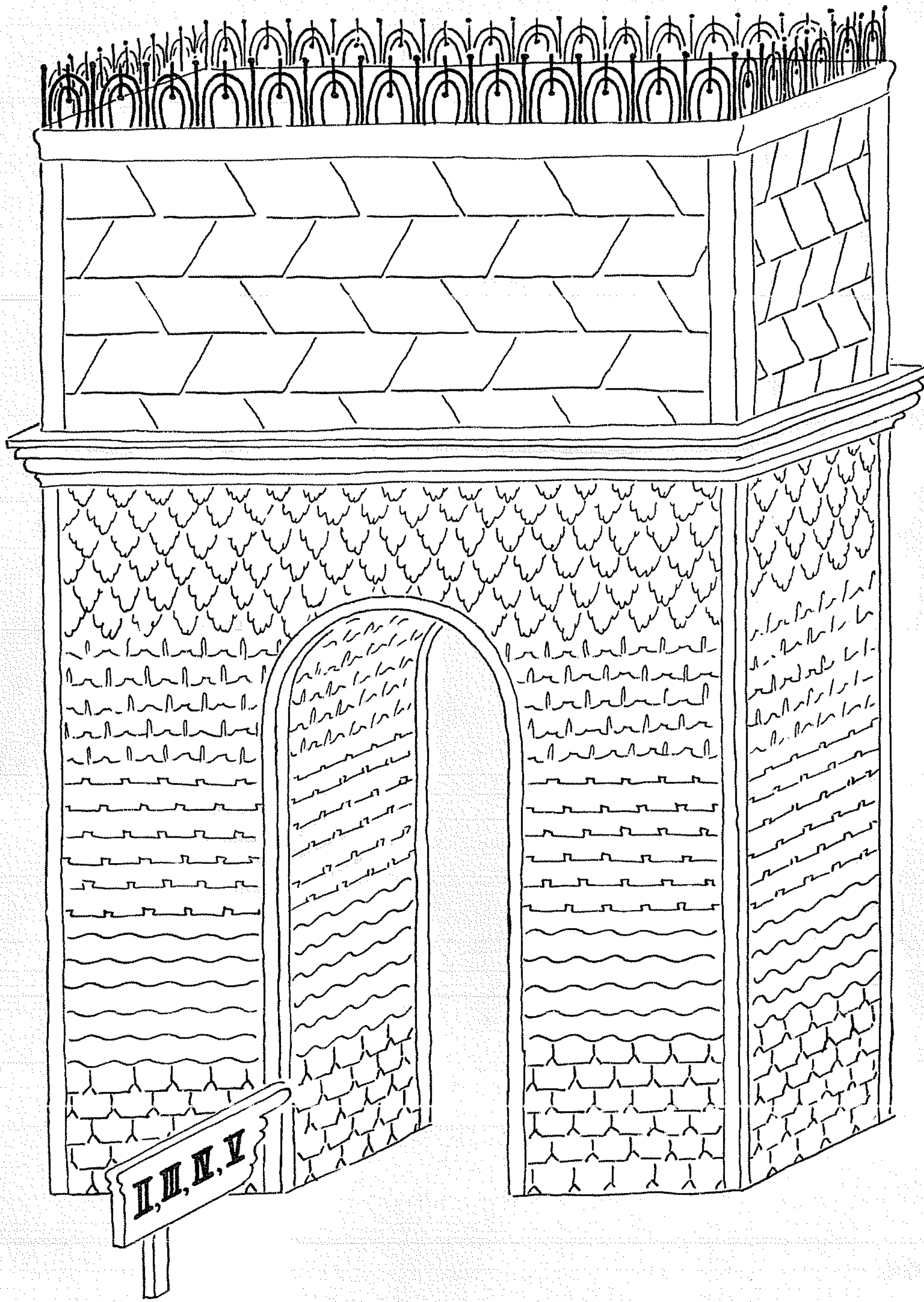
```
10 REM %%% EXERCISE 2.  FILL IN THE MISSING ----- PARTS %%%
20 PRINT "HOW MANY QUARTERS DO YOU HAVE";
30 INPUT Q
40 PRINT "THAT'S EQUAL TO"; ----- ; "PENNIES"
50 PRINT "IT'S ALSO EQUAL TO"; ----- ; "DOLLARS"
60 END
```

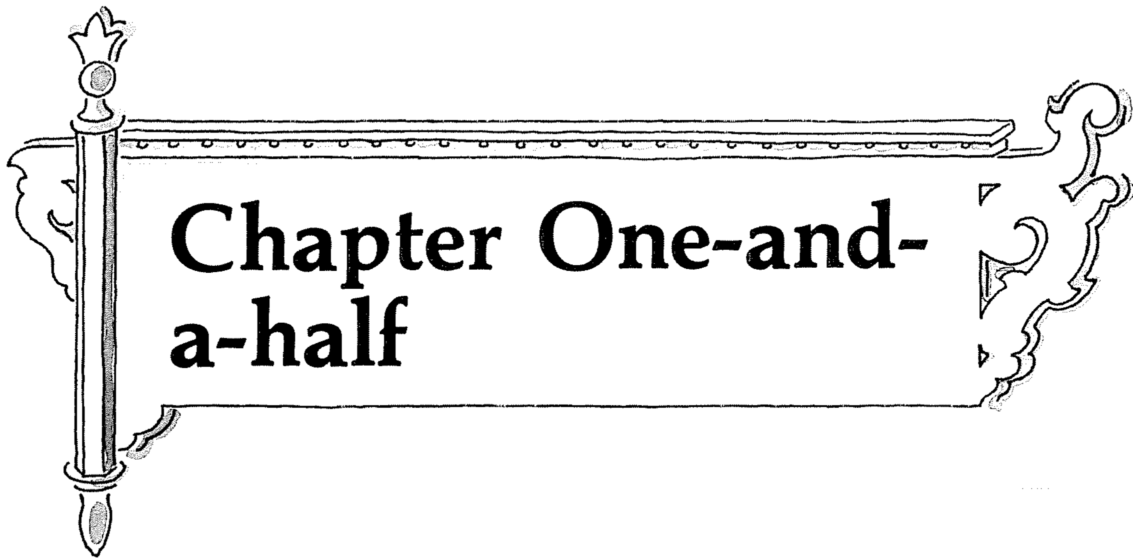
```
10 REM %%% EXERCISE 3.  FILL IN THE MISSING ----- PARTS %%%
20 PRINT "HOW MANY QUARTERS, DIMES, AND NICKELS DO YOU HAVE";
30 INPUT Q,D,N
40 PRINT "THAT'S EQUAL TO"; ----- ; "PENNIES"
50 PRINT "IT'S ALSO EQUAL TO"; ----- ; "DOLLARS"
60 END
```

```
10 REM %%% EXERCISE 4.  SIMULATE A RUN OF THIS PROGRAM %%%
20 PRINT "THIS PROGRAM HAS BEEN"
30 FOR I= 1 TO 3
40    PRINT "BROUGHT TO YOU"
50    FOR J= 1 TO 2*I
60       PRINT "LIVE!!! ";
70    NEXT J
80    PRINT
90 NEXT I
100 END
```

## SOURCES OF FURTHER INFORMATION FOR CHAPTER 1.

1. *BASIC and the Personal Computer* (Addison-Wesley Co., Reading, Mass., 1978). This is a 440-page book that covers BASIC in detail. It also examines a variety of applications, and discusses such subjects as computer art, sorting, minimax games, sequential files, simulations, and color graphics.
2. *Creative Computing.* (51 Dumont Pl., Morristown, NJ 07960) A monthly magazine that features an informal coverage of many topics, and regularly publishes game programs in BASIC.
3. *BYTE.* (70 Main St., Peterborough, NH 03458) The oldest, largest, and best regarded of the small computer magazines. It is worth saving back issues as a reference collection.
4. *Personal Computing.* (1050 Commonwealth Ave., Boston, Mass. 02215) This magazine is aimed at the beginning computer user. It has also carried a large number of articles reporting on the progress of computer chess.
5. *OnComputing.* (70 Main St., Peterborough, NH 03458) This is a new personal computing magazine with a slant toward the beginner. It includes extensive equipment reviews.


II, III, IV, V



# Chapter One-and-a-half

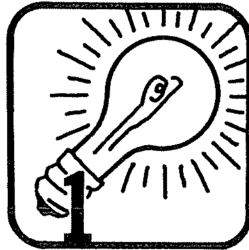
## Structured Program Design

In Chapter 1 we introduced the vocabulary of minimal BASIC and demonstrated how it can be used to write computer programs. As the vocabulary increased, these programs became longer, a pattern which will continue throughout the remainder of the book. To make these longer programs more readable, we'll include a variety of remark statements in the listings.

However, something more than readability is needed to help you think out and design longer programs. Experience has shown that describing and designing computer programs in a consistent, orderly fashion is what helps most. A scheme for doing this in five steps will be explained in this chapter. The use of this scheme will then be illustrated in the design of all the major programs ahead.

In case you're wondering, the longer application programs in this book, or in any other book, weren't immediately written in the form you see. Revisions and improvements are a way of life in the world of computing. Although you may not follow the suggested design sequence for every program you write, we think that keeping this procedure in the back of your mind will make the creative process a little smoother, and the final product clearer and easier to debug.

Here are the five steps in program design we suggest you follow and which we'll illustrate in the chapters ahead.



Step 1 *The Idea* Try to be clear about what it is you want the program to do. If it's a game, you should explain the order of play, the rules to be followed, how winning and losing will be determined, and what the outcome will be. If it's a business program, you should describe its purpose, the transaction data to be gathered, and the final reports that will be produced. In general, you should be clear about the what and why of your idea. Like any preliminary step, you may have to come back to this stage more than once before your program is completed.

Step 2 *A Sample Run* What do you want the computer to print out? What do you want the user to type? Some parts you forgot to include under step 1 may pop up here. If there is more than one possibility in a particular interaction, write out an example of each possibility. Think about what sort of instructions a person using the program might need, and when they should be printed out. How do you tell the user what to type for an input? At the end of the program, should there be a final message to the user? You might even draw a sequence of TV display screens and write the expected results of a run inside them. Putting this step ahead of writing the program may seem a little strange, since it means you must *imagine* the run first. But give it a try; we think you'll eventually find it a powerful technique.

Step 3 *High-Level Design* Now that you know what the program is supposed to do, and what its output will look like, you can start thinking about the major tasks the program must carry out, and the order in which they should be done. Try to hold off from writing actual computer code just yet. Pretend you have someone else to take care of the details—you just have to lay out the general plan of the program (this is the meaning of "high-level design"). You are like an executive giving orders: "First do this, then do this, ... last of all, do this." The result should be an outline (or in the case of simpler programs, a complete statement) of the program, written in English. The English language, when used carefully, is an excellent high-level tool.

Remember to think about what variables you will need. In planning longer programs, it is a good idea to make a table of variable names and their meanings. Decide whether you want to store some values in an array using subscripted variables like X(1), X(2), and so on, or in a two-dimensional array using variables like X(1,1), X(1,2), and so on. This is called deciding on your *data structures.*

If a very similar task is done more than once in the program, such as drawing a dashed line, or stopping to wait for the user to read the display and push a button, try making it into a subroutine.

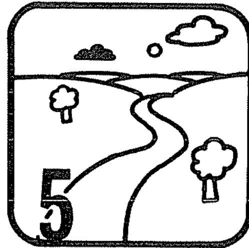
Finally, convert some, or all, of the sentences in your high-level design into BASIC remark statements like:

```
10   REM---"BUCKSHOT" (HUNT SIMULATION GAME)---
100  REM---GIVE INSTRUCTIONS, INITIALIZE VARIABLES---
200  REM---BEGIN GAME LOOP---
                 .
                 .
                 .
1000 REM---SUBROUTINE (DRAWS A LINE)---
1100 REM---SUBROUTINE (DISPLAYS GRAPHICAL OUTPUT)---
                 .
                 .
                 .
```

Each remark statement above describes a *block* of BASIC code for which, in this example, up to 99 numbered statements can be written in step 4.

Step 4. **Coding the Program** With all this planning out of the way, you're almost home free. You can now settle down to writing the actual BASIC statements that make the computer do what you want. By the way, you may have to revise your plans; move a block of code to a different position or add new blocks you hadn't thought of before. Don't despair! This is all part of the art of programming.

*Hint:* If you write a GOTO (or an IF...THEN) statement that sends control back to an earlier statement, and then find yourself writing another GOTO statement to get around a later block of code, you'd better think about rearranging some blocks of code, even if it takes extra typing.

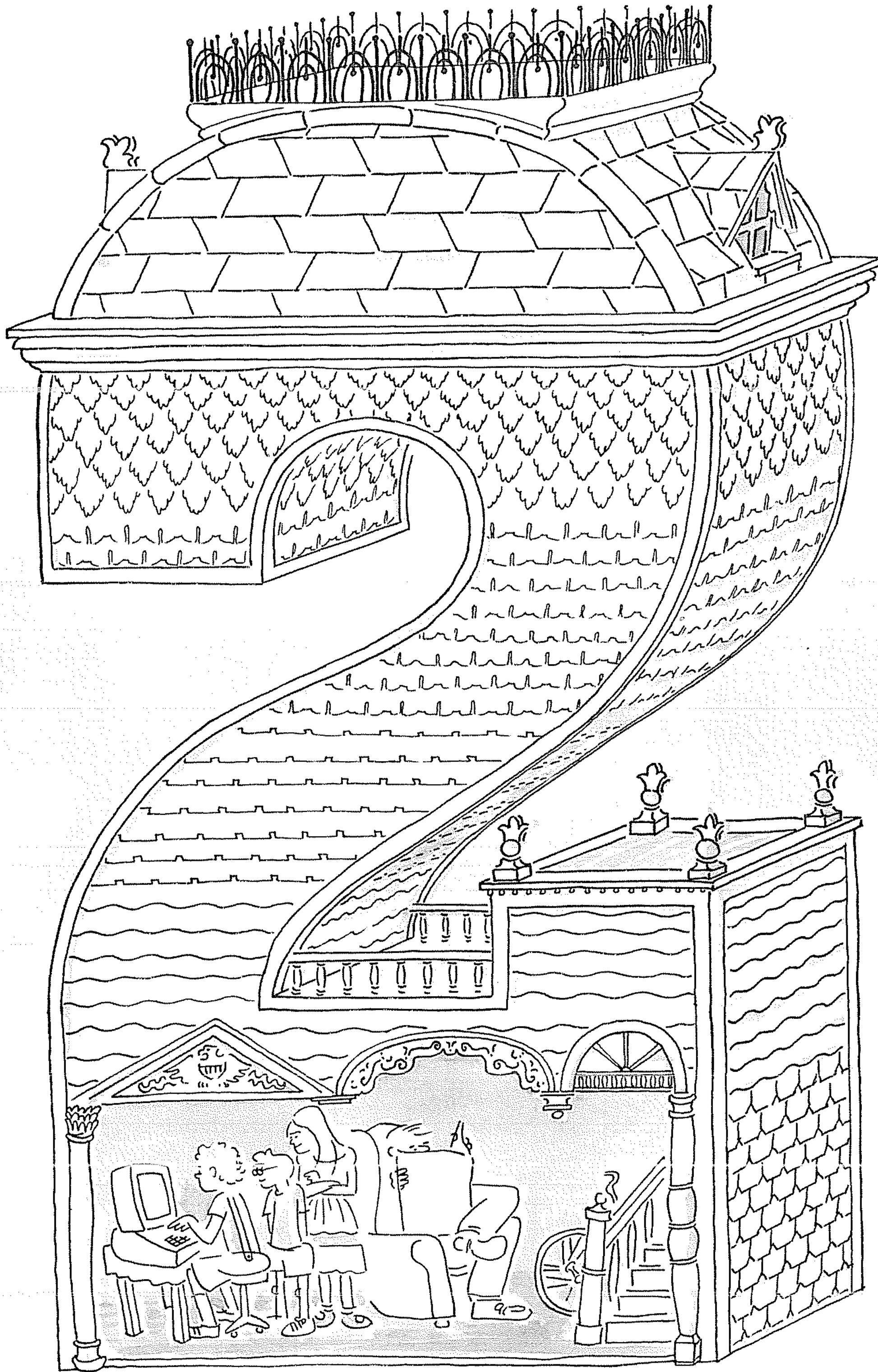


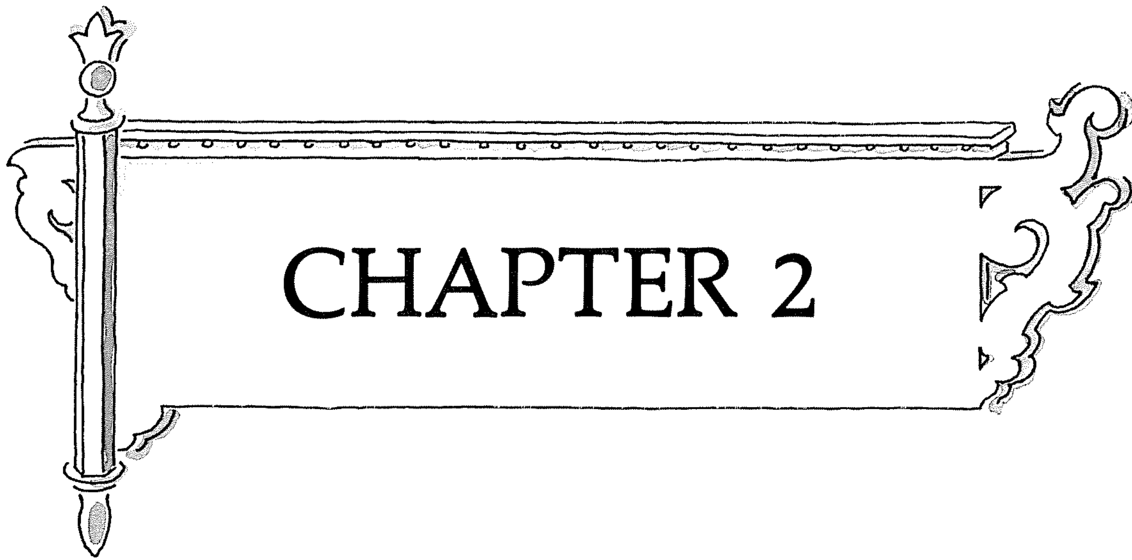
Step 5 (Optional) **Future Extensions and Revisions** Once you have a working program, two things are likely to happen: (1) You will keep on adding sections or making changes, making it "just a bit better" or "just a bit more fun," or (2) you will be inspired to write a whole new program along somewhat the same lines, but for a different purpose. This is the habit-forming part of programming a computer. In moderation, it's quite harmless, and often very intriguing. However there may come a time when moving on to new challenges is the best way to grow. That's when the best advice will be "go back to Step 1".

## SOURCES OF FURTHER INFORMATION.

The five stages of program design discussed in this chapter are related to what some authors call *structured program design*. The idea is to organize the total design process so that the final pieces all fit together neatly. Structured design is a form of discipline that forces you to think about where you are headed before you try to get there.

The phrase *structured programming* has a narrower meaning. It says that the *coding* of a program (step 4) must carefully adhere to the high-level design (step 3), and that the programmer should avoid jumping around (or into or out of) the major blocks in the high-level design. This is why some people associate structured programming with avoiding haphazard use of the GOTO statement. An advanced book on structured program design is *Software Tools* by Kernighan and Plauger (Addison-Wesley Co., Reading, Mass., 1976). A good series of articles on structured programming appeared in *BYTE* magazine in 1978 (July, p. 32, August, p. 143, and September, p. 68). A collection of 17 articles and essays (including 10 reprints from BYTE) on the subject can be found in the BYTE Book *Program Design*.



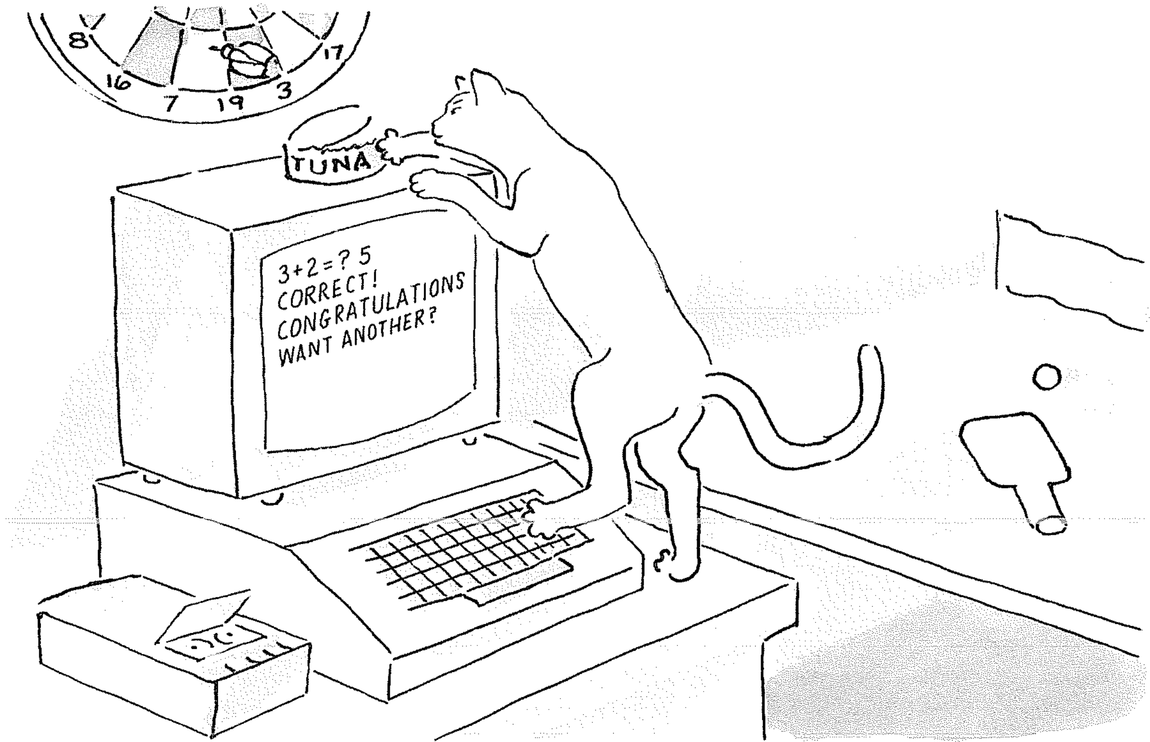



# PUT AN EDUCATIONAL GAME ROOM IN YOUR BASEMENT

## 2.0 COMPUTERS, GAMES, AND LEARNING

Some folks don't like to admit it, but people young and old often learn more at play than in school. It's now possible to combine the playing of games with learning in a new and intriguing way. How? With computer fantasy games based on mathematics, logic, and word manipulation. This use alone can easily justify the cost of a home computer.

In this chapter we're going to look at a number of games, but we're going to concentrate on those that have the bonus of being educational. We think you'll find that the word educational in this case is not synonymous with boring. In fact the situation is just the opposite. That's why kids, including some with gray hair, get very sharp at the skills involved in computer game playing. There's little doubt that children can get a real head start on lots of useful skills if they have access to a home computer.

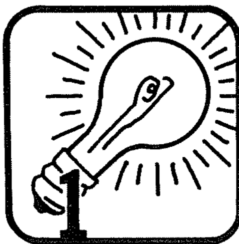



The games we'll examine center on three themes: number games, word games, and science fiction space games. Some of these programs are obviously connected with school learning, while others involve ideas not found in most classrooms. But they all have the elements of suspense and surprise that are characteristic of friendly human-computer interactions.

In addition to their recreational and educational value, the game programs in this chapter were selected for a third reason. They illustrate a number of general programming techniques that will prove valuable in other settings. By the end of the chapter you will have used such exotic ideas as "modular program design," "uniformly distributed pseudorandom numbers," "two-dimensional data structures," and "binary search algorithms." And it won't even hurt.

## 2.1 SIMPLE NUMBER GAMES. ARITH, *DATARITH, DATARIT2, DATARIT3, DATARIT4, RNDARITH, MULT, VARIETY1, VARIETY2, SQROOT*

Although the idea of a quiz has associations with school, it is really a kind of game. After all, if you come up with the right answer, you win; if not, you lose. Our first attempt to describe the process of designing a structured computer program will be an arithmetic quiz. Frankly, this is because a computer naturally "takes" to this sort of quiz. We will go on to word quizzes and space games later.



### Program ARITH

#### The Idea

The idea behind any arithmetic quiz can be stated briefly: give the person some problems, collect the answers, see if they are right or wrong, keep a score, give the person the score. In order to make this into a computer program, we'll have to be a little more specific.

How many problems will the program give? One solution is to ask the person how many he or she wants. What kind of problem? Well, for a start let's just do addition problems. Where do we get the numbers to make up the problems? Again, let's ask the person for the numbers.

Now that we're fairly specific about what we want to do, let's imagine what a run would look like.

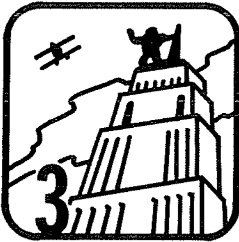
#### Sample Run

Coming up with a run before writing the program is difficult at first. It takes imagination and experimentation. You try to anticipate what the person will need to see, things such as:

- a heading to announce what the program is going to do
- the proper wording of the requests for the number of problems and the numbers to be used
- the form for printing out the problem. This one is done horizontally to save space on a TV display screen, but many school children demand the vertical form
- what to actually print out if the answer is right or wrong, and how to express the score

```
RUN
ADDITION PRACTICE
HOW MANY PROBLEMS DO YOU WANT? 3
TYPE IN 2 NUMBERS SEPARATED BY A COMMA? 2,2
 2 + 2 =  ? 4
RIGHT!!
TYPE IN 2 NUMBERS SEPARATED BY A COMMA? 143,276
 143 + 276 =  ? 219
WRONG.  THE ANSWER IS 419
TYPE IN 2 NUMBERS SEPARATED BY A COMMA? 9,13
 9 + 13 =  ? 22
RIGHT!!
YOUR SCORE IS  66.6667 % RIGHT.
```

In our run, you'll notice that we decided to print out the percent right rather than the percent wrong. We also decided to give the right answer after the person types in the wrong one. This is a good idea for most quiz programs.




### High-Level Design

Now let's try a high-level design for this program. What we want the program to do is:

0. Print a heading.
1. Find out how many problems the person wants.
2. Generate a problem, but first get two numbers from the person.
3. Present the problem.
4. Get the person's answer.
5. Check if the answer is correct or incorrect, print out an appropriate message for either, and count the correct answers.
6. Go back and generate the next problem (step 2).
7. When finished generating problems, print the score.

A set of BASIC REMARK statements for this high-level design might look like this:

```
10 REM---ARITH (USES INPUT TO GET NUMS.)---
100 REM---INTRODUCTION, INITIALIZE VARIABLES---
200 REM---GENERATE PROBLEM---
300 REM---PRESENT PROBLEM---
400 REM---INPUT ANSWER---
500 REM---CHECK IF ANSWER IS RIGHT/WRONG---
600 REM---GET NEXT PROBLEM---
700 REM---REPORT SCORE---
800 REM---FUTURE EXTENSION OF PROGRAM---
```

Line 10 shows that it's a good idea to start with a remark that explains what file name this program will be stored under in mass storage. A little note in parentheses says what is unique or important about this program. Line 100 indicates that this block is where you introduce the program to the user and put values in memory for any variables you may need. This is called initializing the variables because these beginning, or initial, values may change later in the program. So line 100, in this case, takes care of points 0 and 1 of the high-level design.

The rest of the remarks correspond closely to the steps in the high-level design, except for line 800. It's a good idea to reserve a bunch of line numbers toward the end of your program for extensions to the program. You might put your END statement at line 900, leaving 100 line numbers free. Of course it's easy to move an END statement, but why not leave room to put a subroutine or some DATA statements down there.

## Coding the Program

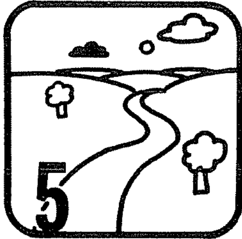
Without further discussion, let's look at what a complete version of this arithmetic quiz program might look like coded in BASIC. Notice that all our remark statements are still there, making it easy to put the working BASIC statements in their proper order.

```
10  REM---ARITH  (USES INPUT TO GET NUMS.)-----
100 REM---INTRODUCTION, INITIALIZE VARIABLES---
110 PRINT"ADDITION PRACTICE"
120 LET R = 0
130 PRINT "HOW MANY PROBLEMS DO YOU WANT";
140 INPUT N
200 REM---GENERATE PROBLEM---------------------
205 FOR I = 1 TO N
210   PRINT "TYPE IN 2 NUMBERS SEPARATED BY A COMMA";
220   INPUT A,B
300   REM---PRESENT PROBLEM--------------------
310   PRINT A;"+";B;"=  ";
400   REM---INPUT ANSWER-----------------------
410   INPUT X
500   REM---CHECK IF ANSWER IS RIGHT/WRONG-----
510   IF X = A + B THEN 550
520     PRINT "WRONG.  THE ANSWER IS";A+B
530     GOTO 610
550   PRINT "RIGHT!!"
560   LET R = R + 1
600 REM---GET NEXT PROBLEM---------------------
610 NEXT I
700 REM---REPORT SCORE-------------------------
710 PRINT "YOUR SCORE IS ";R/N*100;"% RIGHT."
800 REM---FUTURE EXTENSION OF PROGRAM----------
900 END
```

*From now on REM(ARK) statements will be used to help explain what each part of the program does. These are not used by the computer, so typing them in is not necessary unless you want to save a documented form of the program.*

Since there are no new features of BASIC used in this program, let's go right on to the extensions or improvements that can be made.

**Future Extensions and Revisions**




At this point you might decide that asking the person for two numbers every time is boring. There must be a better way. As a matter of fact there are several. Here's one: You can put pairs of numbers for A and B in DATA statements and have the program read in two values every time it generates a problem. Of course, you should warn the person not to ask for more problems than there are pairs of values.

**Project** *DATARITH*

Write a program that modifies ARITH to use DATA statements as the source of numbers for the problems. It is only necessary to change a few lines in ARITH and add the DATA statements, as follows:

```
10  REM---DATARITH  (USES READ,DATA FOR NUMS.)---
100 REM---INTRODUCTION, INITIALIZE VARIABLES---
110 PRINT "ADDITION PRACTICE"
120 LET R = 0
130 PRINT "HOW MANY PROBLEMS (UP TO 10)";
140 INPUT N
150 IF N > 10 THEN 130
200 REM---GENERATE PROBLEM--------------------
205 FOR I = 1 TO N
220   READ A,B
230   DATA 2,3, 3,7, 7,9, 9,12, 12,5
240   DATA 12,7, 13,7, 13,8, 14,9, 15,20
          .
          .
          .
610 NEXT I
```

**Project** *DATARIT2*

Children like hearing the same story over and over. A lot of youngsters like seeing the same arithmetic problems over and over, too. Write a program that uses READ and DATA statements as in the program segment given above. Then, to make it possible for your program to start again at the beginning of the data without having to rerun it, the keyword RESTORE should be used before branching back to the beginning of the program. The statement "840 RESTORE" means "Reset the READ pointer back to the beginning of the DATA."

```
800 REM---FUTURE EXTENSION OF PROGRAM----------
810 PRINT "WANT TO TRY THE SAME PROBLEMS AGAIN
    (Y = YES)";
820 INPUT A$
830  IF A$< > "Y"THEN 900
840 RESTORE
850 GOTO 140
900 END
```

**Project** *DATARIT3*

In DATARITH, if line 150 is omitted, and the user asks for more than 10 problems, the program will terminate with an OUT OF DATA error message when an attempt is made to read an 11th pair. There are at least two ways to cure this limitation: (1) store lots of values in your data (100 pairs?) and hope for the best, or more logically, (2) use the RESTORE keyword whenever the last data pair is read. How do you get the program to **know** when the last pair has been read? Here's one way. Add the following:

```
225 IF INT(I/10) = I/10 THEN RESTORE
```

And here's another way:

```
240 DATA 12,7, 13,7, 13,8, 14,9, 15,20, 0,0
225 IF A = 0 THEN RESTORE: I = I - 1: GOTO 610
```

Naturally, if you change the number of pairs of values you have in the DATA statements, you must change the "10" in the first version of line 225 to whatever number of pairs of values you actually have. If you change your DATA statements and use the second method, you must remember always to make the last pair of values 0,0. Why two zeros? Because the READ statement in this particular program will look for two more values. If it only finds one, you'll still get an OUT OF DATA error message.

**Project** *DATARIT4*

Suppose *DATARITH* had lots of pairs of values, and they were carefully arranged in order of increasing difficulty like this:

```
230  DATA 2,2, 3,2, 1,4, 5,5, 6,2
235  DATA 3,7, 4,5, 8,6, 4,7, 5,8
240  DATA 9,6, 9,9, 10,7, 12,8, 13,6
245  DATA 14,8, 15,6, 15,9, 16,5, 16,9
250  DATA 17,5, 17,9, 18,2, 18,6, 19,5
255  DATA 19,8, 20,5, 21,8, 22,9, 23,8, 0,0
```

Could you change the program so that the person using it could choose easy, medium, or hard problems? This means you would have to get the program to start generating problems from pairs of values not at the

beginning of the data, but further down the list. This can be done by adding a "dummy" READ statement.

```
160  PRINT "HOW HARD? 0=EASY, 1=MEDIUM, 2=HARD";
170  INPUT X
180  IF X = 0 THEN 205
190  FOR I = 1 to X*10
192  READ D,D
194  NEXT I
```

The variable D is a dummy, that is, its value is never used. The FOR loop in line 190 causes X*10 *pairs* of values to be read into D. This causes the pointer which keeps track of what values have been used up to move down X*10 pairs. So the program will start generating problems from the 11th pair of values if X = 1, and from the 21st pair if X = 2.

### Project *RNDARITH*

Another way to get the computer to provide numbers for the problems is to use the RND(0) function. This can be done by eliminating lines 210 and 220 of ARITH and substituting lines which use INT(N), the integer function, along with RND(0) to create surprise numbers for A and B. But now you must decide how big (or small) you want these numbers to be. We suggest the following:

```
220 LET A = INT(RND(0) * 15 + 1)
230 LET B = INT(RND(0) * 15 + 1)
```

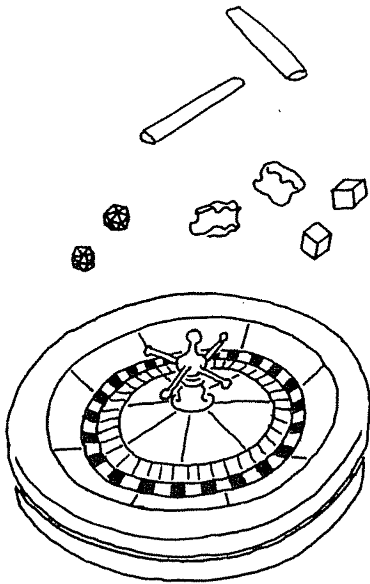
This will give numbers from 1 to 15 for A and B. If you want to have different numbers, you must change the formula while keeping in mind the following ideas:

- INT(...) means you want an integer, not a number with a fraction. Whatever is calculated inside ( ) will lose its fractional part.
- RND(0) gives a random number from 0 to .999999 (in TRS-80 Level II BASIC).
- RND(0) * 15 converts the random number to the range 0 to 14.999, so INT(RND(0) * 15) will give you one of the 15 integers from 0 to 14.
- INT(RND(0) * 15 + 1) means you prefer your random number to range over 15 integers from 1 to 15.

Now suppose you want your problems to have integers that start at 5 and end at 30. That means you want a total of 26 integers (30 - 5 + 1, where the + 1 is needed if you want to include 30). So your formula would read:

```
220 LET A = INT(RND(0) * 26 + 5)
```

It's a good idea to write some short programs which create random numbers with different ranges until you get used to the formula.



**Note on Random Numbers and the RND Function**

Many number and word games benefit from the use of the RND function in BASIC. The space game BABYQ explained in section 2.5 will also depend on RND to keep it interesting. This shouldn't be too surprising. Even the most ancient games used randomizing devices, ranging from sticks that could come to rest in only one of two ways when tossed (binary dice!), to the knuckle bones of sheep which could land in one of four positions. The six-sided cubic die used today was probably introduced by the Lydians of Asia Minor, while Roman dice with 14 and 20 sides have been found. In all cases the intent was to generate an outcome that could not be predicted exactly for each toss. Such an outcome is often called a *random* (as opposed to *deterministic*) event.

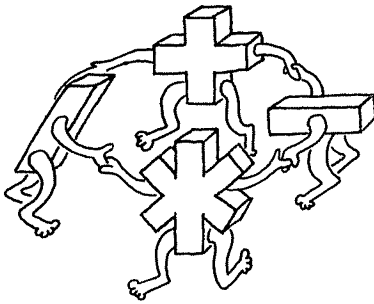
Even when events are random, we can say something about the probability of an individual event happening. If, for example, we find that in throwing one six-sided die 600 times each numbered face comes up about 100 times, we can say that the probability of getting a 1 is 1/6, the probability of getting a 2 is 1/6, the probability of getting a 3 is 1/6, and so on. We can also say that the numbers 1, 2, 3, 4, 5, and 6 are generated with a uniform distribution—no number gets favored over the others.

In BASIC, when you use a statement like LET R = RND(0) (**Warning**: on some machines LET R = RND(1) is the correct form), a number from 0 up to, but not including, 1 is generated. When a statement containing RND(0) is used more than once, or when it's part of a statement inside of a loop, different numbers may be generated each time the statement is executed. The numbers are said to be generated randomly because there doesn't seem to be any way to predict which number will appear next. Actually, the numbers are generated by a small subprogram inside BASIC that modifies the previous number each time it's used. Eventually, after much use, this subprogram will repeat its sequence of random numbers all over again. Since the output of the RND subprogram is deterministic, RND should really be called a pseudo-random number generator. In practice, you'll probably never know the difference, and you can assume that RND is giving you true random numbers with uniform distribution.

One last thing. On some computers the sequence of pseudo-random numbers generated by RND starts all over again each time you run the program. This can be useful when testing a program, but it could spoil the fun for certain games. If this happens on your machine, try putting the statement 10 RANDOM (or 10 RANDOMIZE) at the beginning of the program.

## Project *MULT*

Programs like ARITH, *DATARITH*, and *RNDARITH* can just as easily become quizzes on subtraction, multiplication, or division. But suppose that you want the program to say something besides RIGHT!! every time that the answer is correct. Try writing a multiplication quiz which randomly chooses to print RIGHT!!, CORRECT!!, or VERY GOOD!! when the answer is right. After line 530 and before line 600 you will need to do the following:



- Create a random number from 1 to 3; say, Z.
- Use a statement like: ON Z GOTO 550, 555, 560.
- At each line number mentioned above, print a message, then go on with the program.
- Remember to count right answers.

Surprise messages for the wrong answer are also possible. If the program is to be used among friends, some insults expressed in the latest slang might be fun, but you'd better not upset any strangers; they might mangle your microcomputer.

### **Project** *VARIETY1*

Write a math practice program which codes the following high-level design.

1. Introduction, initialize variables.
2. Ask the person how many problems are wanted.
3. Ask the person what kind: addition, subtraction, multiplication, or division.
4. Generate a problem; pick two random numbers, one ranging from 10 to 99, the other from 1 to 9.
5. Present the problem. Use four subroutines, one for each of the four operations, to present the problem and calculate the correct answer.
6. Check if the person's answer is "very close to" the correct answer. Print messages for right or wrong, count the right answers.
7. When finished, print the number correct.
8. Ask if the person wants more problems.

Some hints: Ask the person to type a number for the operation wanted, say, 1, 2, 3, or 4. Store this number in K, using INPUT K. Then use the statement

```
190 ON K GOSUB 1000, 2000, 3000, 4000
```

to branch to line 1000 if K = 1, line 2000 if K = 2, and so on.

The idea of "very close to" brings in the idea of a tolerance. *Tolerance* is some small amount, like .1, by which the answer is allowed to differ from the correct answer. Since it doesn't matter if the answer if greater or smaller than the correct one, we can use the absolute value function in constructing our conditional. If A holds the correct answer, and N is the person's answer, a test for "0.1 closeness" would look like this:

```
200 IF ABS(N - A) < .1 THEN 300
```

### **Project** *VARIETY2*

Extend the VARIETY1 program in the following ways (expressed as a modified high-level design):

1, 2, 3. The same as in *VARIETY1*.

4. Generate a problem, ask the person to say how many digits

are wanted for each of the two numbers, then pick two random numbers with that many digits.

5. Present the problem. In each subroutine set a variable T to a specific tolerance.
6. Check the person's answer, but use T instead of .1.
7, 8. The same as in *VARIETY1*.

## Project *SQROOT*

Very often in real life an arithmetically exact answer is not needed. Estimating quickly is a valuable skill. Quizzes can be written which reward fairly close answers to problems like: square roots; difficult addition, subtraction, multiplication, or division problems; addition of three or more large numbers; and so on. The definition of fairly close varies with each type of problem. In general, T should probably be expressed as a percentage of the answer, rather than a set amount.

Write a program that asks the person for the approximate square root of a randomly generated number and gives a congratulatory message that depends on how close the answer was to the correct one, but with the error expressed as a percent of the correct value.

## 2.2 TIME OUT TO LEARN SOME EXTENDED BASIC.
LRDEMO, VALDEMO, ASCDEMO, PRUDEMO, BUGPROG, *OKPROG*

The best known extended BASICs are those written by the Microsoft Company (including TRS-80 Level II BASIC, APPLESOFT BASIC, PET BASIC, and many others). Actually there are three sizes of Microsoft BASIC, often referred to as 8 K, 12 K, and 18 K disk extended BASIC. Chapter 4 explains what these terms mean. Since the features of 8K extended BASIC are found within the bigger versions, we will concentrate mostly on the 8K features. Here are the most useful ones:

### Multiple Statements on One Line

You can put several statements on one line of extended BASIC, using the colon to separate statements. You can also omit the key word LET any time you wish. For example instead of

```
10 LET A = 49
20 LET B = 68
30 LET C = -12
```

you can write

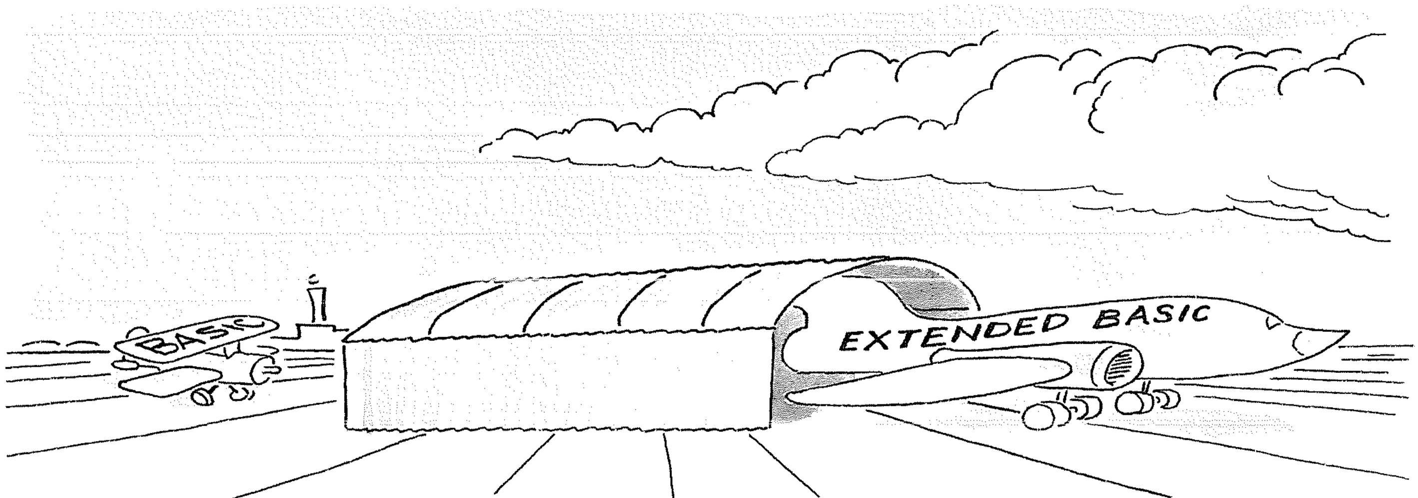
```
10 A = 49: B = 68: C = -12
```

A good way to read the colon here is as the words "and also," so this statement says "let A = 49 and also let B = 68 and also let C = −12."

Here's a one-line FOR loop that uses the colon. It stores the numbers from 1 to 10 in the locations A(1), A(2), A(3), and so on up to A(10):

```
95 FOR K = 1 TO 10: A(K) = K: NEXT K
```

The use of the colon should not be overdone. Only put statements on the same line that are related to the same task; otherwise your program will become very hard to read.



### The Extended IF Statement

The IF statement can be extended in three ways. The first is to allow one or more statements after the word THEN, instead of a line number. Here are two examples:

```
15 IF X > 10 THEN LET Y = 5 * 10
25 IF E = 0 THEN PRINT "OUT OF ENERGY": LET E = 100: GOTO 99
```

In the second example the colon is used to link together three different statements which are to be executed when the condition E = 0 is true. When the condition is false, none of these statements are executed; the program goes on to the next numbered statement.

The second extension of IF. . . THEN uses the word ELSE to show what should be done when a condition is false. Again, colons can be used to group several statements together.

```
350 IF X>9 THEN PRINT "TOO LARGE": X=X-1   ELSE
    PRINT "RESULTS O.K.": S=S+X
```

The third extension is to allow several conditions to be used in the conditional part of the IF statement, provided they are joined by the "logical" connectives AND, OR, or NOT. For example:

```
50 IF X = 9 AND Y = 9 THEN 2000
```

This means that if both conditions are true then branch to line 2000.

```
60 IF E < 20 OR T > 99 THEN 3000
```

This means that if either condition is true then branch to line 3000.

```
70 IF NOT (X = 9 AND Y = 9) THEN 4000
```

This means that if it's not true that both X = 9 and Y = 9, then branch to line 4000.

### What To Do If You Don't Have Extended BASIC

Each of the extended statements we've shown so far can be rewritten in minimal BASIC by using several simpler statements. Here are some examples showing how the extended statements just shown can be translated back to minimal BASIC.

Undoing multiple statements per line is easy. Just write a new line for each colon (:). For example, the FOR loop we showed as line 95 becomes:

```
95  FOR K = 1 TO 10
100   A(K) = K
105 NEXT K
```

The IF. . . THEN followed by multiple statements (line 25 above) is a little trickier to translate.

```
25  IF E = 0 THEN 50
30  REM --------STATEMENTS FOR E NOT=0 GO HERE --------
35  GOTO 99
40  REM --------------------------------------------------------------------
```

```
50  PRINT "OUT OF ENERGY"
55  LET E = 100
99  REM ---------------CONTINUE PROGRAM HERE ---------------
```

The IF. . . THEN. . . ELSE statement (line 350 above) gets translated in a similar manner.

```
350  IF X > 9 THEN 380
355  REM--------------- ELSE BLOCK ------
360  PRINT "RESULTS O.K."
365  LET X = S + X
370  GOTO 400
375  '
380  REM--------------- THEN BLOCK ------
385  PRINT "TOO LARGE"
390  LET X = X - 1
400  REM--------------- CONTINUE PROGRAM ------
```

The apostrophe used in 375 ' is a shorthand for REM allowed in some BASICs. We used it just to space things out for readability. You can also use 375 REM, or just omit this line.

Here's how the AND connective (line 50) is translated:

```
50  IF X = 9 THEN 70
60  GOTO 80
70  IF Y = 9 THEN 2000
80  REM---AT LEAST ONE CONDITION NOT TRUE IF
AT THIS  LINE--- (etc.)
2000 REM---BOTH CONDITIONS TRUE IF AT THIS LINE---
(etc.)
```

The OR condition (line 60) is translated as follows:

```
60 IF E < 20 THEN 3000
65 IF T > 99 THEN 3000
```

The logical NOT can be eliminated by using the complementary condition:

```
NOT X = 9 becomes X <> 9
NOT X < 9 becomes X >= 9
NOT X > 9 becomes X <= 9
```

and so on. When NOT modifies a group of conditions (as in line 70), then some laws of logic (De Morgan's laws) tell us that we must use the complementary conditions and also interchange AND with OR and vice versa. So

```
70 IF NOT (X = 9 AND Y = 9) THEN 4000
```

should be rewritten as

```
70 IF X <> 9 OR Y <> 9 THEN 4000
```

This can then be written in minimal BASIC as

```
70 IF X <> 9 THEN 4000
57 IF Y <> 9 THEN 4000
```

### What To Do If You Run Out of Memory

The programs shown so far are small enough to fit in the memory of just about any machine. Some of the programs ahead are longer, however, and a machine with 4 K bytes (K = 1024, so 4 K = 4096) of memory may not be adequate. The most obvious solution is to buy more memory, and this kind of expansion is discussed in Chapter 4.

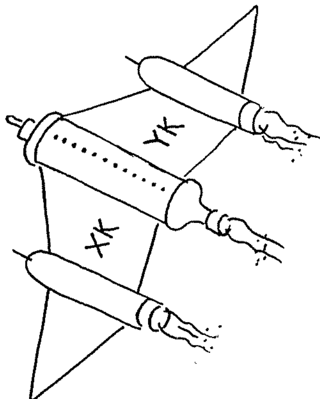
It's also possible to reduce the memory required by a program by "condensing" it as follows:

1. Remove all REMARK (REM or ') statements, and remove the unnecessary spaces from other statements.
2. Use the colon to squeeze several statements on a line, and omit unnecessary keywords such as LET, or THEN in THEN PRINT.
3. Reduce the size of arrays. Instead of DIM A(200), try to get along with A(50). Of course this will also mean simplifying the program to use a more limited range of subscripts.

### Longer Variable Names

Extended BASIC allows up to eight letters for variable names, provided the first two letters are unique. For example, you could refer to the X and Y coordinates of a space ship called Entelechy with the variables XENTEL and YENTEL, while you could call the coordinates of the evil Klipton ship XKLIPT and YKLIPT. That way it's much easier to remember what these variables stand for. Of course that's a lot of typing, so the variable names XE, YE, XK, and YK are perhaps a better choice.

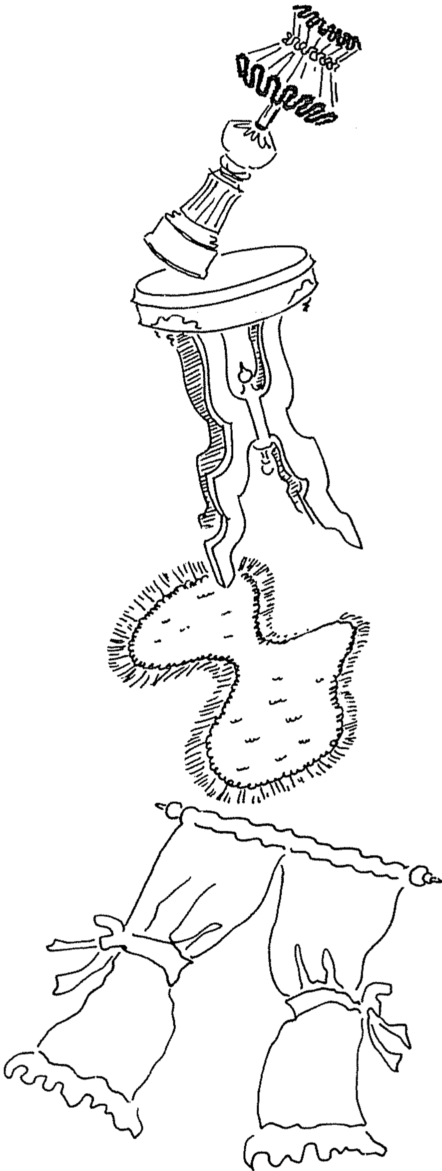
The longer variable names must begin with a letter, and then use only numbers or letters. They must also avoid the reserved words of BASIC (FOR, LET, PRINT, etc). For example, the name SHIFT would cause trouble because it contains IF.




### String Arrays

Let's first review the idea of *numerical* arrays. In section 1.6 of Chapter 1, the subscripted variables D(1), D(2), ..., D(30) are used to mean the sales of a business on day 1, day 2, ..., day 30. This means that the sales data was stored in 30 consecutive locations that occupy a block of memory. We sometimes say that the data in this block is related or structured. All the entries are sales, and adjacent memory locations represent adjacent days on which the sales took place. This kind of structure goes under several names: *array, linear array, one-dimensional array, vector, one-dimensional table,* or simply *table.*

We've also seen that BASIC allows *two-dimensional arrays* of numerical data, also called *two-dimensional tables,* or *matrices.* For example, suppose your store sold three flavors of ice cream, and you wanted to keep tabs on the sales of each flavor for the 30 days of the month. Then a natural way to organize or structure this numerical data is to store it in a table with three columns, one for each flavor, and thirty rows, one for each day, using an array of the form S(I,J), where I references a flavor, and J a day.



In extended BASIC you can also store *string* data in arrays, provided the array name ends in a dollar sign ($). Most extended BASICs allow both one-dimensional string arrays, with variable names like N$(K), and two-dimensional string arrays, with variable names like A$(I,J). The idea is that you can store blocks of strings together, where each string can usually be anywhere from 0 to 255 characters long. So N$(1) might contain the 3-character string "JOE," while N$(2) might contain the 14-character string "J.R. SMYTH II." (However, in some versions of BASIC, N$(K) means "the Kth character in the string N$.") Examples of how one-dimensional string arrays can be used are shown in the programs SALESLIP (section 3.5) and MONEY (section 3.6). For example, in SALESLIP the array D$(I) is used to hold descriptions of items sold in a store. So it can be pictured as a block like the following:

| | |
|---|---|
| D$(1) | RED LAMP |
| D$(2) | SMALL TABLE |
| D$(3) | WOOL RUG |
| D$(4) | WHITE DRAPES |
| ... | ...etc... |

An example showing the use of a two-dimensional string array is found in the program BABYQ (section 2.5). There the array A$(I,J) is used to hold the symbols displayed on a 9 by 9 game board.

### String Functions

Strings can be combined by using the + operator. This is called concatenation. If A$ = "SNOW" and B$ = "FLAKE" then PRINT A$ + B$ will produce "SNOWFLAKE." To do fancier things with strings, extended BASIC also has a number of string functions. Here are some examples showing how these functions work.

LEFT$(A$,M) gives the leftmost M characters of A$, while RIGHT$(A$,N) gives the rightmost N characters. Try this demonstration program.

```
10 REM ----- LRDEMO -----
20 A$="SNOW"
30 B$="FLAKE"
40 PRINT LEFT$(A$,2) + RIGHT$(B$,3)
50 END

RUN
SNAKE
```

MID$(A$,P,N) gives you N characters from the "middle" of A$ starting at position P. So PRINT MID$("SNOWING",2,3) would give you NOW. Actually P can be any position, including P = 1, so the word middle is a little deceptive.

LEN(B$) gives you the length of B$ (which means the number of characters in B$). LEN("SNOW") is 4 and LEN("FLAKE") is 5.

VAL(N$) converts a string made up of decimal digits into a genuine number. This is important when you want to do arithmetic with N$. Here's an example:

```
10 REM ----- VALDEMO -----
20 A$="42 CENTS"
30 C$=LEFT$(A$,2)
40 C=VAL(C$)/100
50 PRINT A$;" = $";C
60 PRINT "3 ITEMS AT ";A$;" COST";3*C
70 END

RUN
42 CENTS = $ .42
3 ITEMS AT 42 CENTS COST 1.26
```

ASC(A$) is the ASCII code function. Each character in a string is represented inside the computer as a number called its ASCII code. ASCII means American Standard Code for Information Interchange. To find what this code is and perhaps do something with it, extended BASIC uses the ASC(A$) function. Since only one code can be found at a time, ASC("APPLE") will only give you the code for the first letter, A, which is 65 in decimal notation. So ASC("APPLE"), ASC("ART") and ASC("A") all return the value 65. To get each of the individual ASCII codes for the characters in a string, you can combine ASC with MID$ as follows:

```
10 REM ----- ASCDEMO -----
20 A$="ABCDE"
30 FOR P=1 TO 5
40   PRINT ASC(MID$(A$,P,1));
50 NEXT P
60 END

RUN
 65  66  67  68  69
```

CHR$(X) is a function which gives you the character corresponding to the ASCII code X. Thus PRINT CHR$(65) would print the character A. *Note:* The ASCII codes from 1 to 32 are used for special control purposes, and do not correspond to printable characters. The printable codes go from 32 to 127. To see what they are, run this program:

```
10 FOR X = 32 to 127
20 PRINT X, CHR$(X)
30 NEXT X
```

The ASCII codes are summarized in Appendix B.

Longer examples showing how string functions can be used will be found in the programs *CIPHER* (section 2.4), MONEY (section 3.6), and *EDIT5000* (section 4.5).

## PRINT USING

This special form of the print statement allows you to control exactly how many digits are printed for numerical quantities, and how many characters are printed for strings. It also lets you control exact spacing between items on an output line. For example, suppose Q represents the quantity of an item that costs C dollars, and D$ holds a description of the item. The following demonstration program shows the difference between PRINT and PRINT USING:

```
10 REM ----- PRUDEMO (PRINT USING) -----
15 LET Q=42
20 LET C=100 - 100/3
25 LET D$="ASH TRAY, SILVER"
30 PRINT "OUTPUT WITH ORDINARY PRINT"
35 PRINT Q, D$, C
40 PRINT "...................................................."
45 PRINT "OUTPUT WITH PRINT USING"
50 F$="QTY: ###        ITEM: %      %        COST: $###.##"
55 PRINT USING F$; Q, D$, C
60 END

RUN
OUTPUT WITH ORDINARY PRINT
 42             ASH TRAY, SILVER                 66.6667
....................................................
OUTPUT WITH PRINT USING
QTY:  42        ITEM: ASH TRAY       COST: $ 66.67
```

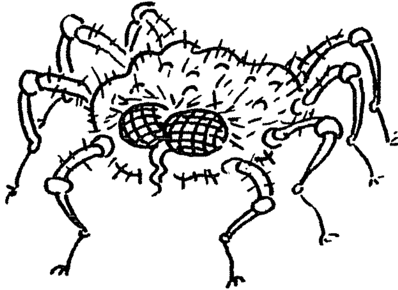
The format F$ is a string that gives "a picture" of the output line. The symbol "###" is a temporary place holder in this string for a 3-digit number, "%      %" is a place holder for an 8-character string, and "###.##" is a place holder for a number which will print in a format that has 3 digits before the decimal, and exactly 2 after the decimal. You control how many digits are printed by the number of # symbols you use. Lots of examples of PRINT USING are given in Chapters 3 and 4 in connection with business application programs. Incidently, although it's hard to see, you should notice that the "%      %" in our format has 6 spaces between the % signs to specify a total of 8 positions.

### CLEAR 500

This statement is used at the beginning of a program that expects to use lots of strings and/or lots of characters. BASIC normally lets you use a total of up to 50 characters in all the strings of a program. But if you use the statement 10 CLEAR 500, then your program will be able to use up to 500 locations of memory for strings, where each location called a byte of memory holds 1 character. You will know that you need this statement if you get the error message "OUT OF STRING SPACE." Don't confuse CLEAR with CLS which simply means "clear the screen" on the TRS-80.

### Direct Mode Computing; Debugging Programs

Many versions of BASIC allow you to write one line programs that execute immediately after you press ENTER or RETURN. These do not have line numbers, and they are not saved in memory. One use of such



programs is to use the computer as a calculator. For example, if you want the volume of a 14 inch diameter sphere using the formula 4/3 * PI * R * R * R just type

PRINT 4/3 * 3.1416 * 7 * 7 * 7 (ENTER)

and you'll get:

1436.76

To compute several quantities, a loop using the colon can be used:

FOR K = 1 TO 10: PRINT K * K;: NEXT K (ENTER)

1 4 9 16 25 36 49 64 81 100

When a program has bugs (errors), you can use direct mode after you run it to investigate what values the variables had. This often gives you an idea about what went wrong. For example, suppose you tried to write a program to compute the sum of the first N integers, but got an obviously wrong answer when you tried it for N = 5 (the answer should be 5 + 4 + 3 + 2 + 1 = 15).

```
10 REM ------------------ BUGPROG -----------------
15 DEFINT A-Z     'THIS MAKES ALL VARIABLES INTEGERS
20 PRINT "SUM OF POSITIVE INTEGERS FROM 1 TO";
30 INPUT N
40 S=1 : I=1
50 IF I>N THEN 90
60   I=I+1
70   S=S+I
80   GOTO 50
90 PRINT "SUM OF POSITIVE INTEGERS REQUESTED IS"; S

>RUN
SUM OF POSITIVE INTEGERS FROM 1 TO? 5
SUM OF POSITIVE INTEGERS REQUESTED IS 21
READY
>PRINT S, N, I
 21            5              6
READY
```

The run of BUGPROG above was followed by use of direct mode to "look at" S, N, and I. This showed that N was 5 but I was 6, and pinpointed the bug as the test at line 50 which should have been

```
50 IF I > = N THEN 90
```

### Project *OKPROG*

Make the change in line 50 suggested above, and then run the program with every possible input data you can think of. Is there any way you can make the program produce incorrect output? If not, you have what computer scientists call a "correct" program, a very rare bird. It is claimed by some that most programs in the world are not correct; that for some set of input data, the program will produce unexpected and sometimes disastrous results. Learning to have a healthy questioning attitude toward computer output may be one of the most valuable contributions personal computing can make to its practitioners.

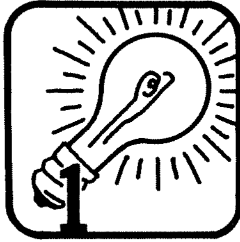
## 2.3 FINDING YOUR WAY AROUND NUMBERLAND.
GUESS1, *GUESS2, SRCH1, SRCH2, SRCH3*

When learning any new subject, there needs to be a period of free play, a chance to skim over the subject matter before trying to absorb the precise details. Some elementary math classes do this with concrete objects and games. But why stop there? This need for experimental fiddling recurs at higher levels, no matter how serious the new subject matter.

One of the most useful contributions computers can make is to allow people to do a lot of fiddling. In particular, computer games make a great medium for playing with mathematics, often at much deeper levels than the player suspects.

One of the simplest number games, "guess my number," illustrates this idea. It's a good way to fiddle with the elementary ideas of larger, smaller, and between, but it also involves some not-so-elementary ideas like "What's the most efficient way of finding a number, with the least number of guesses?", and "What *is* the least number of guesses?"

Let's first look at a simple number guessing game where both the person and the computer get their chance to guess a number. The computer will use a technique called binary search. Then we'll discuss how to apply this technique to the problem of searching for arbitrary data in an ordered list.



## Program GUESS1

### The Idea

The computer will randomly choose a number, let's say from 1 to 100. The person will try to guess it. There will be a message saying "Too high," or "Too low," or "Right, you got it." If the person is really bad at guessing, the program will end the guessing after, say, 10 turns and tell the person what the number was. The program then asks if the person wants to play again. If not, it goes on to the next part of the program.

This program is really going to be two programs in one. In the second part, the computer will guess a number that the person chooses. It will use a method based on the idea of binary search. Thus, it will give the person a hint of how to proceed the next time he or she tries to guess the computer's number.

### A Sample Run

In this sample run of both parts of the guessing game, you will notice that a number of additional features have appeared.

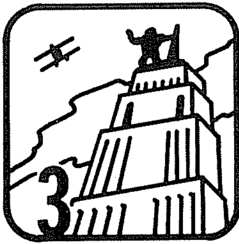
```
GUESS MY NUMBER GAME
I'VE GOT A NUMBER BETWEEN 1 AND 100.
YOUR GUESS IS--? 1000
MUST BE <= 100
YOUR GUESS IS--? -99
MUST BE >= 1
YOUR GUESS IS--? 99
NOPE, TOO HIGH.
YOUR GUESS IS--? 98
NOPE, TOO HIGH.
YOUR GUESS IS--? 97
NOPE, TOO HIGH.
YOUR GUESS IS--? 96
NOPE, TOO HIGH.
YOUR GUESS IS--? 95
NOPE, TOO HIGH.
YOUR GUESS IS--? 94
NOPE, TOO HIGH.
YOUR GUESS IS--? 93
NOPE, TOO HIGH.
YOUR GUESS IS--? 92
NOPE, TOO HIGH.
YOU HAVE HAD 10 TURNS, THAT'S THE LIMIT.
MY NUMBER WAS 78 .  WANT TO PLAY AGAIN (Y = YES)? NO
O.K.
NOW I WANT TO GUESS YOUR NUMBER.
YOU PICK A NUMBER FROM 1 TO 100,
```

```
DON'T TELL ME WHAT IT IS!
JUST TYPE H IF MY GUESS IS TOO HIGH,
L IF MY GUESS IS TOO LOW, AND R IF I GET IT RIGHT.
GOT A NUMBER (Y = YES)? Y
I GUESS... 50 , AM I -- H, L, OR R? H
I GUESS... 25 , AM I -- H, L, OR R? H
I GUESS... 12 , AM I -- H, L, OR R? H
I GUESS... 6 , AM I -- H, L, OR R? H
I GUESS... 3 , AM I -- H, L, OR R? H
I GUESS... 1 , AM I -- H, L, OR R? Z
YOU DIDN'T TYPE H, L, OR R? H
DID YOU GIVE ME THE RIGHT CLUES?
LET ME TRY AGAIN.
I GUESS... 50 , AM I -- H, L, OR R? H
I GUESS... 25 , AM I -- H, L, OR R? H
I GUESS... 12 , AM I -- H, L, OR R? H
I GUESS... 6 , AM I -- H, L, OR R? H
I GUESS... 3 , AM I -- H, L, OR R? H
I GUESS... 1 , AM I -- H, L, OR R? L
I GUESS... 2 , AM I -- H, L, OR R? R
I GOT IT IN 7 TURNS.
WANT TO PLAY AGAIN (Y = YES)? NO
O.K.  BYE NOW.
```

The new features are as follows:

- If the person types in a guess greater than 100 or less then 1, explanatory messages are printed out and the program branches back to the input request.
- When the right number is guessed, the number of turns it took is printed out.
- In the second part, new instructions on what to do are printed for the user.
- The program first asks if the person has a number. This gives time to think. Also if the person does not want to play, a response other than 'Y' will end the program.
- After each guess, there is a reminder that the person is expected to type "H", "L", or "R".
- If anything else is typed, an explanatory message is printed and the program goes back to the input statement.
- Finally, if the person gives inconsistent clues (out of confusion or trying to fool the computer), the program prints a message and starts the guessing all over.
- At the end of either part, the person is asked whether he wants to play again.

## High-Level Design




The high-level design for the two parts of this program could be organized as seven blocks each:

1. Give instructions, initialize variables (a counter, T).
2. Generate a random number, X, between 1 and 100.
3. Check to see if the player has taken too many turns.
4. Ask the player to guess; input the guess as G.
5. Test if person's guess is greater than, less than, or equal to the random number, or if it's illegal. Print messages for each possibility and go to the appropriate next step.
6. When too many turns have been taken, give the answer.
7. Ask if the person wants to play again; if yes, go to 2, otherwise go to 8.
8. Computer guesses number: give instruction, initialize variables (turn counter T, high H, low L).
9. Ask player if he has picked a number; if not, end the program.
10. Generate computer's guess, (see explanation of algorithm in **Coding the Program** section.)
11. Ask for the player's clue.
12. Test if clue is L, H, R, or if it's illegal. Print messages for each possibility, and go to appropriate next step.
13. If clues are inconsistent print mesage and go to 9 (see **Coding** section for how the program knows the clue is inconsistent).
14. After the right answer is found, ask if person wants to play again; if yes, go back to 9.

## Coding the Program

Coding the first part of this program is pretty straightforward. The second part is a little trickier. It must generate a series of guesses which get progressively closer to the person's secret number. If you examine the series of guesses in the sample run of the second part of the program you will see what the program's strategy is.

At the beginning of the game, all the program "knows" is that the highest the secret number could be is 100 and the lowest it could be is 1. It uses a binary search strategy to zoom in on the exact number. It starts by making the first guess a number half-way between the highest and the lowest possibilities. So the first guess is 50. If it's told this number is right, it's finished! If it is given the clue "Too high," then the highest the number could be is 49. If the clue is "Too low," the lowest it could be is 51. The next guess should be halfway between the new highest and lowest values—1 and 49, or 51 and 100, respectively. That means the second guess should be 25 or 75. Highest and lowest values will keep getting closer

together as the program zooms in on the secret number X.

For example, if the second guess is not right, here is a chart showing what the third guess will be for each possible clue.

| SECOND GUESS | CLUE | HIGHEST X CAN BE | LOWEST X CAN BE | THIRD GUESS |
|---|---|---|---|---|
| 25 | Too High | 24 | 1 | 12 |
| 25 | Too low | 49 | 26 | 37 |
| 75 | Too high | 74 | 51 | 62 |
| 75 | Too low | 100 | 76 | 88 |

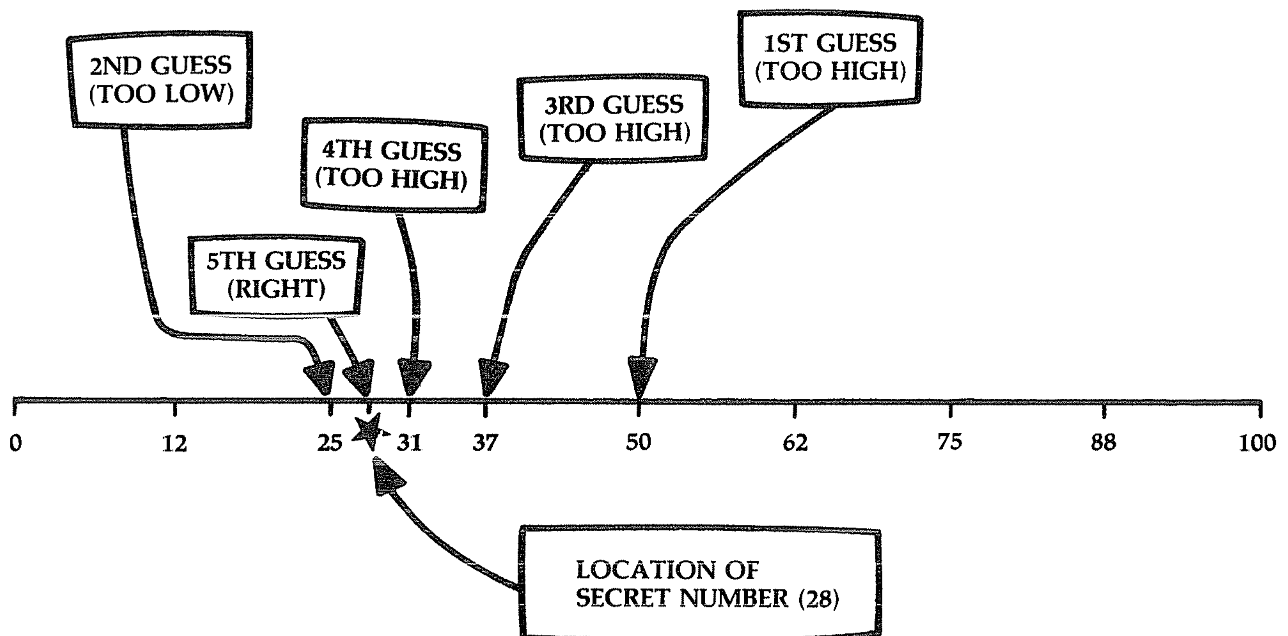
The formula for calculating a guess halfway between the highest and lowest possible values is:

G = (H + L)/2

This gives values of 12.5, 37.5, 62.5, and 88 for the third guess. Since we want a whole number for the guess, we can use the INT function to get the greatest integer less than this expression by using:

G = INT( (H + L)/2)

For each new clue, the L and H values get closer together. If the lowest value gets to be greater than the highest, the program "knows" something is wrong! The clues were inconsistent. This won't happen in the first part where the program is giving out the clues, but it might happen in the second part when the person is giving the clues. It turns out that the best place to put an IF statement testing for inconsistent clues (L > H) is just before the statement that creates the new guess (at line 1230). (Readers who have studied other languages will recognize that this gives a WHILE structure. The block 1240-1350 is executed over and over WHILE L < = H.)



```
10   '**************************************************
11   '*          GUESS1  (USES BINARY SEARCH)           *
12   '**************************************************
100  '--------------------------------------------------
101  '     COMPUTER CHOOSES NUMBER, PERSON GUESSES
102  '--------------------------------------------------
105 CLS
110 PRINT"GUESS MY NUMBER GAME"
120 PRINT"I'VE GOT A NUMBER BETWEEN 1 AND 100."
130 T = 1
200 '-----GENERATE RANDOM NUMBER------------------------
210 X = INT(RND(0)*100 + 1)
300 '-----ASK PLAYER TO GUESS---------------------------
310 IF T > 10 THEN 510
320   PRINT"YOUR GUESS IS--";: INPUT G
400 '-----TEST IF GUESS IS ILLEGAL, >, <, OR = ---------
410   IF G > 100  PRINT "MUST BE <= 100": GOTO 460
420     IF G < 1  PRINT "MUST BE >= 1": GOTO 460
430       IF G > X PRINT "NOPE, TOO HIGH.": GOTO 460
440         IF G < X PRINT "NOPE, TOO LOW.": GOTO 460
450           PRINT"YOU GOT IT IN";T;"TURNS!!!  ";: GOTO 610
460   T = T + 1: GOTO 310
500 '-----TOO MANY TURNS, GIVE ANSWER-------------------
510 PRINT"YOU HAVE HAD";T-1;"TURNS, THAT'S THE LIMIT."
520 PRINT"MY NUMBER WAS";X;".  ";
600 '-----AGAIN?----------------------------------------
610 PRINT"WANT TO PLAY AGAIN (Y = YES)";: A$=" ": INPUT A$
620 IF A$ = "Y" THEN 105
1000 '--------------------------------------------------
1001 '     PERSON CHOOSES NUMBER, COMPUTER GUESSES
1002 '--------------------------------------------------
1005 CLS: PRINT"O.K."
1010 PRINT"NOW I WANT TO GUESS YOUR NUMBER."
1020 PRINT"YOU PICK A NUMBER FROM 1 TO 100,"
1030 PRINT"DON'T TELL ME WHAT IT IS!"
1040 PRINT"JUST TYPE H IF MY GUESS IS TOO HIGH,"
1050 PRINT"L IF MY GUESS IS TOO LOW, AND R IF I GET IT RIGHT."
1100 '-----ASK PLAYER TO PICK NUMBER---------------------
1110 PRINT"GOT A NUMBER (Y = YES)";: A$=" ": INPUT A$
1120 IF A$ <> "Y" THEN 1530
1200 '-----GENERATE COMPUTER'S GUESS--------------------
1210 L = 1: H = 100: T = 0
1230 IF L > H THEN 1410
1240   G = INT((L+H)/2)
1250   PRINT"I GUESS...";G;", AM I -- H, L, OR R";
1260   T = T+1
1300 '-----GET CLUE, TEST IF H, L, R, OR ILLEGAL--------
1310   INPUT C$
1320   IF C$ = "H" THEN H = G-1: GOTO 1230
1330     IF C$ = "L" THEN L = G+1: GOTO 1230
1340       IF C$ = "R" PRINT"I GOT IT IN";T;"TURNS.":GOTO 1510
1350         PRINT"YOU DIDN'T TYPE H, L, OR R";:GOTO 1310
1400 '-----NOT FOUND (CLUE IS INCONSISTENT)-------------
1410 PRINT"DID YOU GIVE ME THE RIGHT CLUES?"
1420 PRINT"LET ME TRY AGAIN. ": GOTO 1210
1500 '-----AGAIN?---------------------------------------
1510 PRINT"WANT TO PLAY AGAIN (y = YES)";: A$=" ": INPUT A$
1520 IF A$ = "Y" THEN 1110
1530 PRINT"O.K.  BYE NOW."
9999 END
```

**Project** *GUESS2*

Write a program similar to GUESS 1 in which the range of numbers, instead of always being 1 to 100, can be chosen by the player. Lines like the following would be added:

```
120 PRINT "WHAT'S THE HIGHEST THE NUMBER CAN BE";: INPUT H
130 PRINT "WHAT'S THE LOWEST THE NUMBER CAN BE";: INPUT L
140 PRINT "I'VE GOT A NUMBER BETWEEN";L;"AND";H
```

Line 210 must also be changed (see project *RNDARITH*, section 2.1).

In line 405, the number of turns allowed should really change, depending on the range of numbers. When using the binary search technique, you can determine the greatest number of guesses needed for any range of numbers by determining which "powers of 2" the range (the number of numbers) falls between, and choosing the higher power. Some "powers of 2" are as follows:

| | | |
|---|---|---|
| $2^0 = 1$ | $2^5 = 32$ | $2^{10} = 1024$ |
| $2^1 = 2$ | $2^6 = 64$ | $2^{11} = 2048$ |
| $2^2 = 4$ | $2^7 = 128$ | $2^{12} = 4096$ |
| $2^3 = 8$ | $2^8 = 256$ | $2^{13} = 8192$ |
| $2^4 = 16$ | $2^9 = 512$ | $2^{14} = 16384$ |

The last number in this table indicates that binary search would take 14 tries at most to find a given integer in a set of anywhere from 8192 to 16383 consecutive ordered integers.

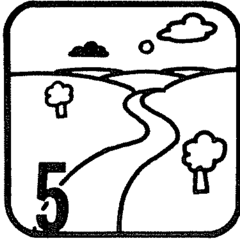
In GUESS1, the range contained one hundred numbers which went from 1 to 100, so the greatest number of guesses required was seven (the actual number depends on the secret number). Notice that if the range contained the one hundred numbers from 200 to 300, the maximum would still be seven guesses; it's the number of numbers that counts. So if you play the game specifying one thousand numbers from 1500 to 2500, it will take ten guesses at most to guess the number. A program can calculate this maximum number of guesses by comparing the size of the range with increasing powers of 2. Another approach is to use the LOG(X) function (see the text box, **Note on Using Logarithms to Analyze a Binary Search**).

If you add this feature to your program and find the greatest number of guesses, T1, needed, you can then limit the number of turns the person can have to, say, T1 + 2. You can also have the computer comment whenever it starts to guess the person's number:

I THINK I CAN GUESS IT IN ... TURNS.

where ... is the value of T1.

In line 1210 new variable names L1 and H1 will be needed, since L and H are now used for the variable starting values. Corresponding changes must be made in 1230, 1240, 1320, and 1330.




### Future Extensions and Revisions

A more usual application of the binary search method is to rapidly look through ordered lists of data items, stored in arrays or on files, for some specific data item. The item you are trying to find is called the key. It may or may not be in the list. The data must be in order, that is, have already been sorted, to make the binary search method work. (Why?) The other method commonly used to find an item of data is the sequential search method. This means simply to start at the beginning of the list, comparing the key item with each data item in turn until the match is found or the list is exhausted. This method works on sorted or unsorted data, but it's slow for large lists.

### Project *SRCH1*

Write a program that uses the binary search method to count how many times certain "standard" words occur in a text (sentence, paragraph, etc) typed in by you. The program should first read an ordered list of the standard words from data statements into an array, A$(100). As an example of a standard list, you can use the following data taken from a recent study of school books (grades 3 through 9). This list represents the 100 most frequently used words found in these books. The list has already been alphabetized to enable the binary search method to work. It consists of the word, followed by its rank (its "popularity" in school books.).

```
240 DATA A, 4,         ABOUT, 48,   AFTER, 94,   ALL, 33
242 DATA AN, 39,       AND, 3,      ARE, 15,     AS, 16
244 DATA AT, 20,       BE, 21,      BEEN, 75,    BUT, 31
246 DATA BY, 27,       CALLED, 96,  CAN, 38,     COULD, 70
248 DATA DID, 83,      DO, 45,      DOWN, 84,    EACH, 47
250 DATA FIND, 87,     FIRST, 74,   FOR, 12,     FROM, 23
252 DATA HAD, 29,      HAS, 62,     HAVE, 25,    HE, 11
254 DATA HER, 64,      HIM, 67,     HIS, 18,     HOW, 49
256 DATA I, 24,        IF, 44,      IN, 6,       INTO, 61
258 DATA 1S, 7,        IT, 10,      ITS, 76,     JUST, 97
260 DATA KNOW, 100,    LIKE, 66,    LITTLE, 92,  LONG, 91
262 DATA MADE, 81,     MAKE, 72,    MANY, 55,    MAY, 89
264 DATA MORE, 63      MOST, 99,    MY, 80,      NO. 71
266 DATA NOT, 30,      NOW, 78,     OF, 2,       ON, 14
268 DATA ONE, 28,      ONLY, 85,    OR, 26,      OTHER, 60
270 DATA OUT, 51,      OVER, 82,    PEOPLE, 79,  SAID, 43
```

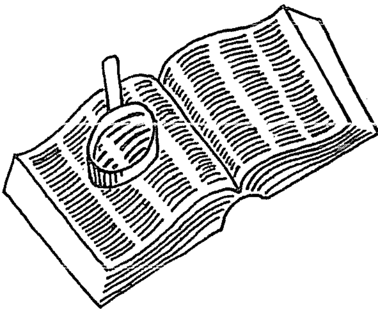
```
272 DATA SEE, 68,     SHE, 54,     SO, 57,      SOME, 56
274 DATA THAN, 73,    THAT, 9,     THE, 1,      THEIR,42
276 DATA THEM, 52,    THEN, 53,    THERE, 37,   THESE, 58
278 DATA THEY, 19,    THIS, 22,    TIME, 69,    TO, 5
280 DATA TWO, 65,     UP, 50,      USE, 88,     VERY, 93
282 DATA WAS, 13,     WATER, 90,   WAY, 86,     WE, 36
284 DATA WERE, 34,    WHAT, 32,    WHEN, 35,    WHERE, 98
286 DATA WHICH, 41,   WHO, 77,     WILL, 46,    WITH, 17
288 DATA WORDS, 95,   WOULD, 59,   YOU, 8,      YOUR, 40
```

Next, the program should ask you to input *your* text, one word at a time. As soon as you input a word, it should print the following information: the word, its position number in the alphabetized standard list (or the message "not found"), and how many times it has been used so far in your text. The program should tell you to type $$$ when finished with inputting. It should print as a final summary the whole standard list of words with two numbers next to each word—the number of times it was used in your text, and the rank.

Incidentally, the highest frequency words, found on standard lists, are mostly building block words: conjunctions, prepositions, pronouns, and very general adjectives and adverbs. The first 250 of these words often make up 80% of a text. By contrast, even very familiar concrete nouns and verbs occur much less frequently. Some researchers believe they can establish the possible authorship of anonymous texts by tabulating the occurrences of these building block words and comparing the totals to those from a known text. This is an example of a kind of research that is only feasible on a large scale with the help of computers.

### Project *SRCH2*

Modify *SRCH1* by substituting a sequential search algorithm for the binary search algorithm. Compare the times needed to find an item at the beginning of the list, in the middle, and at the end. What can you conclude about the two methods for a list of 50 words? For a list of 500 words?



### Project *SRCH3*

Modify *SRCH1* so that the text is stored in DATA statements and then read, instead of inputted. In Chapter 4 we'll talk about using disk data files, and a good extension of *SRCH3* would be to replace the READ and DATA statements by the use of the GET (from a file) statement.

## NOTE ON USING LOGARITHMS TO ANALYZE A BINARY SEARCH

The maximum number of guesses needed to find X in the range L < = X < = H by binary search can be obtained by asking what power of 2 the quantity H − L + 1 is "closest" to. To see what "closest" means, let's examine several ranges of numbers in the *worst case* situation where the answer is the highest number H, but the binary search continues to select the midpoint M.
For example:

| Range at Start = L, H: | 1,3 | 1,4 | 1,5 | 1,6 | 1,7 | 1,8 | 1,9 | 1,10 . . . |
|---|---|---|---|---|---|---|---|---|
| Guess = M: | 2 | 2 | 3 | 3 | 4 | 4 | 5 | 5 . . . |
| New Range: | 3,3 | 3,4 | 4,5 | 4,6 | 5,7 | 5,8 | 6,9 | 6,10 . . . |
| Guess = M: | ③ | 3 | 4 | 5 | 6 | 6 | 7 | 8 . . . |
| New Range: | - | 4,4 | 5,5 | 6,6 | 7,7 | 7,8 | 8,9 | 9,10 . . . |
| Guess = M: | - | ④ | ⑤ | ⑥ | ⑦ | 7 | 8 | 9 . . . |
| New Range: | - | - | - | - | - | 8,8 | 9,9 | 10,10 . . . |
| Guess = M: | - | - | - | - | - | ⑧ | ⑨ | ⑩ . . . |

The circles show where we finally guess the answer H. Notice that for the ranges 1 to 4, 1 to 5, 1 to 6, and 1 to 7 it takes three guesses for this worst case situation. To say this another way, the worst case is three guesses when the range contains 4, 5, 6, or 7 numbers. Now, what might each of these numbers have in common? Well, 4 = 2↑2.0000, 5 = 2↑2.3219, 6 = 2↑2.5850, and 7 = 2↑2.8074. If we take each of these exponents, (the number after the ↑ symbol), add 1, and then take the integer part, we get 3. Aha! But how can we make a formula out of this? If we pull out our old Algebra II book, we find that another way to write 7 = 2↑2.8074 is

$$LOG_2(7) = 2.8074$$

So the maximum number of guesses needed for seven numbers is

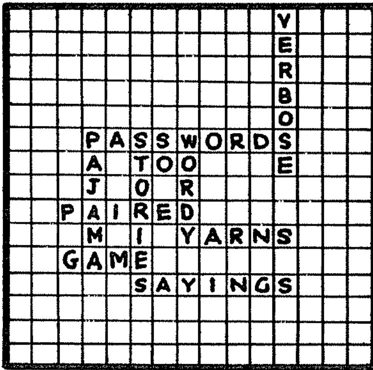
$$INT(LOG_2(7) + 1) = 3$$

In BASIC, there is a function called LOG(X) but it isn't base 2. But the same math book tells us that we can get a base 2 log by using LOG(X)/LOG(2). So the formula for the most guesses needed in a binary search is

INT(LOG(H − L + 1)/LOG(2) + 1)

where H − L + 1 says how many numbers are in the range H to L.

## 2.4 WORD GAMES. SCRAM1, *SCRAM2, SCRAM3, CIPHER*

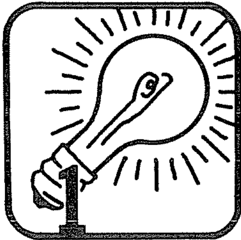



It comes as a surprise to many people that computers can be used to manipulate words as well as numbers. However, manipulating words is quite a different process from understanding them, and as we'll see in this section, it's the programmer who has to provide all the intelligence in word oriented programs.

The principal technique for doing this is to store word meanings right in the program, and then use IF statements to check these meanings against other data, usually a string that was input by the person using the program. A more flexible technique for storing word meanings is to use disk files, treating them as a kind of electronic dictionary. (Disk files will be discussed in Chapter 4.)

All the programs in this section use DATA statements to store words. However the last program, *CIPHER*, also introduces the technique of modifying words or messages by use of a simple mathematical transformation. This technique can be extended considerably, and there is now an active field of research investigating the properties of such transformations (see the box, **Note on Ciphers and Codes**).

### Program SCRAM1



#### The Idea

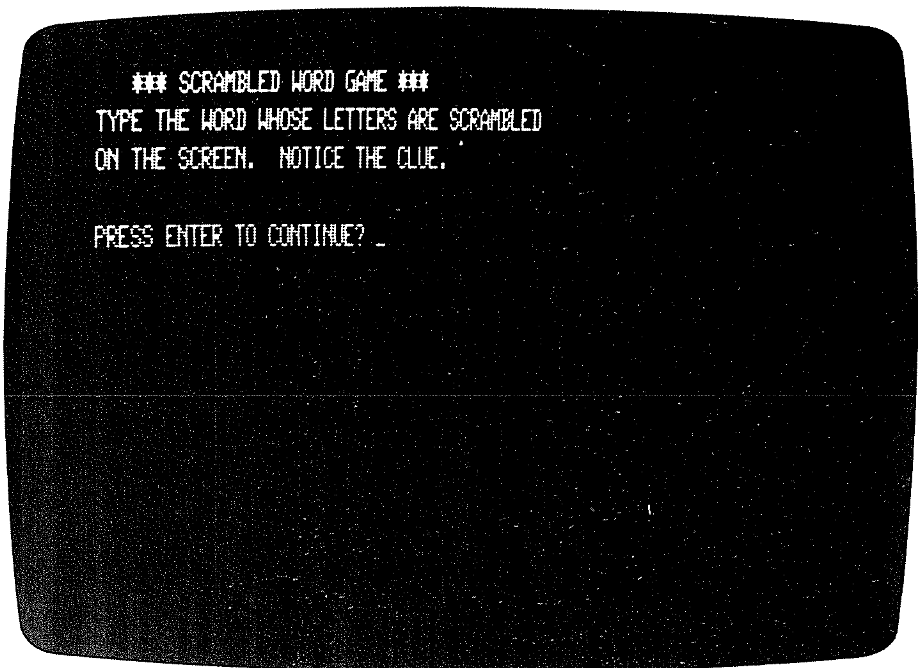
This program is a word quiz. The person will be asked how many words are wanted. The program will present a word to be guessed on the screen with the letters scrambled—and then make it disappear! A clue will also be given: a short definition, a synonym, or a phrase to be filled in. The person will be asked to type in the word. If the answer is wrong, the person gets a second chance and the word is displayed again—and disappears again. When all the words have been presented, the number right and wrong will be displayed.

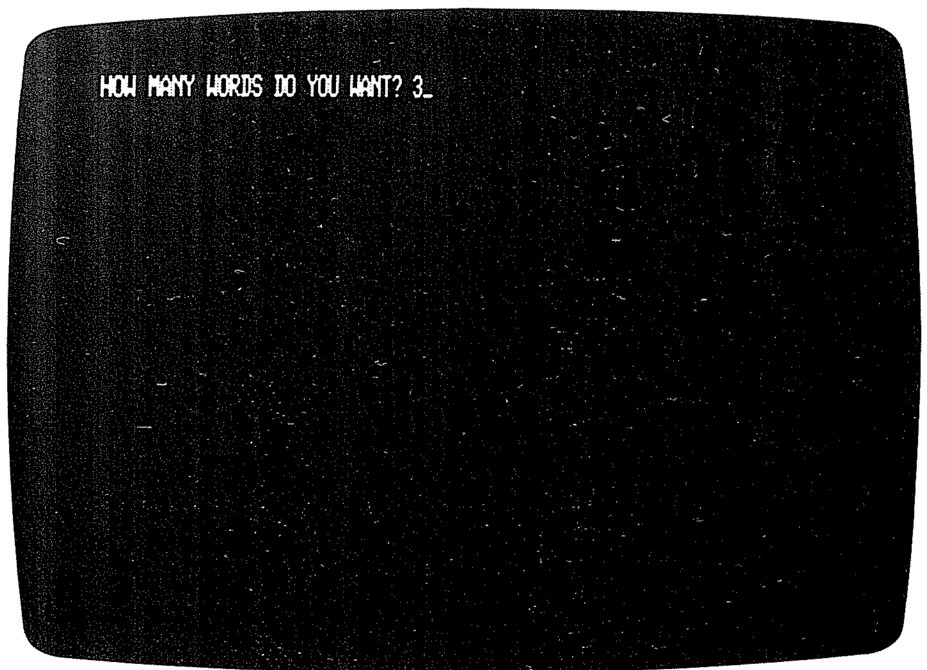
The words, clues, and scrambled words will all be stored in DATA statements. The program will start at a random word each time the program is run or the person asks to play again. Otherwise, the data will be read in order.

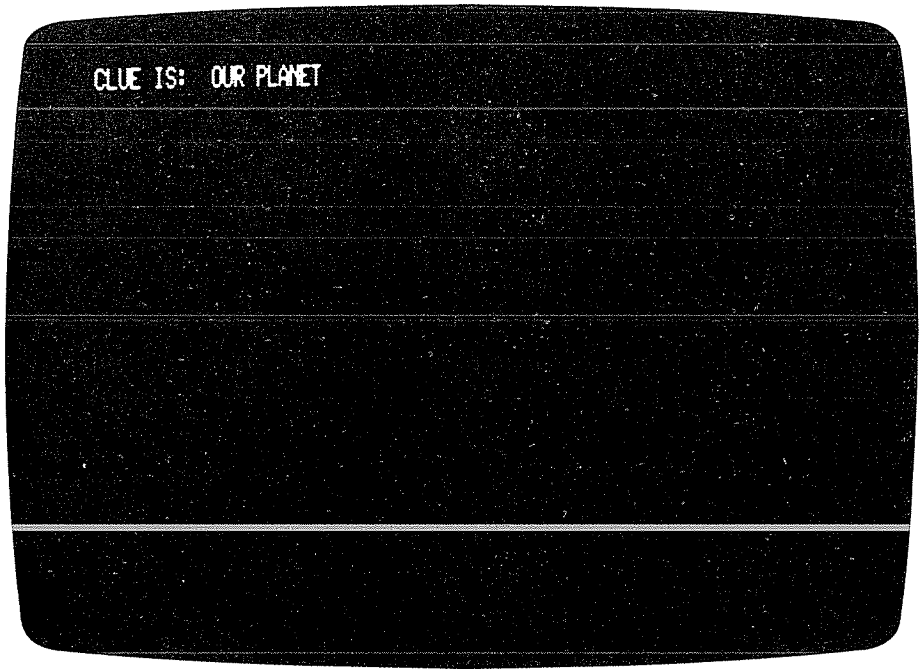
#### A Sample Run

Since the program uses the computer screen to present the scrambled word and make it disappear, we'll show the sample run in a series of photographs.

*This sequence of photographs shows what the interaction between the player and the game SCRAM1 looks like. Since CLS (clear screen) is used between plays, only the necessary information is displayed at any one time.*



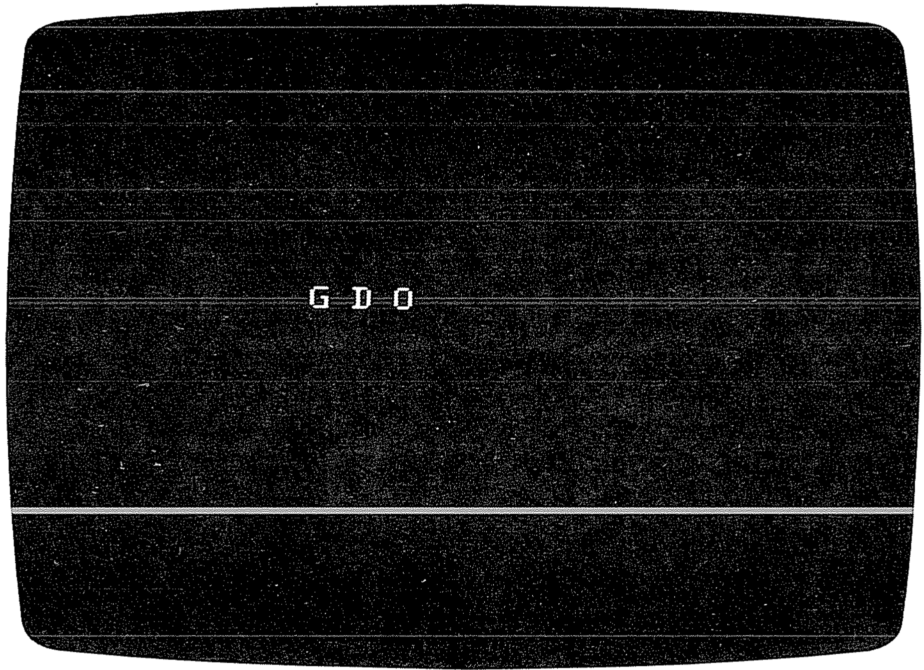



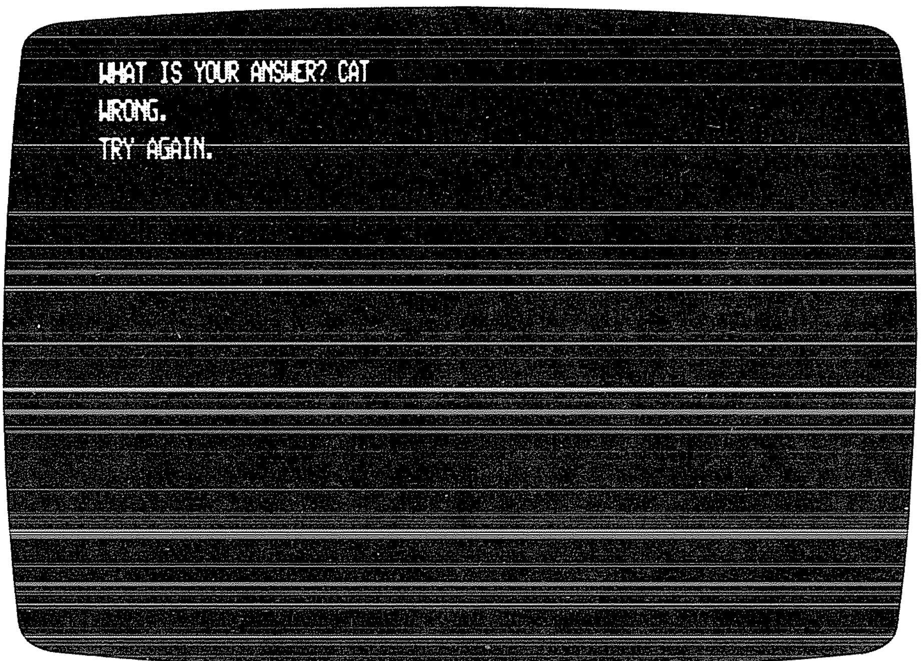

CLUE IS: OUR PLANET

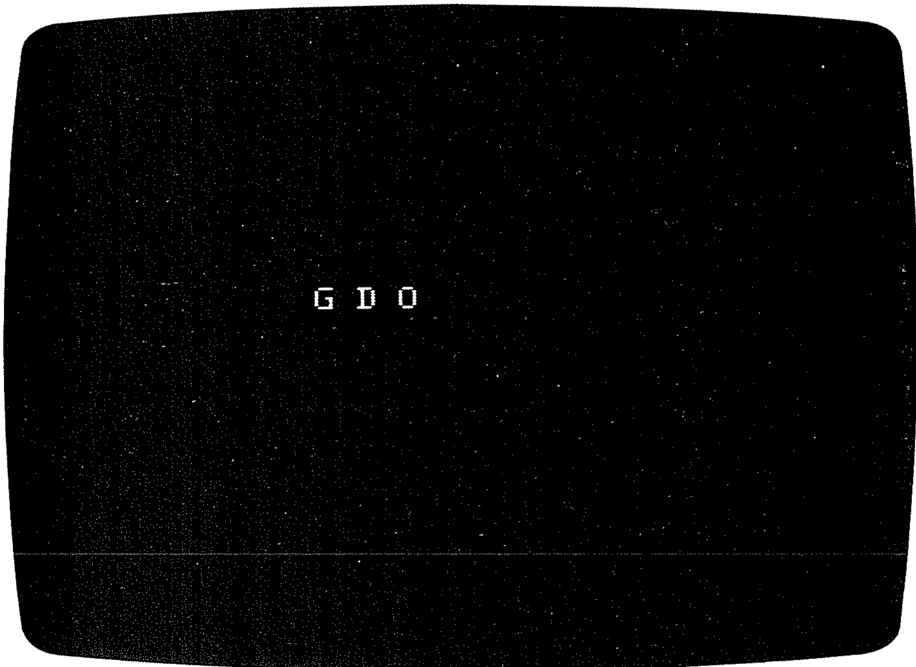
H R E T A

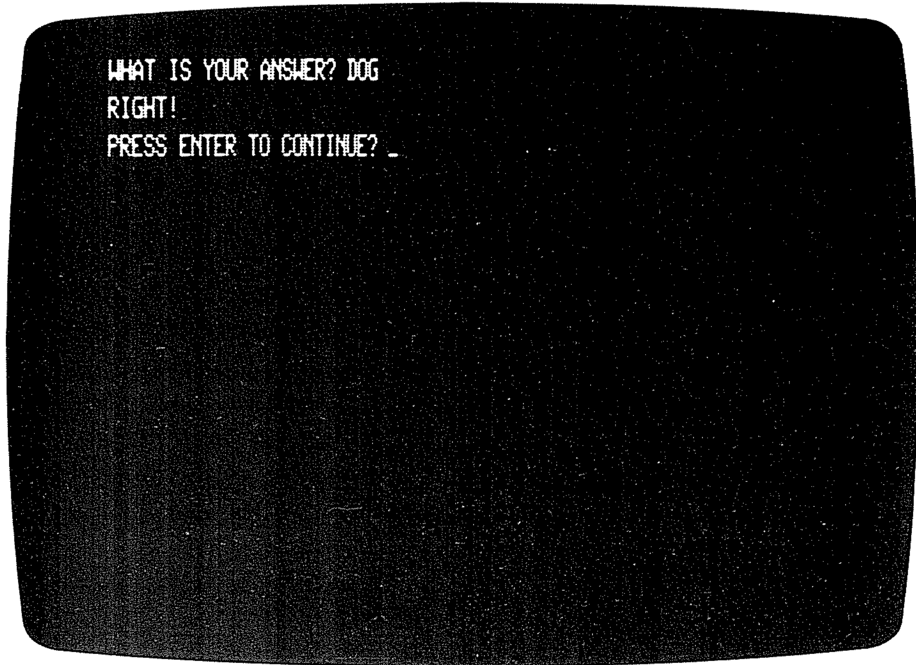
WHAT IS YOUR ANSWER? EARTH
RIGHT!
PRESS ENTER TO CONTINUE? _

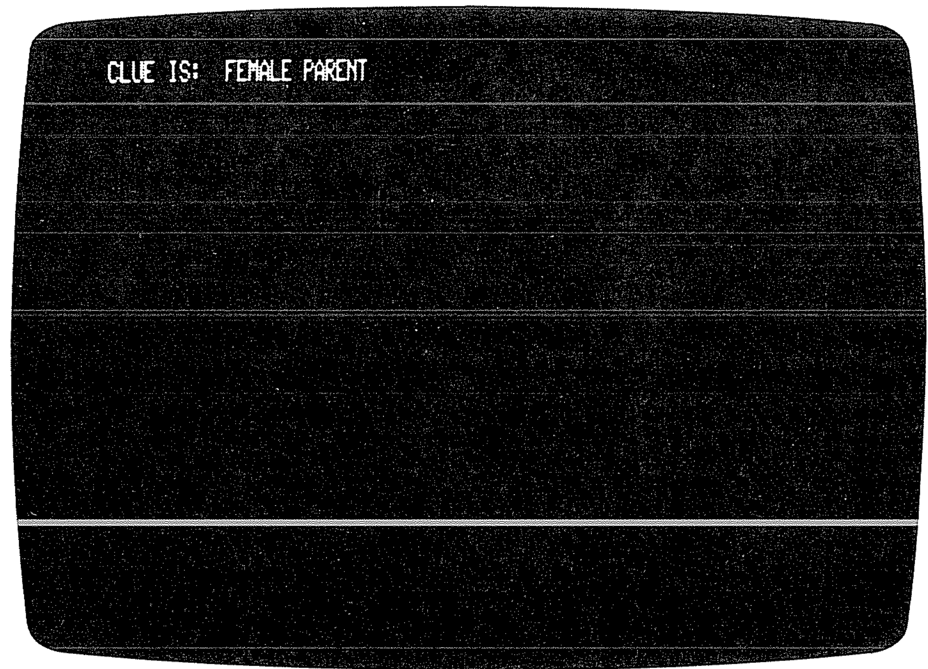
CLUE IS: A PET

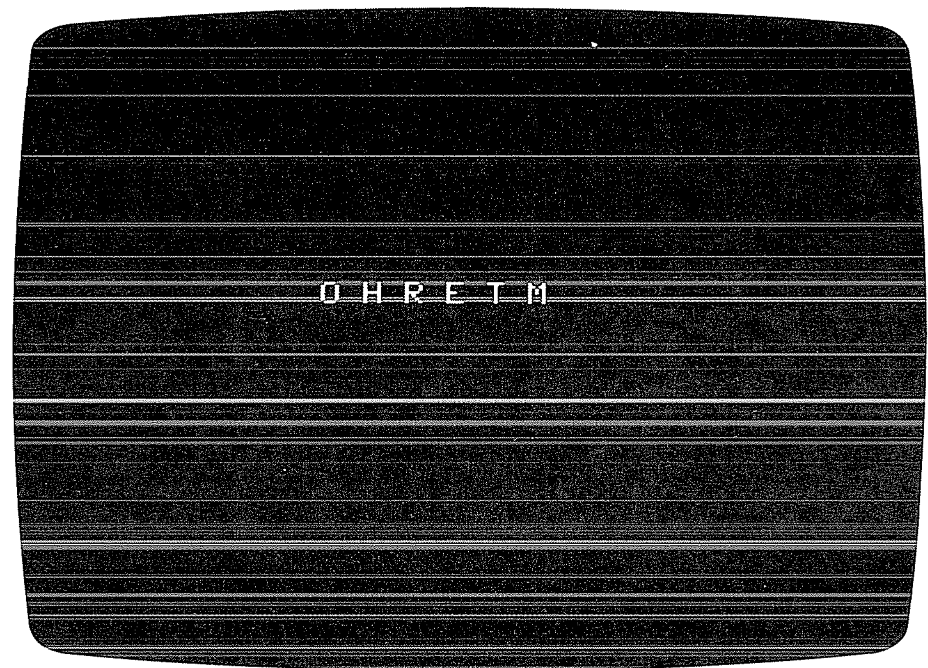

G D O


WHAT IS YOUR ANSWER? CAT
WRONG.
TRY AGAIN.


G D O

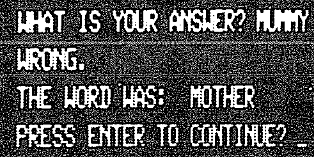

WHAT IS YOUR ANSWER? DOG
RIGHT!
PRESS ENTER TO CONTINUE? _

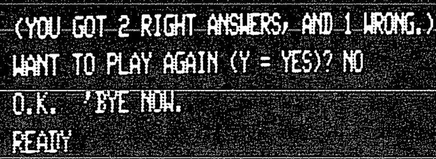

CLUE IS: FEMALE PARENT


O H R E T M

WHAT IS YOUR ANSWER? MOMMY
WRONG.
TRY AGAIN.

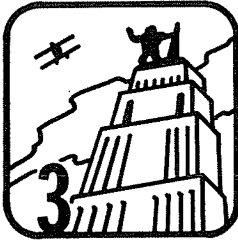
O H R E T M


WHAT IS YOUR ANSWER? MUMMY
WRONG.
THE WORD WAS: MOTHER
PRESS ENTER TO CONTINUE? _


(YOU GOT 2 RIGHT ANSWERS, AND 1 WRONG.)
WANT TO PLAY AGAIN (Y = YES)? NO
O.K. 'BYE NOW.
READY
>_

Notice how often the program stops and waits for the person to press ENTER, or some other response. It's important not to give the person too much to read at one time. How much? It depends on the complexity of what is printed, and it's partly a matter of taste. In general, it's better not to cram the screen full of words. This program avoids printing too much by using the *clear screen* (CLS) command often.

The scrambled word is printed in double-sized letters on the TRS-80. We'll see the details of how to do that later under **Coding the Program.**




## High-Level Design

1. Initialize variables, give instructions to the person. Variables are: D$, a dummy variable (its value is not used); N, the number of words wanted; W, number wrong; R, number right; F, a flag (see section on **Coding the Program**).
2. Generate a quiz word, print the scrambled word and clue.
   a. Start at a random word each turn.
   b. Don't forget to test if you're at the end of the data each time the word (W$), clue (C$), and scrambled word (S$) are read.
   c. Give the person time to read the scrambled word before it disappears. (Use an empty loop to wait a short time.)
3. Get the person's answer, A$; check if it's right or wrong.
   a. If it's right (A$ = W$), count it right, go back to 2.
   b. If it's wrong:
      (1) the first time, give the person another chance.
      (2) the second time, tell what the word was, count the answer wrong, go back to 2.
4. End of the game.
   a. Display the number right and wrong.
   b. Ask if the person wants to play again. If yes, go back to 1, but skip the instructions; if no, end.
5. Data: Words, clues, scrambles. There will be 20 sets of three items each.

Notice that high-level steps 2 and 3 taken together form the main game loop. If the game is to be played by two persons, they should alternate, and each ask for the same number of words.

## Coding the Program

The heart of this program is its data and the manner in which it is read and printed. Because the data remains the same, we want the program to start at a random word each game. To do this we use a "dummy" READ statement in a loop somewhat similar to Project *DATARIT4* in section 2.1. In SCRAM1, the loop counter X goes from 1 to some random integer from 1 to 20; that is, it goes from 1 to INT(20*RND(0)+1). When the expression

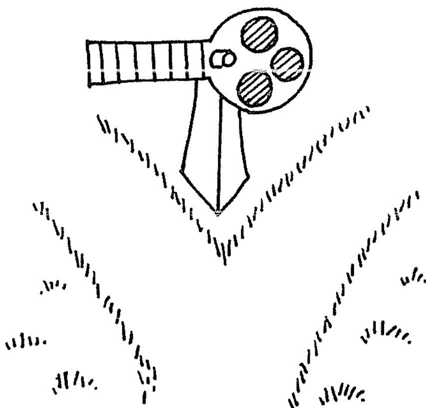
INT(20*RND(0)+1) = 1, the dummy READ in line 206 is executed once. So the "real" READ in line 220 gets its data from the second DATA statement. When INT(20*RND(0)+1) = 20, the dummy READ goes through 20 DATA statements, and the real READ uses the 21st DATA statement.

Will the first DATA statement ever be read? Yes. This is because of the way the main game loop from line 210 to 370 is written. Three data items are read in at once. This makes sense because they are related: the word, the clue for that word, and the scrambled version of that word. Immediately the program tests with an IF statement, to see if the data pointer is at the end of the data. If it is, RESTORE sets the pointer back to the first DATA statement. This "recycles" the data (as in Project *DATARIT3*, section 2.1), and so whenever the dummy loop reads 20 data triples, the first quiz word printed out will be from the first DATA statement.

Since we are going to make the clue and the scrambled word disappear from the screen, and the computer works fast, an empty loop is inserted at line 240 to make the computer slow down. You may decide that 300 times around the loop is either too much or too little. If so, change it. (If you are using a computer with printed output, these lines as well as all the clear screen commands can be deleted. In that case the method of printing out the scrambled word will also need to be altered.)

On the TRS-80, double-width letters are obtained by pressing shift and right arrow. The way to do this inside a program is to use the CHR$( ) function. Since 23 is the ASCII code for shift-right arrow (on the TRS-80), PRINT CHR$(23) will have the same effect as pressing those keys. Not all but many special control features of computers can be obtained using CHR$( ) (see section 2.2, and Appendix B). A clear screen command switches off the double-width letters, and goes back to normal sized ones. PRINT @ is another interesting feature of the TRS-80. Here we've used it simply to move the cursor six lines down and 16 spaces over before starting to print the scrambled word (PRINT @ will be explained more fully later, but notice that the value used, 400, is 6 * 64 + 16 = 400). Other computers may have a different way to let you start printing where you want. You'll have to be a little creative at this point if you're using a different machine.

In order to give the person a second chance before being counted wrong, the program must know whether a wrong answer has been given once or twice for a particular quiz word. For that purpose we make use of a *flag*. This is a general idea in computer programming that will come in handy many times. The flag is simply a variable whose value changes for different program segments. Depending on the meaning the programmer decides upon, these values then control how the program branches, something like signal flags on a highway.



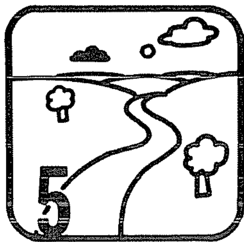
The flag variable, F, in SCRAM1 starts out as a 0. If the person gets a wrong answer the first time, F is changed to a 1 and the person gets another chance. The next time the person gets a wrong answer, F = 1 so the program branches differently. The program counts this answer as

wrong, tells what the answer was, and goes on to the next word.

But flags can be a little tricky: once F is changed to a 1, you must be careful to change it back to a 0 as soon as its signalling job is done. Since, on the second chance, the person might get either a right or a wrong answer, F is changed to 0 in both lines 320 and 330, even though sometimes F would be 0 anyway. In this program, there are other positions at which F could be turned on and off; the ones you see in the listing were chosen to be close to the most obvious place a change needs to be made.

```
10  '***********************************************
11  '*      SCRAM1  (WORD QUIZ)                    *
12  '***********************************************
100 '-----------------------------------------------
101 '      INITIALIZE, GIVE INSTRUCTIONS
102 '-----------------------------------------------
105 CLS
110 PRINT"    *** SCRAMBLED WORD GAME ***"
120 PRINT"TYPE THE WORD WHOSE LETTERS ARE SCRAMBLED"
122 PRINT"ON THE SCREEN.  NOTICE THE CLUE."
125 PRINT
128 PRINT"PRESS ENTER TO CONTINUE";: INPUT D$: CLS
130 PRINT"HOW MANY WORDS DO YOU WANT";: N=0: INPUT N
135   IF N < 1 THEN 130
140 CLS: RESTORE
150 W=0: R=0: F=0
200 '-----------------------------------------------
201 '      GENERATE QUIZ WORD
202 '-----------------------------------------------
205 FOR X = 1 TO INT(20*RND(0)+1)        'WILL START AT
206 READ D$,D$,D$                        'RANDOM WORD
207 NEXT X                               'EACH GAME
210 FOR I = 1 TO N
220   READ W$, C$, S$
225     IF W$ = "$" THEN RESTORE: GOTO 220      'RECYCLE DATA
230   PRINT"CLUE IS:   ";C$
240   FOR Z = 1 TO 300: NEXT Z                  'WAIT
245 '-----DISPLAY SCRAMBLED WORD-----------------------
250   CLS: PRINT CHR$(23)                       'BIG LETTERS
252   PRINT @ 400,;                      'MOVE CURSOR
255   FOR J = 1 TO LEN(S$)               'PRINT S$
256     PRINT MID$(S$,J,1);" ";          'ONE LETTER
257   NEXT J                             'AT A TIME
270   FOR Z = 1 TO 300: NEXT Z                  'WAIT
280   CLS
300 '-----------------------------------------------
301 '      GET ANSWER, CHECK IF RIGHT/WRONG
302 '-----------------------------------------------
310   PRINT"WHAT IS YOUR ANSWER";: A$=" ": INPUT A$
320   IF A$=W$ PRINT "RIGHT!": F=0: R=R+1: GOTO 360
325     PRINT"WRONG."
330     IF F=1 THEN F=0: W=W+1: GOTO 355
340       PRINT"TRY AGAIN."
350       F=1: GOTO 240
355     PRINT"THE WORD WAS:   ";W$
360   PRINT"PRESS ENTER TO CONTINUE";: INPUT D$: CLS
370 NEXT I
400 '-----------------------------------------------
```

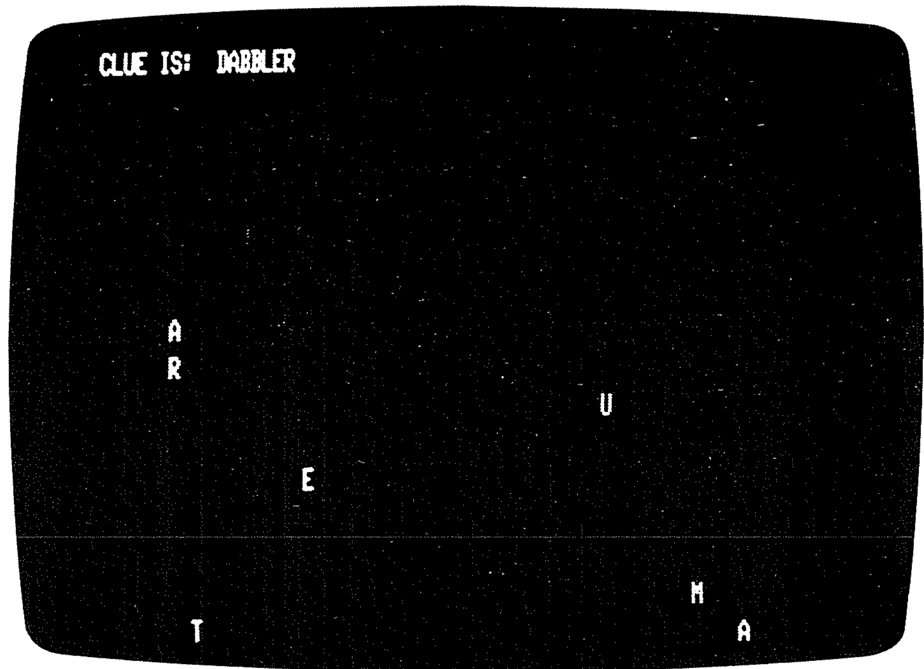
```
401 '     END OF GAME, DISPLAY SCORE
402 '-----------------------------------------------------
420 PRINT"(YOU GOT";R;"RIGHT ANSWERS, AND";W;"WRONG.)"
430 PRINT"WANT TO PLAY AGAIN (Y = YES)";: A$=" ": INPUT A$
440 IF A$="Y" THEN 130
450 PRINT"O.K.  'BYE NOW.": GOTO 999
500 '-----------------------------------------------------
501 '     DATA:  WORDS, CLUES, SCRAMBLES
502 '-----------------------------------------------------
510 DATA HOME, WHERE YOU LIVE, MHEO
515 DATA OLD, NOT NEW, LDO
520 DATA EARTH, OUR PLANET, HRETA
525 DATA DOG, A PET, GDO
530 DATA MOTHER, FEMALE PARENT, OHRETM
535 DATA QUICKLY, NOT SLOWLY, YKIQLCU
540 DATA CHILDREN, YOUNGSTERS, EDICNRLH
545 DATA INDIANS, COWBOYS & -------, SADININ
550 DATA FIRE, DON'T PLAY WITH ----, REIF
555 DATA BECAUSE, WHY? -------, EASEUCB
560 DATA REMEMBER, OPPOSITE OF FORGET, EEBREMMR
565 DATA SOMETIMES, NOT ALWAYS, EIEOSMTMS
570 DATA MOUSE, QUIET AS A -----, EUOMS
575 DATA DIFFERENT, NOT THE SAME, TEEFDNRFI
580 DATA WHALE, LIVES IN THE SEA, EAWLH
585 DATA SCIENTIST, DOES EXPERIMENTS, TINISSTEC
590 DATA TRAIN, RAILROAD, NATIR
595 DATA WAVES, THE SEA HAS -----, SVWEA
600 DATA NECESSARY, NEEDED, YASCNRSEE
605 DATA INSECT,BUG,TENCSI
610 DATA $,$,$
999 END
```



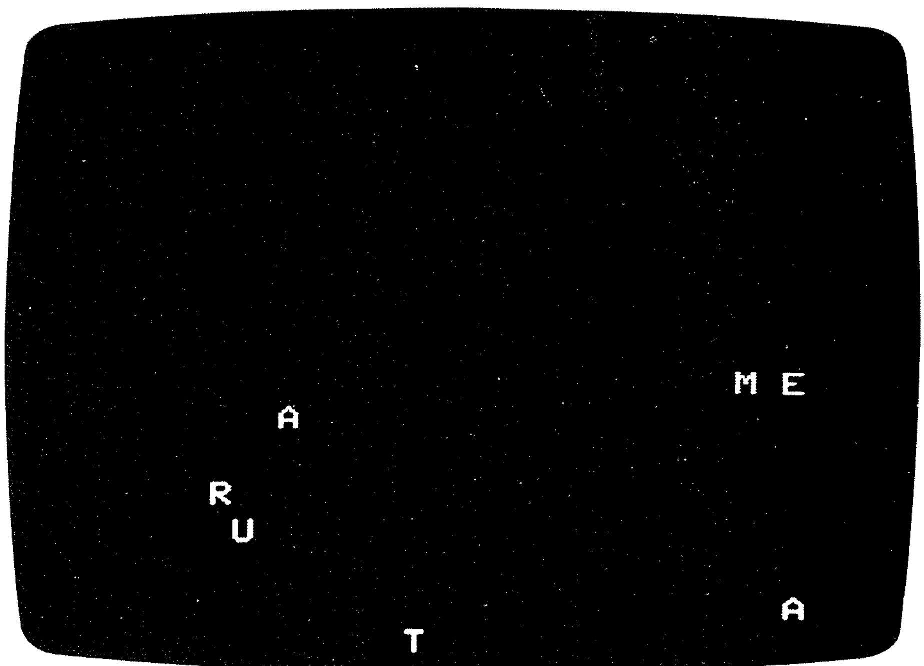

## Future Extensions and Revisions

After playing SCRAM1 a while, you begin to memorize the data. This can be an asset if you're interested in memorizing that data, but it can be a liability if you're looking for a surprise each time. The simplest way to change a program like this is to type in a new batch of data from time to time.

The program can also be changed so that the scrambled word is presented differently. The letters can be scattered all over the screen at random positions just to make guessing a bit more interesting. It would also be fun if the first presentation of the word was in regular-sized letters, and the second chance used double-width letters for emphasis. Another extension would be to choose words randomly from the available data.




*These two photographs show the clue given in SCRAM2. The first scrambled word is given with normally-sized letters, and erased from the screen after about a second. If the player doesn't get the word, the word is repeated with double-sized letters as shown in the second photograph, and again erased after a brief display. The timing for the display could be varied for different levels of players by changing the "do nothing" wait loop (see line 270 in SCRAM1) to FOR Z = 1 TO V : NEXT V, where V is varied from about 900 for beginners to about 50 for speed reading experts.*



## Project *SCRAM2*

Write a program that has the above improvements. These changes are summarized in the following high-level design:

1. Initialize and give instructions. (Same as SCRAM1, but dimension an array P(15) for the random positions.)
2. Generate a quiz word.
   a. Choose a random word each time a word is presented. This means the randomizing loop with the dummy READ should be inside the main loop. Precede it with a RESTORE.
   b. Read the data and print the clue. (Similar to SCRAM1).
   c. The loop for printing the scrambled word will contain several processes:
      (1) Choose a random position on the screen (a random number to be used by a PRINT @ statement).
      (2) Make sure this position has not already been used to print a letter.
      (3) Print the letter.
3. Check the answer. (Same as SCRAM1.)
4. End of the game, print the score. (Same as SCRAM1.)
5. The data. (Same kind as SCRAM1 but you may as well use different words.)

## Some Hints on Coding the Program

Since there is a RESTORE statement within the loop, there is no need to test if the data pointer is at the end of the data each time the real READ statement is executed. But since no recycling of the data is done, the first DATA statement will never be read. To remind yourself of this fact (and not waste a good word) make the first DATA statement look like this:

```
510 DATA X, X, X
```

Follow it with 20 more DATA statements (each containing a word, clue, and scrambled word). The dummy READ loop will randomly read from 1 to 20 DATA statements before the real READ gets a chance. There will be 21 DATA statements altogether.

## More About PRINT @

In this and the next project we will create some random numbers which will be used by the PRINT @ statement, so we need to visualize clearly how this works on the screen. The TRS-80 (Model I) has a screen which displays 16 lines of 64 characters each, a total of 1024 characters. PRINT @ refers to these character-spaces by numbering them consecutively from 0 (upper left corner) to 1023 (lower right corner) as follows:

| 0 | 1 | 2 | 3 | | 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|
| 64 | 65 | 66 | 67 | | 124 | 125 | 126 | 127 |
| 128 | 129 | 130 | 131 | | 188 | 189 | 190 | 191 |
| 192 | 193 | 194 | 195 | | 252 | 253 | 254 | 255 |
| 256 | 257 | 258 | 259 | | 316 | 317 | 318 | 319 |
| 320 | 321 | 322 | 323 | | 380 | 381 | 382 | 383 |
| 384 | 385 | 386 | 387 | | 444 | 445 | 446 | 447 |
| 448 | 449 | 450 | 451 | | 508 | 509 | 510 | 511 |
| 512 | 513 | 514 | 515 | | 572 | 573 | 574 | 575 |
| 576 | 577 | 578 | 579 | | 636 | 637 | 638 | 639 |
| 640 | 641 | 642 | 643 | | 700 | 701 | 702 | 703 |
| 704 | 705 | 706 | 707 | | 764 | 765 | 766 | 767 |
| 768 | 769 | 770 | 771 | | 828 | 829 | 830 | 831 |
| 832 | 833 | 834 | 835 | | 892 | 893 | 894 | 895 |
| 896 | 897 | 898 | 899 | | 956 | 957 | 958 | 959 |
| 960 | 961 | 962 | 963 | | 1020 | 1021 | 1022 | 1023 |

It's best to think of this mass of numbered spaces in line length batches; that is, in groups of 64. For *SCRAM2* we don't want to use the whole screen for scattering our scrambled word, perhaps just 10 lines out of the 16. We know by now that

```
INT(RND(0) * (64 * 10))
```

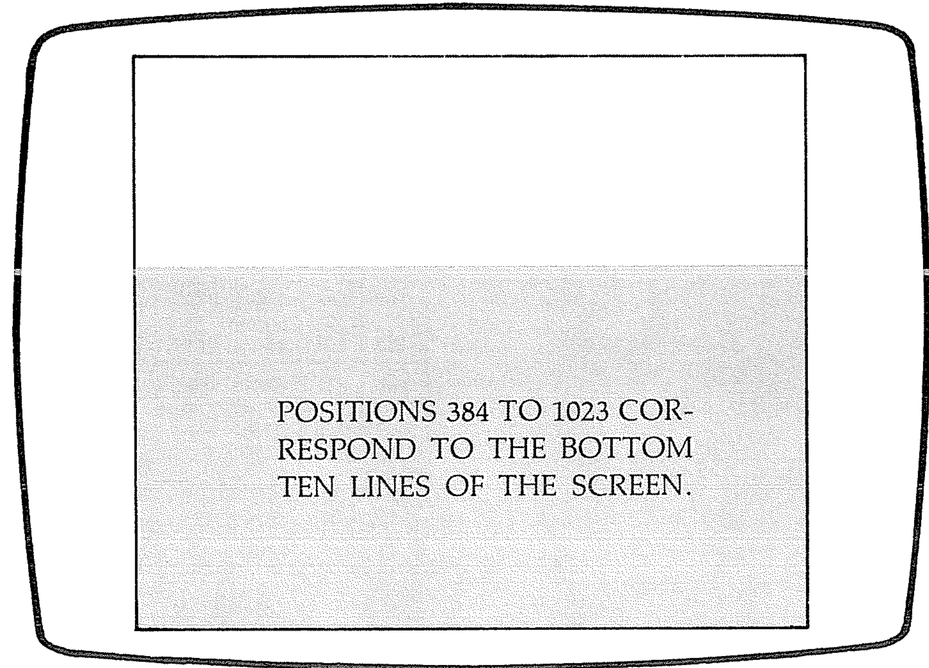
will give us random integers from 0 to 639. (We've left the 640 as the expression "64 * 10" so you can keep track of how many lines-full of character spaces there are.)

POSTIONS 0 TO 639 CORRES-
POND TO THE TOP TEN
LINES OF THE SCREEN.

But we want to reserve six lines at the top of the screen for presenting the clue, asking for the answer, responding, telling the person "right", "wrong", etc. So we'll use

```
INT(RND(0) * (64 * 10) + (64 * 6))
```

to give us random integers from 384 to 1023. These correspond to PRINT @ positions on lines 7 through 16 of the screen:




Now we have the right range of numbers, but in order to print double-width letters on the TRS-80, we must generate only random *even* integers. Even integers are those that have two as a factor, so all we have to do is figure out what expression will generate random integers in a range exactly half what we want, and then multiply it by 2. Better still, let the computer do the work.

```
INT((RND(0) * (64 * 10) + (64 * 6))/2) * 2
```

For example, 385/2 = 192.5, the "integer part of" 192.5 = 192, and 192 *2 = 384.

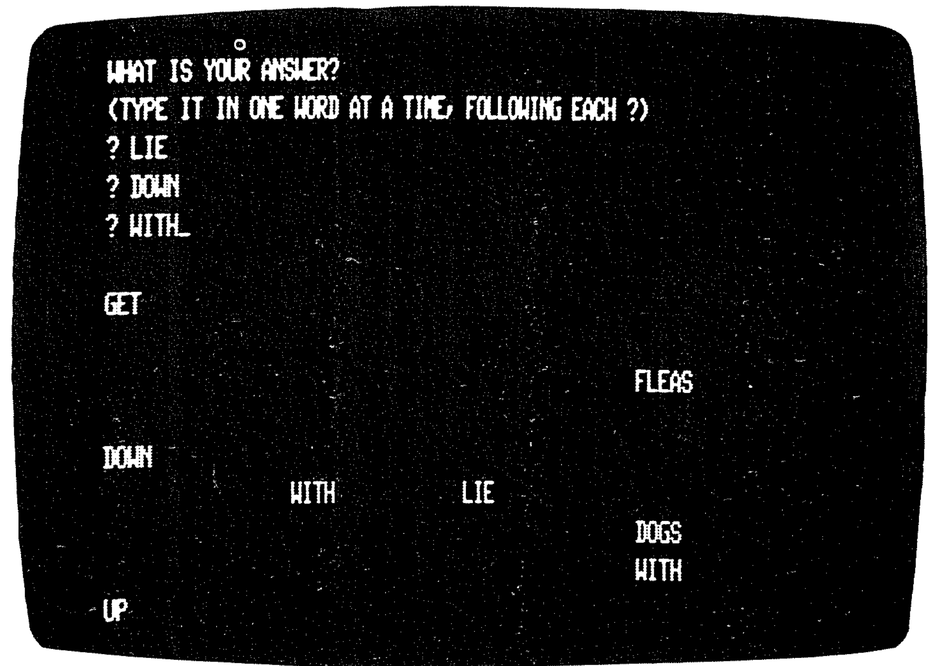
Previously we suggested the values for PRINT @ positions be stored in an array. This is because the next thing that must be done in the program is to check if this position has been used before to print a letter of the current quiz word. If this is not done the program might randomly choose the same position more than once and overprint, which would make it too difficult to guess the word. A high-level design for this part of the program is as follows:

1. Choose the random position, store it in an array.
2. If this is the first one, skip the next step.
3. Compare this random position with each previous one, and if it's the same position, go back to step 1.
4. If it's not the same, go ahead and print the letter in that position.

*Note:* Successive PRINT @ statements require a semicolon at the end of the statements, otherwise each one will do a line feed and carriage return, erasing the line on the screen below the one which was just used and rolling up the contents of the screen if the position was on the last line. The semicolon prevents this.

## Project *SCRAM3*

Now that you have a couple of word guessing games, you might as well go on to bigger and better things,—say a sentence guessing game. Write a program that takes the words of a sentence or a proverb, scatters them over the screen in random positions, and in a new random order each time. Ask the person to type in the words in the correct order. No clue is needed, but you'd better not make the scrambled words disappear. Start at a random proverb each time the program runs or if the person wants to play again.




*This is a sample run of SCRAM3. The scrambled proverb is first printed in the lower part of the screen. The player then tries to type it in the correct order after each ? mark starting near the top of the screen. As new words are entered, parts of the scrambled proverb may be erased, so the player should try to figure out the whole proverb before typing in any words.*

First think about how your data will be written. This will affect how your dummy READs and real READs work. Since this program will deal with words separately, the data should probably look like this:

510 DATA A, BAD, PENNY, ALWAYS, COMES, BACK, $

Some proverbs take up more than one DATA statement, so the dollar sign ($) allows a test for checking the end of one proverb. The last DATA statement is:

630 DATA $$$

This triple dollar sign is an arbitrary string we've selected to allow testing for the end of all the proverbs.

No proverb has more than 15 words, so the arrays for storing random positions P( ), scrambled word order S( ), the words of the proverb W$( ), and the person's answer A$( ) are all dimensioned at 15.

A high-level design for this program will be similar to SCRAM1 and *SCRAM2*. A more detailed high-level design for part 2 of *SCRAM3* is as follows:

2. Generate the quiz proverb.
   a. Choose a random proverb for each game. This randomizing process should be outside the main game loop, as in SCRAM1. It will have two parts: (1) a dummy READ loop that moves a pointer through the data one word at a time, and (2) a loop that then reads up to the end of whatever proverb this data pointer randomly landed on. This way the real READ statement will start at the first word of a proverb. In each loop, care should be taken to test for the end of the data.
   b. Read the words of a proverb into an array W$( ). Remember the number of words read (ie:, don't reuse the counter variable, preserve it).
   c. Create a random ordering for the words. To do this fill an array S( ) (the same length as W$( )) with the numbers from 1 to the number of words in W$( ), but in scrambled (random) order. Start with a random number R, in the proper range, and place a 1 in S(R). Repeat for 2, 3, etc. You can test if S(R) has already been filled if you put zeroes in S( ) before you start. If it has been filled, just fill the next empty variable in the array (R = R + 1). If the loop gets to the end of the array while looking for an empty spot to place a number, just reset R to 1 and keep looking.
   d. Display the words scattered randomly. As in *SCRAM2*, fill an array with random positions for use by a PRINT @ statement, checking to see that none has been used before. This time, we suggest using only every 16th number in the range, using the same trick as was used to get even numbers. The PRINT statement will look like this:

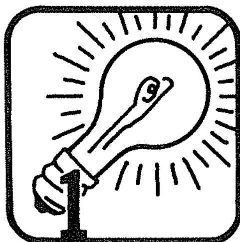
      280 PRINT @ P(K), W$(S(K));

   P(K) is a randomly chosen position, while W$(S(K)) is a ran-

domly chosen word from the proverb. Notice it uses a subscripted subscript!

## Project *CIPHER*

Write a secret code game. This project will use the technique learned in the previous word quiz programs, but its structure is somewhat different. The first three steps of the design process are shown in full.



### Idea

To make secret codes into a game, this program will present a coded message, then ask the person to type in a letter, and the letter he or she thinks it stands for. The program will attempt to decode the message using the replacement scheme denoted by those two letters (see **Note on Ciphers and Codes**). The program will print the coded message, and the message will still be gibberish if the letters were wrong. If they are right, that's the end of the game, a score is printed, and the person can play again. If they are wrong, the person has three choices:

1. Try to decode the message again (type in two more letters).
2. Try a new message in the same code.
3. Give up.

In order to play this game well, the person should understand what kind of code or cipher is being used, what kind of messages will be sent, and have some ideas about looking for common English words in the coded messages. Since this will make the instructions quite long, the person should have the choice of skipping them.

### Sample Run

```
   *** DECODE THE SECRET MESSAGE ***
DO YOU NEED INSTRUCTIONS (YES/NO)? YES

   HERE IS A CODED MESSAGE:  Q HUT QFFBU
        HERE IT IS DECODED:  A RED APPLE

PRESS ENTER TO SEE WHAT THE CODE WAS?
*     * *             *       *   *
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
Q R S T U V W X Y Z A B C D E F G H I J K L M N O P

  IN THIS GAME, NEW CODES ARE MADE BY 'SHIFTING'
  THE ENCODING ALPHABET TO A NEW POSITION
```

```
PRESS ENTER TO SEE A NEW CODE?

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
E F G H I J K L M N O P Q R S T U V W X Y Z A B C D

PRESS ENTER TO CONTINUE?

   TO PLAY THIS GAME YOU MUST--
      1.  LOOK AT THE CODED MESSAGE ON THE SCREEN.
      2.  PICK ONE LETTER AND FIGURE OUT WHAT
             LETTER WAS ENCODED TO GET IT.

PRESS ENTER TO CONTINUE?

   HERE ARE SOME CLUES:
      --ALL THE MESSAGES ARE OLD FAMILIAR SAYINGS.
      --SOME ONE-LETTER WORDS IN THEM ARE: I, A.
      --SOME TWO-LETTER WORDS IN THEM ARE:
            TO, AT, IS, IN, IT, OF.
      --SOME THREE-LETTER WORDS IN THEM ARE:
            AND, ALL, BUT, NOT, THE.

PRESS ENTER TO CONTINUE?

   YOUR SCORE WILL DEPEND ON HOW MANY MESSAGES
   YOU NEED TO SEE BEFORE DISCOVERING THE CODE
   (1 = 100%  2 = 50%  3 = 33%  4+ = 0).

PRESS ENTER TO START THE GAME?

HERE IS THE CODED MESSAGE:

OIIT CSYV ICIW STIR FIJSVI QEVVMEKI, LEPJ WLYX EJXIVAEVHW

   WHAT LETTER DO YOU THINK YOU CAN DECODE? I
   WHAT LETTER DO YOU THINK IT STANDS FOR? E

THE DECODED MESSAGE IS:

KEEP YOUR EYES OPEN BEFORE MARRIAGE, HALF SHUT AFTERWARDS

CONGRATULATIONS.
   YOU HAVE BROKEN THE CODE AFTER RECEIVING
    1 MESSAGE(S).
   YOUR SCORE IS 100 %.
WANT TO SEE ANOTHER MESSAGE--WITH A NEW CODE? YES

HERE IS THE CODED MESSAGE:

RHN VTG EXTW T AHKLX MH PTMXK, UNM RHN VTG'M FTDX BM WKBGD

   WHAT LETTER DO YOU THINK YOU CAN DECODE? T
   WHAT LETTER DO YOU THINK IT STANDS FOR? I

THE DECODED MESSAGE IS:

GWC KIV TMIL I PWZAM BW EIBMZ, JCB GWC KIV'B UISM QB LZQVS
```

```
SORRY, THAT'S NOT IT.
WOULD YOU LIKE TO ...
  1. TRY DECODING THE MESSAGE AGAIN.
  2. TRY DECODING A DIFFERENT MESSAGE (SAME CODE).
  3. GIVE UP ON THIS CODE.

TYPE 1, 2, OR 3? 1

HERE IS THE CODED MESSAGE:

RHN VTG EXTW T AHKLX MH PTMXK, UNM RHN VTG'M FTDX BM WKBGD

   WHAT LETTER DO YOU THINK YOU CAN DECODE? T
   WHAT LETTER DO YOU THINK IT STANDS FOR? A

THE DECODED MESSAGE IS:

YOU CAN LEAD A HORSE TO WATER, BUT YOU CAN'T MAKE IT DRINK

CONGRATULATIONS.
   YOU HAVE BROKEN THE CODE AFTER RECEIVING
    1 MESSAGE(S).
   YOUR SCORE IS 100 %.
WANT TO SEE ANOTHER MESSAGE--WITH A NEW CODE? YES

HERE IS THE CODED MESSAGE:

DEJXYDW YI SUHJQYD YD JXYI MEHBT RKJ TUQJX QDT JQNUI

   WHAT LETTER DO YOU THINK YOU CAN DECODE? Y
   WHAT LETTER DO YOU THINK IT STANDS FOR? S

THE DECODED MESSAGE IS:

XYDRSXQ SC MOBDKSX SX DRSC GYBVN LED NOKDR KXN DKHOC

SORRY, THAT'S NOT IT.
WOULD YOU LIKE TO ...
  1. TRY DECODING THE MESSAGE AGAIN.
  2. TRY DECODING A DIFFERENT MESSAGE (SAME CODE).
  3. GIVE UP ON THIS CODE.

TYPE 1, 2, OR 3? 2

HERE IS THE CODED MESSAGE:

EX, MXQJ Q JQDWBUT MUR MU MUQLU MXUD VYHIJ MU FHQSJYSU JE TUSUYL
U

   WHAT LETTER DO YOU THINK YOU CAN DECODE? E
   WHAT LETTER DO YOU THINK IT STANDS FOR? O

CONGRATULATIONS.
   YOU HAVE BROKEN THE CODE AFTER RECEIVING
    2 MESSAGE(S).
   YOUR SCORE IS 50 %.
WANT TO SEE THE MESSAGES AND CODED MESSAGES
YOU RECEIVED IN THIS CODE (Y/N)? Y
NOTHING IS CERTAIN IN THIS WORLD BUT DEATH AND TAXES
DEJXYDW YI SUHJQYD YD JXYI MEHBT RKJ TUQJX QDT JQNUI
```

```
OH, WHAT A TANGLED WEB WE WEAVE WHEN FIRST WE PRACTICE TO DECEIV
E
EX, MXQJ Q JQDWBUT MUR MU MUQLU MXUD VYHIJ MU FHQSJYSU JE TUSUYL
U


WANT TO SEE ANOTHER MESSAGE--WITH A NEW CODE? YES

HERE IS THE CODED MESSAGE:

CSY GER PIEH E LSVWI XS AEXIV, FYX CSY GER'X QEOI MX HVMRO

   WHAT LETTER DO YOU THINK YOU CAN DECODE? X
   WHAT LETTER DO YOU THINK IT STANDS FOR? S

THE DECODED MESSAGE IS:

XNT BZM KDZC Z GNQRD SN VZSDQ, ATS XNT BZM'S LZJD HS CQHMJ

SORRY, THAT'S NOT IT.
WOULD YOU LIKE TO ...
  1. TRY DECODING THE MESSAGE AGAIN.
  2. TRY DECODING A DIFFERENT MESSAGE (SAME CODE).
  3. GIVE UP ON THIS CODE.

TYPE 1, 2, OR 3? 3
   YOUR SCORE IS 0%.
WANT TO SEE ANOTHER MESSAGE--WITH A NEW CODE?NO
   *** END OF SECRET CODE GAME ***
```

### High-Level Design

1. Initialize, give instructions.
   a. Ask if person wants instructions; if not go to 1c.
   b. Explain about codes, the order of playing, clues, scoring.
   c. "Press ENTER to start game."
2. Read and encode the message.
   a. Pick a random proverb each game.
   b. Pick a random number R for the code (from 1 to 25).
   c. Read the proverb into one string M$( ) using concatenation (see section 2.2).
   d. Create a coded mesage CM$( ) by adding R to the ASCII value of each letter in M$( ). (Punctuation marks and spaces should be the same in CM$( ) as they were in M$( ).
3. Display the coded message; get the input; decode the message.
   a. Print the coded message, get the person's input, X$, Y$.
   b. Subtract the ASCII value of one input letter from the other to get a decoding value, Q.
   c. Reverse the coding process, creating a decoded message, DM$.
   d. Print the decoded message, compare it with M$( ).

e. If it is correct, go to 4; if not, give the person three choices:
   (1) Try to decode the message again; go to 3a.
   (2) Try a new message in the same code; go to 2c.
   (3) Give up; go to 4b.

4. End of game; display score; print out messages and coded messages.
   a. For correct decoding: calculate score, print message.
   b. Recalculate score if the person gave up; print score.
   c. If there was more than one message looked at, ask if person wants to see them. If not, go to 4e.
   d. Print messages and coded messages.
   e. Ask if person wants to play again. If yes, go to 2.
5. Data: old, familiar proverbs again.

## Hints on Coding the Program

Since this program deals with the message as a whole, rather than as separate words, the DATA statements should contain parts of the proverbs that are as large as possible. Since these parts will contain spaces and punctuation marks, they will be surrounded by quotation marks. For example:

```
505 DATA "IT'S BETTER TO HAVE LOVED AND LOST THAN NEVER TO"
510 DATA "HAVE LOVED AT ALL", $
        .
        .
        .
610 DATA $$$
```

In this case $ allows a test for the end of one proverb, while $$$ allows a test for the end of all the data.

To find out if a character within the message is not a letter, test if its ASCII value is greater than 90 or less than 65.

If adding the coding or decoding value makes a number greater than 90 or less than 65, a non-letter character will result; be sure to test for this and avoid it by adding or subtracting 26. Here is a simple example of adding a coding value called SH (for shift):

```
134 PRINT:PRINT "PRESS ENTER TO SEE A NEW CODE";: INPUT D$
138 PRINT: FOR I = 65 TO 90: PRINT CHR$(I);" ";: NEXT I
140 PRINT: FOR I = 1 to 26: PRINT CHR$(92);" ";: NEXT I; PRINT
142 SH = 41
144 FOR I = 65 to 90
148 IF I + SH > 90 THEN SH = SH - 26
149 PRINT CHR$(I+SH);"";
150 NEXT I: PRINT
```

This code will print an alphabet (line 138) and a shifted alphabet (line 149).

This program takes up a lot of string space. We found that CLEAR 900 was necessary.

The messages and coded messages, M$( ) and CM$( ) are stored in arrays so that if the person asked for more than one message in the same game, he or she can ask to have them all printed out at the end of the game.

When creating M$( ), CM$( ), or DM$ by repeatedly concatenating characters, you must start with a null or empty string. To make sure of this you need statements like:

```
220 M$(C) ="": CM$(C) = ""
314 DM$ = ""
```

**Notes on Ciphers and Codes**

The words code, cipher, and cryptogram all refer to schemes for disguising information in some manner. However, the three words have different technical meanings (not always observed) as follows:

A *cryptogram* is defined as "something written in code or cipher," so it's a general word that includes the other two. A *code* is defined as a system of symbols, letters, or words used to transmit arbitrarily defined meanings. When a college administrator says "we give equal emphasis to research and teaching," it could be a code that means "publish or perish." If a baseball coach signals a bunt by scratching his nose, that's also a code. There's no way to *decipher* a code by a mathematical formula; you need a dictionary-type listing called a codebook. The file names of programs in this book are a kind of code. You can guess what *BABYZAP* might mean, but to be sure you have to look it up.

A *cipher*, on the other hand, uses some kind of mathematical transformation for methodically changing the parts of a message. Two types of transformation used are *transposition* and *substitution*. A transposition scheme is one that changes the position of the symbols in a message, turning a word like SECRET into, say, CREEST. The various SCRAM programs in this section use transposition.

The second and more useful transformation is substitution. This involves replacing each symbol in the message with another symbol taken from the basic set of characters. The most famous substitution scheme is the Caesar cipher (reportedly used by Julius Caesar) in which each letter of the alphabet is replaced by a letter a fixed distance, R, away. The distance from A to B is 1, A to C is 2, A to D is 3, and so on. The distance going backwards in the alphabet is taken as a negative number. So for R = −1, IBM becomes HAL. The project *CIPHER* uses a Caesar substitution cipher, choosing the distance R randomly for each game.

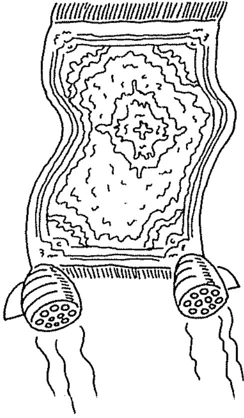
The Caesar cipher is easy to "crack" (decipher) since all the letters have the same R added to them. This means letters that appear frequently in English, such as E, also appear frequently in the cipher text, and it's easy to guess what they are. A better scheme is one that uses several distances in succession (say 7, 3, 5) for successive letters, repeating this *key* pattern as often as needed. But unless the pattern is very long, experts can also crack this kind of cipher. The only theoretically uncrackable cipher is one where the key is as long as the message, and where the key is never used more than

once. This scheme gives rise to the *one-time-pad* cipher, in which a fresh, and very long, key is printed on each page of a book (the pad). A page is used once by both sender and receiver, who have duplicate copies, and then destroyed. Such pads, printed on very thin paper with very tiny numbers have reportedly been rolled up to look like cigarettes.

The one-time pad scheme is unbreakable, but delivering all those pads around the world without interception becomes a monumental problem. A way to avoid this problem has recently been developed, using what is called a *public key* encryption scheme. This means that keys for *enciphering* messages can be made public, with a personal key for each receiver. But only the receiver will know how to decipher a message written with his key. This is because the publicly known enciphering scheme is not easily reversed. (Most substitution ciphers are easy to reverse. For example you can reverse the cipher which changes IBM to HAL by using +1 instead of −1.)

The enciphering schemes used in public key systems are called *one-way trap door* functions. They're called one-way because the enciphering process is almost impossible to reverse. It has been estimated that even if an extremely fast computer were used, it would take millions of years to find a reverse key. They're called trap door functions because if you know where the "secret lever" is, which means if you know a secret number, the reverse process is relatively easy, taking only a few seconds on a computer. A long and somewhat technical article on public key encryption systems appeared in the August 1979 issue of *Scientific American*. Two other articles on encryption can be found in the 1979 March and April issues of BYTE magazine. A fascinating book on the history of ciphers and codes is *The Code Breakers* by David Kahn.

## 2.5 Space Games. BABYQ, *BABYZAP*

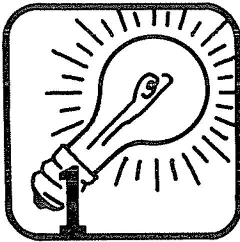


Let's now leave the land of computer number and word games, and climb aboard the modern magic carpet of fantasy space games. These come in all sizes and shapes, and untold numbers of them have been written for microcomputers. Some are simple, some very complex; a few have even attracted a national following. But like the photos in the wallet of a proud parent, computer games mean a lot more to their inventors than to the casual observer.

Our goal in this section is to help you take part in inventing a "space trek" game. It may be simpler than other space games you have seen, but because you will understand how it was designed and put together, it, and the extensions you devise, may end up being among your favorites.

### Program BABYQ

### The Idea



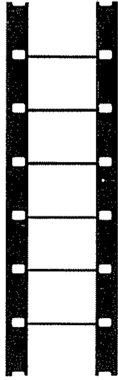

This will be a game program with two adversaries: the human player (as captain of a ship called E), and the computer (in the guise of an evil force called K). These two adversaries will take turns making moves within the confines of a miniature galaxy, displayed on the video screen as a 9 by 9 grid. There will be well-defined rules for determining the outcome of each move. The human player will have the goal of reaching a position called the "star gate," but of course a lot can happen along the way.

To clarify how all of this goes together, let's take a look at the rules in more detail and illustrate how they work with a sample run.

### Sample Run

The run is shown as a sequence of time frames (indicated by printing TF = 1, TF = 2, TF = 3, and so on at the right of the screen). The rules that govern what happens for each time frame are the following:

1. The game is played on a 9 by 9 grid. A position is described with integer coordinates X and Y.
2. The player of the game is the commander of the E (Entelechy Quest) ship. The E ship starts out at position 1,1 with 100 units of energy (EE = 100) and in time frame 1 (TF = 1).
3. The E ship can be moved to any X,Y position on the grid, provided the distance travelled is not greater than EE/100. Distance travelled is calculated by the Pythagorean theorem. Example: To go from 2,3 to 5,7 means going along a triangle with sides of length 3 (5-2) and 4 (7-3), so distance equals SQR(9 + 16) = 5. This move is not allowed if EE is less than 500.



4. The E ship can increase its energy 10% per time frame by resting, that is, by staying in its present position.
5. After each E ship move, a K-mine is planted at a randomly chosen position of the grid. If this position is the same as the new E position, the E ship is blown back to start, 1,1, and its energy is reduced by 30%.
6. If the E ship moves into a position where there is an old K-mine planted, the results are the same as in 5.
7. The E ship's goal is to go through the star gate at position 9,9. When the letter G appears in this position the gate is closed, and trying to go through will bounce E back to start. When it is open it will appear blank, and going through will give the player the winning message "YOU HAVE ACHIEVED STAR DIMENSION," along with a rating score. This rating = 10 * EE/(TF + 1).
8. The star gate is protected by a star barrier shown with the symbol "*". Moving to one of these positions will bounce the E ship back to start.
9. The game is ended before Star Dimension is achieved if either too much time is taken (TF > 99), or if the energy of the E ship gets too low (EE < 20).

With these rules in place, we're ready for our first trek. Here we go:

```
9  -  -  -  -  -  -  -  *
8  -  -  -  -  -  -  -  *  *
7  -  -  -  -  -  -  -  -  -
6  -  -  -  -  -  -  -  -  -
5  -  -  -  -  -  -  -  -  -
4  -  -  -  -  -  -  -  -  -
3  -  -  -  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -  -
1  E  -  -  -  -  -  -  -  -
   1  2  3  4  5  6  7  8  9   EE = 100    TF = 1

YOUR MOVE (MAX. DIST. = EE/100):    X,Y =? 2,2
INSUFFICIENT ENERGY, DISTANCE REQUESTED = 1.41421
YOUR MOVE (MAX. DIST. = EE/100):    X,Y =? 1,1
RESTING: ENERGY INCREASED 10%
K-MINE RELEASED AT 1 , 7
PRESS ENTER FOR NEXT TIME FRAME  --READY?
```

In this first time frame, E tries to move a distance of 1.414 (from 1,1 to 2,2). But with an energy of EE = 100, the maximum distance allowed is 100/100 = 1. So E decides to rest (stay at 1,1) instead.

```
9   -  -  -  -  -  -  -  *  G
8   -  -  -  -  -  -  -  *  *
7   K  -  -  -  -  -  -  -  -
6   -  -  -  -  -  -  -  -  -
5   -  -  -  -  -  -  -  -  -
4   -  -  -  -  -  -  -  -  -
3   -  -  -  -  -  -  -  -  -
2   -  -  -  -  -  -  -  -  -
1   E  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9   EE = 110    TF = 2

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 1,1
RESTING: ENERGY INCREASED 10%
K-MINE RELEASED AT 2 , 2
PRESS ENTER FOR NEXT TIME FRAME  --READY?
```

Notice that the energy of the E ship is now EE = 110, a 10% increase from resting. Also notice that the K-mine released in time frame 1 is now displayed at 1,7. E continues to rest to build up its energy, EE.

```
9   -  -  -  -  -  -  -  *  G
8   -  -  -  -  -  -  -  *  *
7   K  -  -  -  -  -  -  -  -
6   -  -  -  -  -  -  -  -  -
5   -  -  -  -  -  -  -  -  -
4   -  -  -  -  -  -  -  -  -
3   -  -  -  -  -  -  -  -  -
2   -  K  -  -  -  -  -  -  -
1   E  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9   EE = 121    TF = 3

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 1,1
RESTING: ENERGY INCREASED 10%
K-MINE RELEASED AT 5 , 9
PRESS ENTER FOR NEXT TIME FRAME  --READY?
```

EE is now 121, but more K-mines are starting to appear. The star gate at 9,9 (which was open in time frame 1) is closed, but E isn't close enough to worry about this yet.

```
9   -  -  -  -  K  -  -  *  G
8   -  -  -  -  -  -  -  *  *
7   K  -  -  -  -  -  -  -  -
6   -  -  -  -  -  -  -  -  -
5   -  -  -  -  -  -  -  -  -
4   -  -  -  -  -  -  -  -  -
3   -  -  -  -  -  -  -  -  -
2   -  K  -  -  -  -  -  -  -
1   E  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9   EE = 133    TF = 4

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 1,1
RESTING: ENERGY INCREASED 10%
K-MINE RELEASED AT 8 , 7
PRESS ENTER FOR NEXT TIME FRAME  --READY?
```

A K-mine has appeared very close to E (at 2,2). But this is still a miss. E decides to rest one more time and then...

```
9   -  -  -  -  K  -  -  *
8   -  -  -  -  -  -  -  *  *
7   K  -  -  -  -  -  -  K  -
6   -  -  -  -  -  -  -  -  -
5   -  -  -  -  -  -  -  -  -
4   -  -  -  -  -  -  -  -  -
3   -  -  -  -  -  -  -  -  -
2   -  K  -  -  -  -  -  -  -
1   E  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9   EE = 146    TF = 5

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 2,2
K-MINE RELEASED AT 5 , 5
######:::::  !!!  ::::::######
YOU HAVE HIT A K-MINE.  YOU'RE BLOWN BACK TO START.
ENERGY LOSS IS 30%
PRESS ENTER FOR NEXT TIME FRAME  --READY?
```

E becomes aggressive and deliberately (?) moves to 2,2. The explosion blows E back to 1,1 and reduces EE by 30%, but...

```
9   -  -  -  -  K  -  -  *
8   -  -  -  -  -  -  -  *  *
7   K  -  -  -  -  -  -  K  -
6   -  -  -  -  -  -  -  -  -
5   -  -  -  -  K  -  -  -  -
4   -  -  -  -  -  -  -  -  -
3   -  -  -  -  -  -  -  -  -
2   -  -  -  -  -  -  -  -  -
1   E  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9   EE = 102    TF = 6

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 1,1
RESTING: ENERGY INCREASED 10%
K-MINE RELEASED AT 1 , 3
PRESS ENTER FOR NEXT TIME FRAME  --READY?
```

In time frame 6 you can see that the K-mine at 2,2 has been disintegrated by E's previous move.

```
9   -  -  -  -  K  -  -  *
8   -  -  -  -  -  -  -  *  *
7   K  -  -  -  -  -  -  K  -
6   -  -  -  -  -  -  -  -  -
5   -  -  -  -  K  -  -  -  -
4   -  -  -  -  -  -  -  -  -
3   K  -  -  -  -  -  -  -  -
2   -  -  -  -  -  -  -  -  -
1   E  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9   EE = 112    TF = 7

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 1,1
RESTING: ENERGY INCREASED 10%
K-MINE RELEASED AT 4 , 7
PRESS ENTER FOR NEXT TIME FRAME  --READY?
```

E decides to lie low for the next 4 time frames and rebuild energy. (Notice that the star gate is open again. This happens about 30% of the time.)

```
9   -  -  -  -  K  -  -  *  G
8   -  -  -  -  -  -  -  *  *
7   K  -  -  K  -  -  -  K  -
6   -  -  -  -  -  -  -  -  -
5   -  -  -  -  K  -  -  -  -
4   -  -  -  -  -  -  -  -  -
3   K  -  -  -  -  -  -  -  -
2   -  -  -  -  -  -  -  -  -
1   E  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9   EE = 124    TF = 8

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 1,1
RESTING: ENERGY INCREASED 10%
K-MINE RELEASED AT 6 , 9
PRESS ENTER FOR NEXT TIME FRAME  --READY?
```

Meanwhile, the K-mines keep on coming. So far all the K-mines are showing up at new positions. However, it's possible for a K-mine to be released at the same position twice.

```
9   -  -  -  -  K  K  -  *  G
8   -  -  -  -  -  -  -  *  *
7   K  -  -  K  -  -  -  K  -
6   -  -  -  -  -  -  -  -  -
5   -  -  -  -  K  -  -  -  -
4   -  -  -  -  -  -  -  -  -
3   K  -  -  -  -  -  -  -  -
2   -  -  -  -  -  -  -  -  -
1   E  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9   EE = 136    TF = 9

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 1,1
RESTING: ENERGY INCREASED 10%
K-MINE RELEASED AT 5 , 7
PRESS ENTER FOR NEXT TIME FRAME  --READY?
```

One suggestion for extending the game is to mark a position with two K-mines differently, and make this a mine that can "reach out" to impact E. More on this idea later.

```
9   -  -  -  -  K  K  -  *  G
8   -  -  -  -  -  -  -  *  *
7   K  -  -  K  K  -  -  K  -
6   -  -  -  -  -  -  -  -  -
5   -  -  -  -  K  -  -  -  -
4   -  -  -  -  -  -  -  -  -
3   K  -  -  -  -  -  -  -  -
2   -  -  -  -  -  -  -  -  -
1   E  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9   EE = 150    TF = 10

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 2,2
K-MINE RELEASED AT 7 , 7
PRESS ENTER FOR NEXT TIME FRAME  --READY?
```

E decides to start moving, and goes to 2,2.

```
9   -  -  -  -  K  K  -  *  G
8   -  -  -  -  -  -  -  *  *
7   K  -  -  K  K  -  K  K  -
6   -  -  -  -  -  -  -  -  -
5   -  -  -  -  K  -  -  -  -
4   -  -  -  -  -  -  -  -  -
3   K  -  -  -  -  -  -  -  -
2   -  E  -  -  -  -  -  -  -
1   -  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9   EE = 150    TF = 11

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 3,3
K-MINE RELEASED AT 1 , 9
PRESS ENTER FOR NEXT TIME FRAME   --READY?
```

Another diagonal move by E, this time to 3,3.

```
9   K  -  -  -  K  K  -  *
8   -  -  -  -  -  -  -  *  *
7   K  -  -  K  K  -  K  K  -
6   -  -  -  -  -  -  -  -  -
5   -  -  -  -  K  -  -  -  -
4   -  -  -  -  -  -  -  -  -
3   K  -  E  -  -  -  -  -  -
2   -  -  -  -  -  -  -  -  -
1   -  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9   EE = 150    TF = 12

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 4,4
K-MINE RELEASED AT 3 , 4
PRESS ENTER FOR NEXT TIME FRAME   --READY?
```

E goes to 4,4 but realizes that to move more than "1 over and 1 up," more energy is needed...

```
9   K  -  -  -  K  K  -  *
8   -  -  -  -  -  -  -  *  *
7   K  -  -  K  K  -  K  K  -
6   -  -  -  -  -  -  -  -  -
5   -  -  -  -  K  -  -  -  -
4   -  -  K  E  -  -  -  -  -
3   K  -  -  -  -  -  -  -  -
2   -  -  -  -  -  -  -  -  -
1   -  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9   EE = 150    TF = 13

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 4,4
RESTING: ENERGY INCREASED 10%
K-MINE RELEASED AT 5 , 6
PRESS ENTER FOR NEXT TIME FRAME   --READY?
```

...so E rests again. Now 10% of 150 = 15, so the new EE should = 165.

```
9   K  -  -  -  K  K  -  *
8   -  -  -  -  -  -  -  *  *
7   K  -  -  K  K  -  K  K  -
6   -  -  -  -  K  -  -  -  -
5   -  -  -  -  K  -  -  -  -
4   -  -  K  E  -  -  -  -  -
3   K  -  -  -  -  -  -  -  -
2   -  -  -  -  -  -  -  -  -
1   -  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9   EE = 165    TF = 14

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 4,4
RESTING: ENERGY INCREASED 10%
K-MINE RELEASED AT 7 , 3
PRESS ENTER FOR NEXT TIME FRAME  --READY?
```

More rest. A quick calculation shows that to move 2 units in one direction requires at least 200 units of energy, so...

```
9   K  -  -  -  K  K  -  *
8   -  -  -  -  -  -  -  *  *
7   K  -  -  K  K  -  K  K  -
6   -  -  -  -  K  -  -  -  -
5   -  -  -  -  K  -  -  -  -
4   -  -  K  E  -  -  -  -  -
3   K  -  -  -  -  -  K  -  -
2   -  -  -  -  -  -  -  -  -
1   -  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9   EE = 181    TF = 15

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 4,4
RESTING: ENERGY INCREASED 10%
K-MINE RELEASED AT 9 , 6
PRESS ENTER FOR NEXT TIME FRAME  --READY?
```

...E rests again. If E should want to go 2 units in one direction and 1 unit in another, the distance is SQR(1 + 4) = 2.236, so EE will have to be 224. (If you look ahead to TF = 17 you'll see that E didn't know this.)

```
9   K  -  -  -  K  K  -  *
8   -  -  -  -  -  -  -  *  *
7   K  -  -  K  K  -  K  K  -
6   -  -  -  -  K  -  -  -  K
5   -  -  -  -  K  -  -  -  -
4   -  -  K  E  -  -  -  -  -
3   K  -  -  -  -  -  K  -  -
2   -  -  -  -  -  -  -  -  -
1   -  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9   EE = 199    TF = 16

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 4,4
RESTING: ENERGY INCREASED 10%
K-MINE RELEASED AT 9 , 4
PRESS ENTER FOR NEXT TIME FRAME  --READY?
```

One more rest. E seems to be lucky, not being hit despite all the K-mines being released.

```
9   K  -  -  -  K  K  -  *  G
8   -  -  -  -  -  -  -  *  *
7   K  -  -  K  K  -  K  K  -
6   -  -  -  -  K  -  -  -  K
5   -  -  -  -  K  -  -  -  -
4   -  -  K  E  -  -  -  -  K
3   K  -  -  -  -  -  K  -  -
2   -  -  -  -  -  -  -  -  -
1   -  -  -  -  -  -  -  -  -
    1  2  3  4  5  6  7  8  9   EE = 219   TF = 17

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 6,5
INSUFFICIENT ENERGY, DISTANCE REQUESTED = 2.23607
YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 6,4
K-MINE RELEASED AT 8 , 1
PRESS ENTER FOR NEXT TIME FRAME  --READY?
```

Another miss by the K-mine. Just what is the chance of being hit?

```
9   K  -  -  -  K  K  -  *  G
8   -  -  -  -  -  -  -  *  *
7   K  -  -  K  K  -  K  K  -
6   -  -  -  -  K  -  -  -  K
5   -  -  -  -  K  -  -  -  -
4   -  -  K  -  -  E  -  -  K
3   K  -  -  -  -  -  K  -  -
2   -  -  -  -  -  -  -  -  -
1   -  -  -  -  -  -  -  K  -
    1  2  3  4  5  6  7  8  9   EE = 219   TF = 18

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 8,4
K-MINE RELEASED AT 3 , 1
PRESS ENTER FOR NEXT TIME FRAME  --READY?
```

Missed again! The reason is that the probability of a K-mine having the same X coordinate as E is 1/9, and the probability that it will have the same Y coordinate is also 1/9, but the probability that both coordinates will be the same is $(1/9) * (1/9) = 1/81$.

```
9   K  -  -  -  K  K  -  *  G
8   -  -  -  -  -  -  -  *  *
7   K  -  -  K  K  -  K  K  -
6   -  -  -  -  K  -  -  -  K
5   -  -  -  -  K  -  -  -  -
4   -  -  K  -  -  -  -  E  K
3   K  -  -  -  -  -  K  -  -
2   -  -  -  -  -  -  -  -  -
1   -  -  K  -  -  -  -  K  -
    1  2  3  4  5  6  7  8  9   EE = 219   TF = 19

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 8,6
K-MINE RELEASED AT 4 , 5
PRESS ENTER FOR NEXT TIME FRAME  --READY?
```

Made more confident by this lesson from probability theory, E decides to go on the move again.

```
9   K  -  -  -  K  K  -  *  G
8   -  -  -  -  -  -  -  *  *
7   K  -  -  K  K  -  K  K  -
6   -  -  -  -  K  -  -  E  K
5   -  -  -  K  K  -  -  -  -
4   -  -  K  -  -  -  -  -  K
3   K  -  -  -  -  -  K  -  -
2   -  -  -  -  -  -  -  -  -
1   -  -  K  -  -  -  -  K  -
    1  2  3  4  5  6  7  8  9   EE = 219    TF = 20

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 9,7
K-MINE RELEASED AT 4 , 9
PRESS ENTER FOR NEXT TIME FRAME   --READY?
```

Of course, E must be careful not to land on a K-mine already planted, or hit a * (and be bounced back to the start).

```
9   K  -  -  K  K  K  -  *  G
8   -  -  -  -  -  -  -  *  *
7   K  -  -  K  K  -  K  K  E
6   -  -  -  -  K  -  -  -  K
5   -  -  -  K  K  -  -  -  -
4   -  -  K  -  -  -  -  -  K
3   K  -  -  -  -  -  K  -  -
2   -  -  -  -  -  -  -  -  -
1   -  -  K  -  -  -  -  K  -
    1  2  3  4  5  6  7  8  9   EE = 219    TF = 21

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 9,7
RESTING: ENERGY INCREASED 10%
K-MINE RELEASED AT 3 , 2
PRESS ENTER FOR NEXT TIME FRAME   --READY?
```

E is now as close as possible to G, but the gate is closed. So E stays put, building up more energy (to survive any possible hit by a new K-mine).

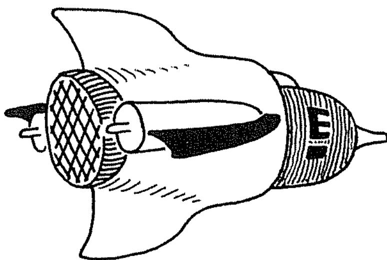
```
9   K  -  -  K  K  K  -  *  G
8   -  -  -  -  -  -  -  *  *
7   K  -  -  K  K  -  K  K  E
6   -  -  -  -  K  -  -  -  K
5   -  -  -  K  K  -  -  -  -
4   -  -  K  -  -  -  -  -  K
3   K  -  -  -  -  -  K  -  -
2   -  -  K  -  -  -  -  -  -
1   -  -  K  -  -  -  -  K  -
    1  2  3  4  5  6  7  8  9   EE = 241    TF = 22

YOUR MOVE (MAX. DIST. = EE/100):   X,Y =? 9,7
RESTING: ENERGY INCREASED 10%
K-MINE RELEASED AT 4 , 7
PRESS ENTER FOR NEXT TIME FRAME   --READY?
```

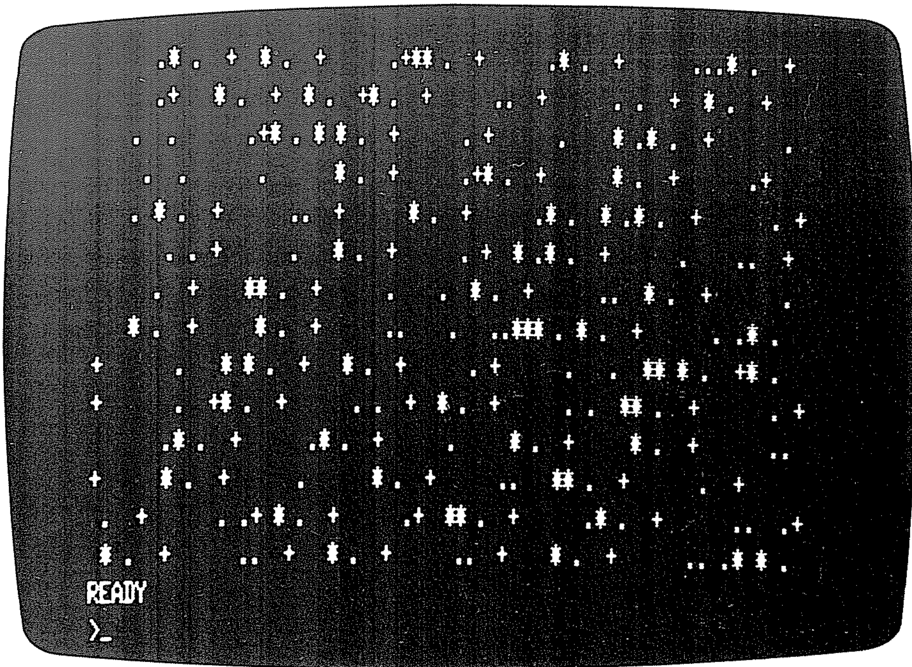
The gate is still closed, so E continues to rest (and start sweating).

```
9   K  -  -  K  K  K  -  *
8   -  -  -  -  -  -  -  *  *
7   K  -  -  K  K  -  K  K  E
6   -  -  -  -  K  -  -  -  K
5   -  -  -  K  K  -  -  -  -
4   -  -  K  -  -  -  -  -  K
3   K  -  -  -  -  -  K  -  -
2   -  -  K  -  -  -  -  -  -
1   -  -  K  -  -  -  -  K  -
    1  2  3  4  5  6  7  8  9    EE = 265    TF = 23

YOUR MOVE (MAX. DIST. = EE/100):    X,Y =? 9,9
****************************************
****************************************
*** YOU HAVE ACHIEVED STAR DIMENSION ***
***     YOUR RATING IS 110.76        ***
****************************************
****************************************
```

The gate is open! E quickly types 9,9 and makes it through to the next dimension (another galaxy with another set of game rules?).

As a little surprise, if you achieve Star Dimension the game ends with a star pattern that we can't really show in a book. The photo gives some idea of what it looks like, but there is also a random on/off motion of the star symbols you'll want to see on a real screen.





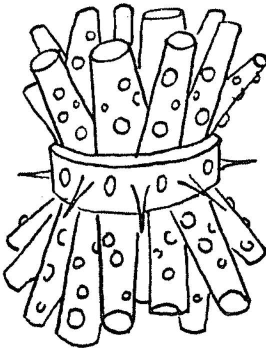

*When the player of BABYQ finally makes it through the star gate, a score is printed followed by a blinking pattern of stars. These are created by lines 2050 to 2070 in the program.*

## High-Level Design; Stepwise Refinement

From our run you can see that the BABYQ program is going to be a little more complicated than most of the other games we've described, so trying to do the high-level design in one "pass" won't be easy. It's time to introduce another professional programming technique called *stepwise refinement.*

The idea is to first develop a high-level design that describes only the major steps in the program, sweeping as many details as possible under the rug. Then we'll go back and expand each of these major steps into a medium-level design. The language used will still be English, but it will start to sound more like extended BASIC. (This is why some books call the instruction written at this stage *pseudo code.* It's code that is almost understandable by a real computer, but not quite.)

Using this approach, we'll first describe the design of BABYQ in terms of eight major steps, and then see how these can be refined into pseudo code, followed by a translation into the actual code of BASIC:



1. Get things set up. Initialize the board, energy allocation, positions of objects, time frame, etc.
2. Set up a loop to run through the game's various time frames.
3. Display the game board to the player(s).
4. Give the E ship its turn to play; check validity of the move.
5. Move the E ship; check for star barrier or star gate collision.
6. Give the enemy K creatures their turn; check validity of the move.
7. Check for a collision between E and a K-mine; take appropriate action where called for.
8. If the game is finished display the final results; otherwise go back to 2 for a new time frame.

As you can see, the instructions in our program are fairly general, and they next need to be spelled out in greater detail. For example, step 4 could be refined into more specific steps as follows:

4(a). Request E to type in the new position desired.
4(b). If E didn't stay within the 9 by 9 board limit, repeat the input request.
4(c). Calculate the distance E wants to move.
4(d). If E doesn't have enough energy to move the distance requested, repeat the input request.

Step 5 could be refined into five steps:

5(a). Erase the old E symbol by replacing it with a "-".
5(b) Put the new E coordinates in XE and YE. If E has bumped into the star barrier, go to a special "barrier routine."
5(d). If E is trying to go through the star gate, go to a special "gate routine."
5(e). If E didn't move at all increase its energy, EE, by 10%.

Step 6 might be refined as four finer steps.

6(a). Use the random number generator RND(0) to determine the coordinates XK and YK for the K-mine.

6(b). Make sure a mine wasn't placed in an illegal position (in the star gate or on the star barrier). If it was, go back to 6(a).

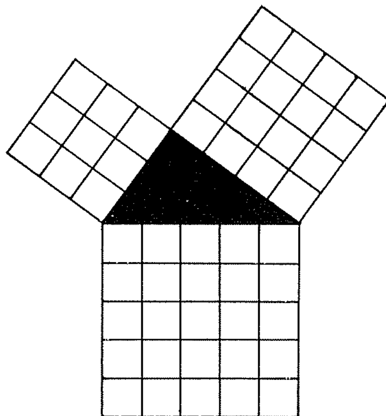
6(c). When the K-mine position is legal, put the symbol K in position XK,YK of the string array that represents the game board.

6(d). Print a message telling the player where the K-mine was released.

Similar refinements would be made for the other steps in our program. There is no single best way in which to do this refinement. There is plenty of room for individual variation and style among programmers. Try your hand at it; you could very well come up with a better approach.

## Coding the Program

The coding of BABYQ shown in the following listing adheres closely to the high-level design just discussed. An additional help that should be given to readers of longer programs is a table showing what variables are used, and what they mean. Here's such a table for BABYQ, followed by the program itself:

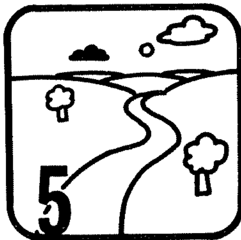


| Variable | Meaning |
|---|---|
| A$(I,J) | The 9 by 9 string array to hold the game board symbols. I points to the row in the array, and J points to the column. |
| XE,YE | The X and Y coordinates of the E ship's present position. |
| TF | Time frame (number of plays). |
| EE | Energy of the E ship. |
| X,Y | Coordinates for the E ship's desired next position (input by the player). |
| DE | Distance from E ship's present position (XE,YE) to the desired next position (X,Y). |
| XK,YK | The X and Y coordinates of the latest K-mine. |

**NOTE:** The distance DE is calculated using the Pythagorean theorem which says that DE = "the square root of the sum of the squares of the two sides of the right triangle which has DE as its hypotenuse."

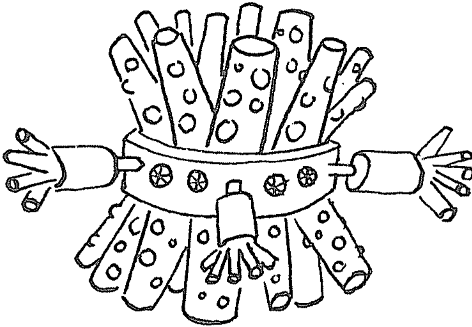
```
10 '**********************************************************
11 '*           BABYQ   (QUEST TYPE SPACE GAME)               *
12 '**********************************************************
20 CLS
100 '-------------------------------------------------------
101 '          INITIALIZE VARIABLES
102 '-------------------------------------------------------
110 PRINT"*** BABY QUEST RUNNING -- STAND BY ***"
120 DIM A$(9,9)
130 FOR I = 1 TO 9
140   FOR J = 1 TO 9
150     LET A$(I,J) = "-"
160   NEXT J
170 NEXT I
180 XE = 1: YE = 1: TF = 0: EE = 100
190 A$(9,9) = "G": A$(9,8) = "*": A$(8,9) = "*": A$(8,8) = "*"
200 '=======================================================
201 '          MAIN LOOP STARTS HERE
202 '
210 TF = TF + 1: CLS
215 IF RND(0) < 0.3 THEN A$(9,9) = " " ELSE A$(9,9) = "G"
220 A$(XE,YE) = "E"
300 '
301 '----------DISPLAY UNIVERSE----------------------------
310 FOR Y = 9 TO 1 STEP -1: PRINT Y;"  ";
320   FOR X = 1 TO 9
330     PRINT A$(X,Y);"   ";
340   NEXT X
350   PRINT
360 NEXT Y
370 PRINT"     1   2   3   4   5   6   7   8   9 ";
380 PRINT TAB(43);"EE =";INT(EE);"  TF =";TF
400 '
401 '----------INPUT & PROCESS E MOVE---------------------
410 PRINT"YOUR MOVE (MAX. DIST. = EE/100):   X,Y =";:INPUT X,Y
420 IF X<1 OR X>9 OR Y<1 OR Y>9 PRINT"ILLEGAL COORD.":GOTO 410
430 DE = SQR((X-XE)[2 + (Y-YE)[2)
440 IF DE <= EE/100 THEN 510
450 PRINT"INSUFFICIENT ENERGY, DISTANCE REQUESTED =";DE
460 GOTO 410
500 '
501 '----------MOVE E SHIP; CHECK FOR GATE COLLISION-------
510 A$(XE,YE) = "-"
520 XE = INT(X): YE = INT(Y)
530 IF A$(XE,YE) = "*" THEN 1000
540 IF XE = 9 AND YE = 9 THEN 2000
550 IF DE = 0 PRINT"RESTING: ENERGY INCREASED 10%":EE = 1.1*EE
600 '
601 '----------PLACE A K-MINE------------------------------
610 XK = INT(9 * RND(0) + 1)
620 YK = INT(9 * RND(0) + 1)
630 IF XK * YK >= 64 THEN 610
640 A$(XK,YK) = "K"
650 PRINT"K-MINE RELEASED AT";XK;",";YK
700 '
701 '----------CHECK FOR COLLISION WITH K-MINE------------
710 IF A$(XE,YE) = "-" THEN 810
720 PRINT"######:::::  !!!  ::::::######"
730 PRINT"YOU HAVE HIT A K-MINE.  YOU'RE BLOWN BACK TO START."
740 PRINT"ENERGY LOSS IS 30%": EE = .7 * EE
750 A$(XE,YE) = "-"
760 XE = 1: YE = 1
```

```
800 '
801 '----------CHECK FOR END OF GAME----------------------
810 IF EE < 20 OR TF > 99 THEN 3000
820 PRINT"PRESS ENTER FOR NEXT TIME FRAME  --READY";: INPUT A$
900 GOTO 210
990 '
998 '                  END OF THE MAIN LOOP
999 '=================================================
1000 '-------------------------------------------------
1001 '                STAR GATE BARRIER COLLISION
1002 '-------------------------------------------------
1016 PRINT: PRINT"YOU HIT THE STAR GATE BARRIER"
1020 PRINT "POW!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
1025 PRINT "IT'S BACK TO START FOR YOU"
1030 XE=1: YE=1
1090 GOTO 820
2000 '-------------------------------------------------
2001 '                STAR DIMENSION CHECK
2002 '-------------------------------------------------
2005 IF A$(9,9)="G" PRINT "YOU TRIED TO ENTER A CLOSED GATE
     ..... YOU ARE SENTENCED TO START OVER":XE=1:YE=1:GOTO 820
2007 '
2008 ' ---------------VICTORY !!! ---------------
2009 PRINT "*****************************************"
2010 PRINT "*****************************************"
2020 PRINT "*** YOU HAVE ACHIEVED STAR DIMENSION ***"
2025 PRINT "***    YOUR RATING IS";10*EE/(TF+1);TAB(37);"****"
2026 PRINT "*****************************************"
2028 PRINT "*****************************************"
2030 FOR I=1 TO 1000:NEXT I
2050 FOR I= 1 TO 500
2060 PRINT @ RND(1010)-1, "* .  +      .";
2070 NEXT I
2090 GOTO 10000
3000 '-------------------------------------------------
3001 '                E OUT OF ENERGY
3002 '-------------------------------------------------
3010 PRINT "YOU ARE FINISHED --- THIS IS THE END"
3020 PRINT "ENERGY ="; EE; "TIME FRAME ="; TF
10000 END
```



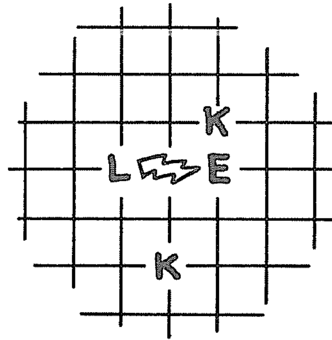

## Future Extensions

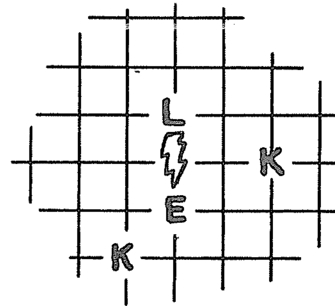
In the present version of the game, the K-mines are placed at random positions of the grid, so it's possible for more than one K-mine to be placed in the same position. When that happens, a fancier game might convert the original K-mine into an L-mine (L for "lollapalooza"). The L-mines could, like the more advanced pieces in chess, be given the ability to "reach out" and hit the E ship from a distance. Another extension would be to add a refueling base where the E ship could go to increase EE. However, this base would sometimes disappear, showing up again at a new position that was randomly selected.



## Project *BABYZAP*

Extend BABYQ so that repeated K-mines change into L-mines, where L-mines act just like K-mines when they hit E directly. However, they should also be able to hit the E ship with "ray guns." These rays can shoot only in directions parallel to the X or Y axes, and only have an effect up to a distance of 2 units. A ray gun hit shouldn't blow the E ship back to start, but it should deplete the E ship's energy by 20/DL percent, where DL is the distance from E to the L-mine (ie: energy should decrease by 20% or 10% for distances of 1 or 2). When an L-mine shoots its ray gun, it should be weakened and revert to a K-mine.



## 2.6 GAMES AND GRAPHICS. SETPLOT, TRIGPIX, *ARROW2*, *(ELBLASTO)*

Computer output is what you see printed on a video screen or on the paper of a computer printer. It is usually made up of *alphanumeric* characters. These include the alaphabet, numbers, and special symbols (like #, −, or *). Using the TAB(X) function, it's possible to produce pictorial or *graphical* output with these symbols. An example of an alphanumeric graph was shown in section 1.8 in the project *TABGRAPH*. The plotting there was done using the asterisk (*) symbol.

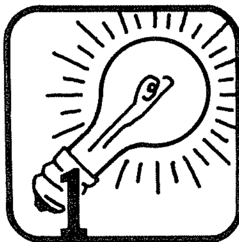
Several versions of BASIC permit you to put other graphical symbols on the video screen. Some also have statements that allow you to turn small "points" (which are actually rectangles of light) on or off. Since these points are smaller than the alphanumeric characters, more of them can be plotted per inch, giving a picture of higher resolution.

On some computers (eg: the Apple II) the BASIC statement that plots a small point on the screen at a position X units to the right, and Y units down is PLOT X,Y. On the TRS-80 computer, the plotting statement is SET(X,Y), while RESET(X,Y) is used to erase a point.

The values of X and Y can be calculated by using assignment (LET) statements in the program. On the TRS-80, setting X to values from 0 to 127 would move the plotting point from the extreme left to the extreme right, while setting Y to values from 0 to 47 would move the point from top to bottom. The choice of these numbers was a TRS-80 hardware design decision. Other machines use different values.

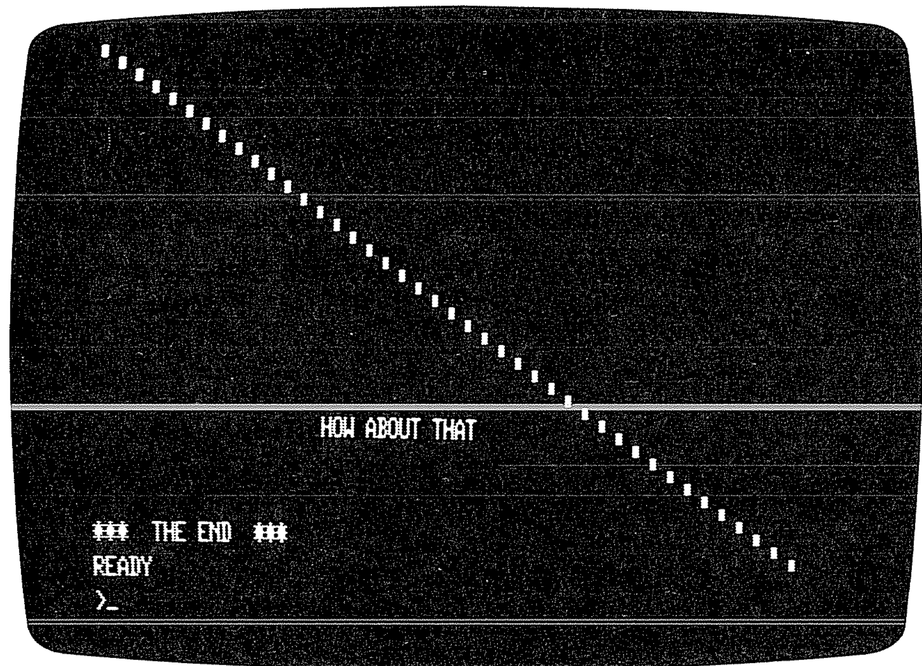
### **Program** SETPLOT

#### The Idea



Here's a program to plot 43 points on a TRS-80 screen. The points form an approximation to a straight line that goes from the upper left hand corner toward the lower right hand corner of the screen. Each successive point is plotted at a position that is 3 units over and 1 unit down from the previous position.

## Sample Run




*The output of SETPLOT is an approximation to a straight line made up of 43 SET(X,Y) plotting symbols (small rectangles), followed by the printing of some messages at different parts of the screen using the PRINT @ feature. When the program finishes, the computer prints READY, and this erases the last plot symbol. To avoid this, add the "stall" statement 100 GOTO 100. If you do this, you'll have to press BREAK to get out of the program.*

## Coding the Program

For this short program we'll dispense with the high-level design and go right to the code. The first point is plotted at the position (0,0). Then "displacements" of 3 and 1 are added to (0,0), giving (3,1). This addition continues in a FOR loop, giving points at (6,2), (9,3), and so on up to (126,42).

```
10 REM --- SETPLOT (SHOWS USE OF SET(X,Y) AND PRINT @ ---
12 CLS      'THIS MEANS CLEAR SCREEN ON THE TRS-80
15 X=0 : Y=0
20 SET(X,Y)
30 FOR P=1 TO 42
40   LET X=X+3
50   LET Y=Y+1
60   SET(X,Y)
70 NEXT P
80 PRINT @ 660, "HOW ABOUT THAT";
90 PRINT @ 832, "***  THE END  ***"
```

This program also illustrates the use of the PRINT @ feature in TRS-80 BASIC. This allows you to start printing alphanumeric information at any one of 1024 positions on the screen. The positions on the first line are numbered 0, 1, 2, 3, and so on up to 63. So the second line begins with position 64, the third line begins with position 128, and so on. To print the message "HOW ABOUT THAT" on line 13, 20 positions to the right on the video display you would use

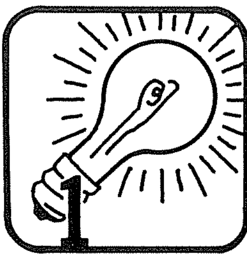
60 PRINT @ 13 * 64 + 20, "HOW ABOUT THAT";

or simply

60 PRINT @ 852, "HOW ABOUT THAT";

The semicolon (;) is used at the end if you want to suppress the normal linefeed after a PRINT (which would erase the next line).

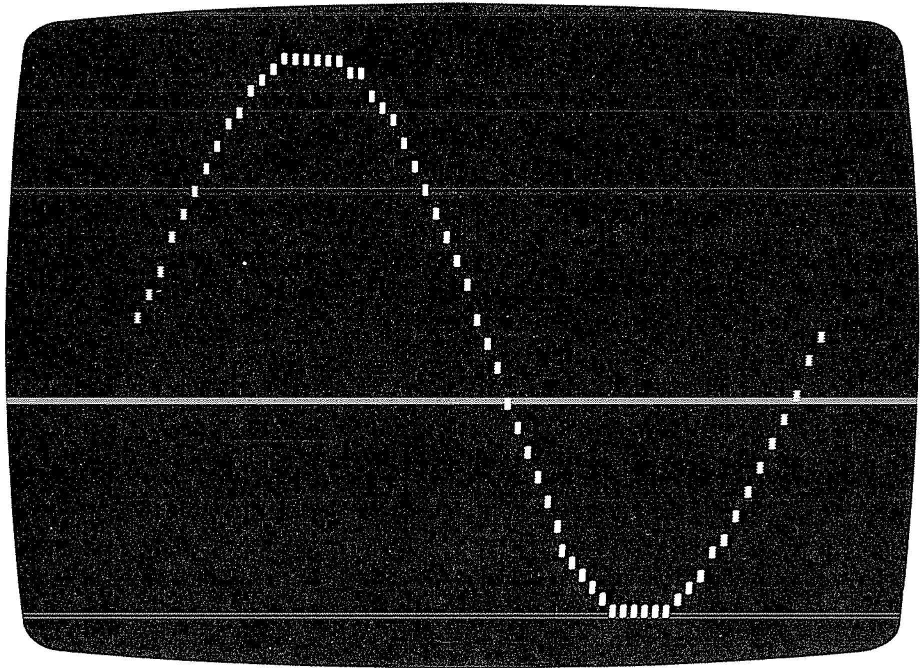
## Program TRIGPIX




### The Idea

Graphs of values of X and Y determined by more complicated mathematical formulas can also be plotted. Here's a program that plots the SIN(X) function for 64 values of the angle X, for X = 0, 0.1, 0.2, 0.3, and so on up to 6.3.

## Sample Run



*The program TRIGPIX uses the SET(X, Y) function to plot a curve generated by one cycle of the sine function, scaling the output to fill the screen.*

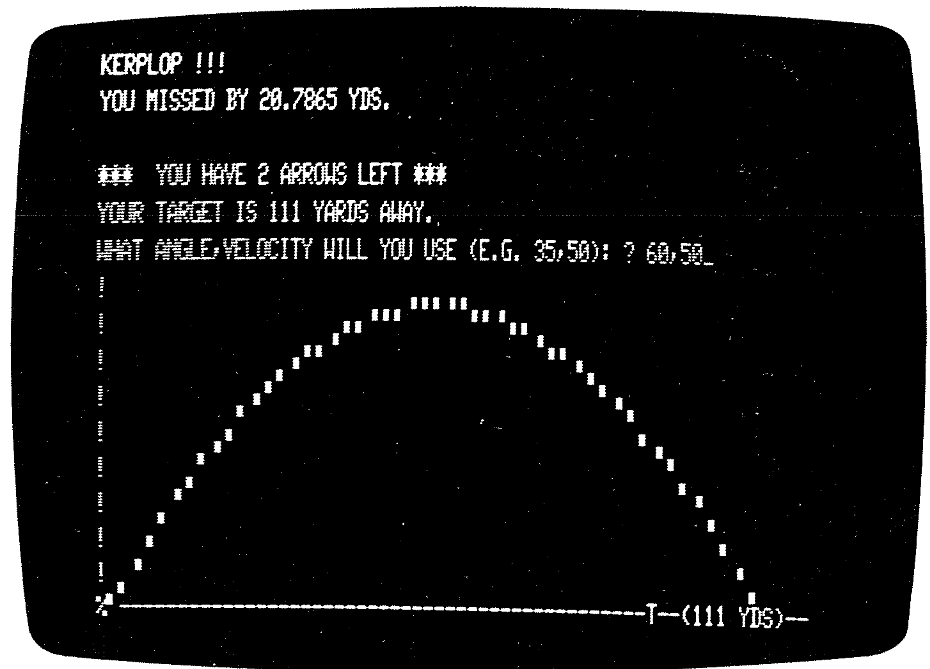
## Coding the Program

Again we'll skip the high-level design and go right to the code. Since X goes from 0 to 6.3, and trigonometry books tell us that SIN(X) goes from −1 to +1, the values of X and Y have to be scaled from these mathematical values to TRS-80 "plotting values." The scaled plotting values corresponding to X and Y are called X1 and Y1 in our program.

```
10 REM --- TRIGPIX (USES SET(X,Y) TO PLOT SIN(A) ON TRS-80)
20 CLS
30 FOR X=0 TO 6.3 STEP .1
40   LET X1=20*X
50   LET Y=SIN(X)
60   LET Y1=47-23.5*(Y+1)
70   SET(X1,Y1)
80 NEXT X
```

## Project ARROW2

Study the subject of *projectile motion* (the book *BASIC and the Personal Computer* explains the key formulas on pages 265 and 267). Then write a program that allows the user to shoot an arrow at a randomly placed target. The trajectory should be displayed using the SET(X,Y) statement. Here is a photo of what the output might look like.




*Here's a sample of what the output of ARROW2 should look like. The target T is always printed in the same place, but the distance this represents is chosen randomly for each run.*

## Superproject *(ELBLASTO)*

Solutions to superprojects are not given in the book, so you're on your own for this one. After studying *ARROW2*, write a program that "shoots" the great El Blasto from a circus cannon toward a net. The user should be allowed to adjust the angle of the cannon, and the initial velocity for each shot. When El Blasto misses the net, a CRUNCH!! message should be printed at the approximate spot. When he hits the net, he should bounce up with joy. You might also want to make use of double-sized letters (available on the TRS-80) for the CRUNCH!! or other) messages. The technique for producing double-sized letters was explained in section 2.4 as part of project *SCRAM2*. Since switching to double-sized letters on the TRS-80 affects the whole screen, it is usually preceded by a clear screen

command (CLS); switching back to regular-sized letters is also accomplished with the clear screen command.

### Exercises for Chapter 2

Here are some more exercises to try before moving ahead. Try them as pencil and paper exercises first, and then check your results on a computer.

```
10 REM %%% EXERCISE 5.  FILL IN THE MISSING ----- PARTS %%%
20 PRINT "TYPE 3 NUMBERS SEPARATED BY COMMAS ";
30 INPUT -----
40 PRINT "WHAT IS"; A; "+"; B; "+"; C
50 INPUT X
60 IF ----- THEN 90
70    PRINT "NOPE. ANSWER WAS"; -----
80    GOTO -----
90 PRINT "VERY GOOD !!!"
100 PRINT "WANT ANOTHER (Y=YES)";
110 INPUT R$
120 IF -----  THEN 20
130    PRINT "SO LONG FOR NOW"
140 END
```

```
10 REM %%% EXERCISE 6.  SIMULATE A RUN OF THIS PROGRAM
20 FOR I=1 TO 5
30   FOR J=1 TO 6
40   PRINT TAB(10*(J-1));I*J;
50   NEXT J
60   PRINT
70 NEXT I
80 END
```

```
100 REM %%% EXERCISE 7.  SIMULATE A RUN OF THIS PROGRAM %%%
110 DIM W$(21)
120 PRINT "AFTER EACH ? TYPE ONE WORD OF A SENTENCE."
130 PRINT "TYPE A PERIOD WHEN DONE.  LIMIT IS 20 WORDS."
140 REM ----- INPUT WORDS INTO STRING ARRAY -----
145 K=1
150 IF K>20 THEN 210
160   PRINT "WORD # "; K;
170   INPUT W$(K)
180   IF W$(K)= "." THEN 210
185   LET W$(K)=W$(K) + " "
190   K=K+1
195 GOTO 150
210 PRINT: PRINT "HERE'S ANOTHER WAY OF SAYING THAT":PRINT
220 FOR J=K TO 1 STEP -1
230   PRINT W$(J);
240 NEXT J
250 PRINT: PRINT: PRINT "DONE"
260 END
```

## SOURCES FOR FURTHER INFORMATION FOR CHAPTER 2

From the long, historical point of view, computer games are simply the latest extension of an already rich cultural invention. So if you get serious about creating your own computer games, you should take a look at this heritage. There is at least one historian, Huizinga, who argues that all of culture reveals a gamelike quality. The following list of references is a good starting point for further study.


1. Ahl, David. *BASIC Computer Games.* (Morristown, N.J.: Creative Computing Press, 1978).

2. Falkener, Edward. *Games Ancient and Oriental and How to Play Them.* (New York: Dover Press, 1961).

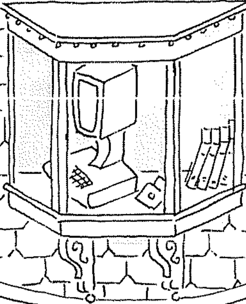
3. Huizinga, Johan. *Homo Ludens, A Study of the Play-Element in Culture.* (Boston: The Beacon Press, 1950).

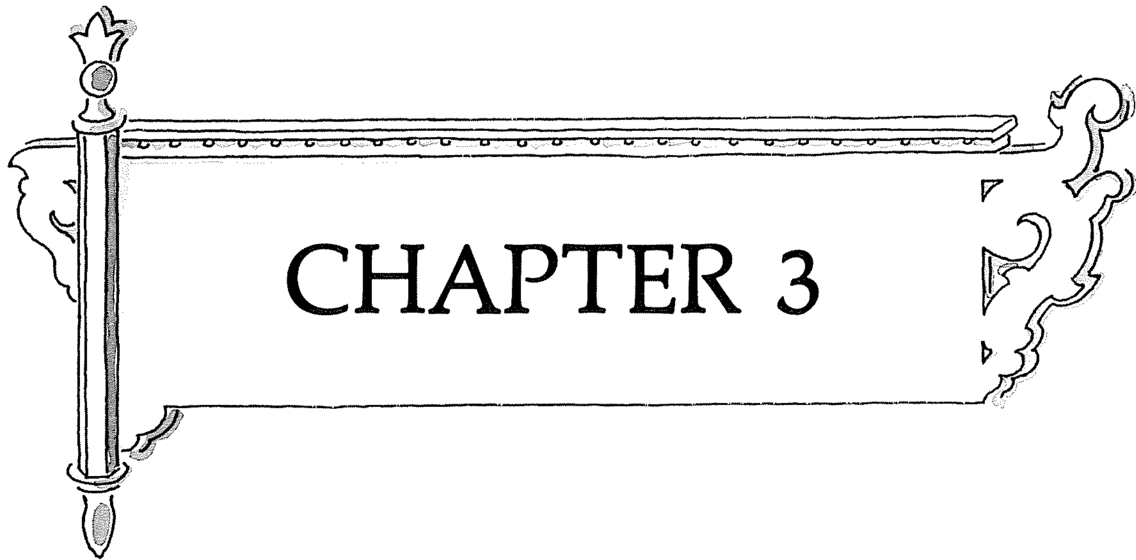
4. McConville, Robert. *The History of Board Games.* (Palo Alto, Calif., Creative Publications, 1974).

5. People's Computer Company. *What To Do After You Hit a Return.* (Menlo Park, Calif.: People's Computer Co., 1975).

6. *Recreational Computing,* People's Computer Co., Box E, 1263 El Camino Real, Menlo Park, CA 94025.

7. Emmerichs, Jack. *SUPERWUMPUS* (BYTE Publications, Peterborough NH 03458; 1978)


TRI-STATE
SINCE 1969
SUPPLIES




# GETTING SERIOUS: HOME FINANCE AND BUSINESS PROGRAMS

## 3.0 INTRODUCTION

The computer applications we're about to examine are often called "serious" because they're connected with the financial complications involved in making a living. They are also connected with tasks most of us would like to see disappear: keeping records, calculating taxes, balancing budgets, and trying to decipher the complicated formulas institutions such as banks and brokers live by.

There has been a fair amount of "oversell" on what a computer can do in these areas. The truth is that introducing a computer into your financial life may turn out to *increase* your work. But it is still worth trying, and for three good reasons:

1. The organizational discipline needed to get your personal or business financial operations in a form that a machine can comprehend will make your future record-keeping much easier, even if you eventually decide to throw out the computer.
2. There will be spin-offs in the form of insights you have not had before: revelations about both good and foolish practices of which you weren't aware simply because you hadn't taken the time to track them down.
3. If you get involved in the program design, and if it actually *works*, you will experience a satisfaction that might just turn the unpleasant chores of record-keeping into fun.

In summary, expect use of business and home finance computer systems to increase your work load at first, even if someone else writes the programs. But once the startup period is behind you, there should be less and less demand on your time for unproductive clerical tasks. There should also be an increase in the control and understanding of your finances.

## 3.1 WHAT'S POSSIBLE? THE WORLD OF DATA PROCESSING

There are three basic operations involved in any computerized business system: the gathering of data; the *processing* of this data; and the producing of output that gives new insights into what the original data means. The complexity of a business computer program therefore depends on how much data there is, and how fancy the user wants to be in processing it for output. A fourth factor that complicates larger business systems is that of security. This involves programming a system of checks and balances to prevent or detect errors and/or tampering with the system. Security can also involve encryption processes that are based on sophisticated mathematics. Except for some simple checks on the validity of input data, and the equality of sums calculated in more than one way, the business programs in this book do not address the question of security.

The following table shows the kind of data the programs in this chapter work on, the processing they perform and the output that they produce. The programs have been chosen to illustrate the most common types of business-related data processing and there are undoubtedly several variations on each of these themes that will occur to you. Also keep in mind that the addition of disk drives and a printer (explained in Chapter 4) will permit further expansion of these ideas.

### SUMMARY OF PROGRAMS AND PROJECTS IN CHAPTER 3

| PROGRAM | INPUT DATA USED | PROCESSING/OUTPUT |
|---|---|---|
| 1. LOAN1 | Amount of money borrowed<br>Annual interest rate<br>Time to pay back (in months) | Uses the formula for add-on interest to calculate and print equalized monthly loan payments |
| 2. LOAN2 | Same as LOAN1 | Uses the formula for interest on declining balance to produce a chart of declining monthly loan payments. |
| 3. SAVE | Balance in savings account<br>Annual interest rate<br>Time left in bank (in days) | Uses formula for compound interest to print a table showing the interest accumulated daily and the total balance. |
| 4. AMORT | Same as LOAN1 | Uses the amortization formula to find equalized monthly payments with interest on declining balance. Also shows por- |

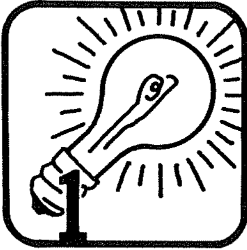
| | | |
|---|---|---|
| | | tion of each payment that is interest, and portion that repays the loan. Calculates and prints totals two ways. |
| 5. *FINANCE* & *FINAN2* | These are user-selected "menu" programs that combine programs 1, 2, 3, and 4 into one package. | |
| 6. ESTIMATE | Uses input statements to get variable information about a building to be painted; uses data statements to get cost of paints and labor. | Calculates areas for each kind of paint coverage, adds in labor costs, and prints total job cost. |
| 7. SALESLIP | Uses a screen-oriented input format to gather sales data at a sales counter; allows for changes or corrections in data. | Uses simple arithmetic and tax formulas to produce an itemized bill; calculates change. |
| 8. *HARDSALE* | Same as SALESLIP. | Produces a "hard copy" (printed on paper) version of the itemized bill. |
| 9. MONEY & *MONEY2* | Uses a screen-oriented input format to gather data on all checks and deposits for a personal financial system. | Produces a statement showing sum of checks written, sum of deposits, and final balance in the account; the program includes an editor and "hooks" to a disk file version. |

*Note:* Further extensions of MONEY and *MONEY2* are given in Chapter 4, along with several programs in the important area of *word processing*. The use of disk-based data files is also discussed there.

## 3.2 SIMPLE FINANCE PROGRAMS: LOAN1, LOAN2, SAVE, AMORT, *FINANCE, FINAN2*

In this section we're going to look at four programs that calculate the interest on either a loan or on savings. They are similar in many ways, but different enough to merit separate treatment. At the end of the section we'll suggest a project that puts all four programs into a single package called *FINANCE.*

### LOAN1



#### The Idea

When you buy something on the installment plan, you are usually told that the brand new gadget you would like can be had for only a small payment of X dollars per month. Somewhere in the fine print you also find out how many months it will take to finish paying off your debt.

To arrive at the monthly payment X, some loan companies first calculate interest on the total cost of the item, then add it to the cost, and divide by the number of months.

Example: Suppose a motorbike costs $3000, and the payments are to be spread over 36 months. If the interest is 9% per year, then,

Interest = .09 * 3000 * 36/12 = $810

Each monthly payment = (3000 +810)/36 = $105.83

This is called calculating payments with add-on interest. It is not a very favorable method for the consumer, since it charges full interest over the entire life of the loan even though part of the original loan is paid back each month. Let's write a program that sheds some light on the amounts involved.

#### Sample Run

The program first of all asks the person to input the amount to be borrowed, often called the principal, (in our example, the cost of the bike,) the number of months to pay it all back, and the interest rate per year. Then it prints out a report giving the amount of the monthly payment, the number of months to pay, the principal, the amount of interest paid per month, the total interest paid, and the total amount paid back.

```
* INSTALLMENT PAYMENTS WITH ADD-ON INTEREST *
AMOUNT BORROWED (PRINCIPAL)? 3000
NUMBER OF MONTHS TO PAY? 36
INTEREST RATE PER YEAR (6.5% = 6.5)? 9

MONTHS          PRINCIPAL         INTEREST        MONTHLY
TO PAY                            PER MO.         PAYMENT
-----------------------------------------------------------
  36         $  3,000.00       $      22.50      $     105.83
                                  TOTAL           TOTAL AMOUNT
                                  INTEREST        PAID BACK
-----------------------------------------------------------
                               $     810.00      $  3,810.00
AGAIN (Y = YES)? NO
```

Notice that a good deal of attention is paid to arranging this output on the screen or paper. A good arrangement can help in reading the information. This is where the PRINT USING statements of extended BASIC come in handy. Also, there was some thought about how the person should type in the interest rate. The decison was that asking a person to type 6% as 6 rather than as .06 would produce fewer errors on input. The program must then multiply R * .01 to get the correct interest rate for actually calculating the interest.

At the end, the program asks if the person wants to use it again. This is nicer than simply ending the program and having the person type RUN to start it again.

## High-Level Design

The program can be designed in terms of four blocks:

1. Get input: principal P, months M, interest rate R.
2. Do calculations: R = R * .01, total interest TI = P * M * R / 12.
3. Print out, in readable form: M, P, TI/M, (P + TI)/M, TI, P + TI.
4. Ask if person wants to use program again; if yes go to 1, otherwise end the program.

## Coding the Program

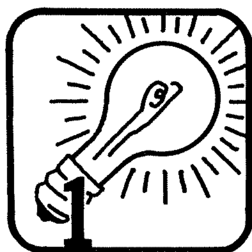
Since we plan later to include this program as part of a larger one, we will use line numbers from 200 to 300.

```
200 '**************************************************
201 '*                    LOAN1                       *
202 '**************************************************
205 CLS: A$=" "
207 PRINT"* INSTALLMENT PAYMENTS WITH ADD-ON INTEREST *"
210 '----------GET INPUT--------------------------------
212 PRINT"AMOUNT BORROWED (PRINCIPAL)";: INPUT P
215 PRINT"NUMBER OF MONTHS TO PAY";: INPUT M
216 IF M <= 0 OR INT(M) <> M THEN 215
220 PRINT"INTEREST RATE PER YEAR (6.5% = 6.5)";: INPUT R
222 '----------CALCULATE--------------------------------
225 R = R * .01
226 TI = P * M * R / 12              'TOTAL INTEREST
227 '----------PRINT REPORT-----------------------------
229 PRINT
230 PRINT"MONTHS", "PRINCIPAL", "INTEREST", "MONTHLY"
240 PRINT"TO PAY",,              "PER MO.",  "PAYMENT"
250 FOR K = 1 TO 60: PRINT"-";: NEXT K: PRINT
255 F1$ ="  ##          $###,###.##     $###,###.##       $###,###.##"
260 PRINT USING F1$; M, P, TI/M, (P+TI)/M
265 PRINT
270 PRINT,,"TOTAL","TOTAL AMOUNT"
271 PRINT,,"INTEREST","PAID BACK"
275 FOR K = 1 TO 60: PRINT"-";: NEXT K: PRINT
279 F2$ ="$###,###.##       $###,###.##"
280 PRINT TAB(29);: PRINT USING F2$; TI, P+TI
285 '----------AGAIN?-----------------------------------
290 PRINT"AGAIN (Y = YES)";: INPUT A$: IF A$ ="Y" THEN 205
```

In line 205, CLS clears the screen of the video display. Making A$ a blank lets you press ENTER to mean "no" in response to the input statement in line 290. On some systems, if you have previously input a 'Y' into A$, later pressing ENTER or RETURN will not change this.

In line 226 the total interest is calculated. Since the interest rate is given for one year but the period of the loan is in months, not years, the interest rate R is divided by 12. The quantity R/12 is often called the monthly interest rate. The remainder of the calculations are pretty straightforward.

## SAVE



### The Idea

Banks are limited by law to maximum interest rates for each type of savings account. However, they can increase the actual interest paid by compounding the interest several times a year, often daily. Since the interest is added to the balance each day and the following day's interest is calculated on this slightly larger balance, a higher actual interest rate

results. The difference is not great, but it is attractive to investors.

The effective annual yield can be made even higher by compounding daily with a rate of R/360 based on a 360-day year, while giving the actual interest for a 365-day year. Using this technique, a 5.25% annual rate produces a yield of 5.47%.

### Sample Run

This program will ask for the person's starting balance, the number of days left in the bank, and the interest rate per year. It will then print a table showing each day's balance and interest earned. Finally it will print the balance and total interest earned.

```
* SAVINGS ACCT. INTEREST, COMPOUNDED DAILY *
STARTING BALANCE? 3000
NUMBER OF DAYS COMPOUNDED? 60
INTEREST RATE PER YEAR (6.5% = 6.5)? 5
--CONTINUE? Y

DAY             BALANCE          INTEREST
-------------------------------------------------------------
 0               3000             0
 1               3000.42          .416667
 2               3000.83          .416725
 3               3001.25          .416782
 4               3001.67          .41684
 5               3002.08          .416898
 6               3002.5           .416956
 7               3002.92          .417014
 8               3003.34          .417072
 9               3003.75          .41713
 10              3004.17          .417188
 11              3004.59          .417246
--CONTINUE? Y

 12              3005             .417304
 13              3005.42          .417362
 14              3005.84          .41742
 15              3006.26          .417478
 16              3006.67          .417536
 17              3007.09          .417594
 18              3007.51          .417652
 19              3007.93          .41771
 20              3008.34          .417768
 21              3008.76          .417826
 22              3009.18          .417884
 23              3009.6           .417942
--CONTINUE? Y

 24              3010.02          .418
 25              3010.43          .418058
 26              3010.85          .418116
 27              3011.27          .418174
 28              3011.69          .418232
 29              3012.11          .41829
```

```
 30              3012.53         .418348
 31              3012.94         .418406
 32              3013.36         .418464
 33              3013.78         .418523
 34              3014.2          .418581
 35              3014.62         .418639
--CONTINUE? Y
 36              3015.04         .418697
 37              3015.46         .418755
 38              3015.87         .418813
 39              3016.29         .418872
 40              3016.71         .41893
 41              3017.13         .418988
 42              3017.55         .419046
 43              3017.97         .419104
 44              3018.39         .419162
 45              3018.81         .419221
 46              3019.23         .419279
 47              3019.65         .419337
--CONTINUE? Y


 48              3020.07         .419395
 49              3020.49         .419454
 50              3020.9          .419512
 51              3021.32         .41957
 52              3021.74         .419628
 53              3022.16         .419687
 54              3022.58         .419745
 55              3023            .419803
 56              3023.42         .419862
 57              3023.84         .41992
 58              3024.26         .419978
 59              3024.68         .420037
--CONTINUE? Y


 60              3025.1          .420095
--CONTINUE? Y

                BALANCE          TOTAL INTEREST
------------------------------------------------------------
              $ 3,025.10       25.102700
AGAIN (Y = YES)? NO
```

This program will be written for a microcomputer with a 16-line video screen display, so the program will pause after each main section of the program and wait for the user to hit any key (such as ENTER) before printing any more. It will also pause after every 11 lines of the table. This leaves room for showing the heading, and the line taken by the word "CONTINUE." Finally, the program asks if the person wants to use it again, as in the previous program.

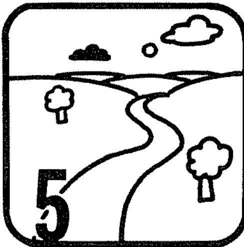
## High-Level Design

1. Get input: Balance P, days D, interest rate R.
2. Do calculations: R = R * .01 / 360; initialize total interest TI = 0. (TI will be accumulated during the creation of the table, below.)
3. Calculate and print table: day number J, current balance P, and interest for each day I1.
4. Print a summary showing the final balance and the total interest.
5. Ask if person wants to use program again; if yes go to 1, otherwise, end the program.

## Coding the Program

```
400 '*******************************************************
401 '*                          SAVE                       *
402 '*******************************************************
405 CLS: A$=" "
410 PRINT"* SAVINGS ACCT. INTEREST, COMPOUNDED DAILY *"
412 '----------GET INPUT---------------------------------
415 PRINT"STARTING BALANCE";: INPUT P
417 PRINT"NUMBER OF DAYS COMPOUNDED";: INPUT D
418 IF D <= 0 OR INT(D) <> D THEN 417
420 PRINT"INTEREST RATE PER YEAR (6.5% = 6.5)";: INPUT R
422 PRINT"--CONTINUE";: INPUT D$
425 '----------CALCULATE & PRINT TABLE--------------------
430 R = R * .01 / 360          'BANKER'S YEAR = 360 DAYS
435 PRINT"DAY", "BALANCE", "INTEREST
437 FOR K = 1 TO 60: PRINT"-";: NEXT K: PRINT
438 PRINT 0, P, 0
439 TI = 0
440 FOR J = 1 TO D
443   IF J/12 = INT(J/12) PRINT"--CONTINUE";: INPUT D$
444   I1 = P * R
445   TI = TI + I1
450   P = P + I1
455   PRINT J, P, I1
460 NEXT J
465 PRINT"--CONTINUE";: INPUT D$
470 '----------SUMMARY--------------------------------
472 PRINT,"BALANCE","TOTAL INTEREST"
474 FOR K = 1 TO 60: PRINT"-";: NEXT K: PRINT
477 X$="              $##,###.##      ###.######"
478 PRINT USING X$; P, TI
480 '----------AGAIN?---------------------------------
485 PRINT"AGAIN (Y = YES)";: INPUT A$: IF A$="Y" THEN 405
```

In line 430, R is changed to the interest for one "banker's day" (1/360 of a year) before beginning the loop that makes the table. TI must be made zero in case you use the program more than once without typing RUN. In line 443, the condition says, "if J divides evenly by 12, that is, if J/12 has no fractional part, wait for input." Its effect is to pause at every 12th J. D$ is a "dummy" variable. Its value is not used, it just allows you to type in anything (including the ENTER key) to make the program continue executing.



### Future Extensions and Revisions

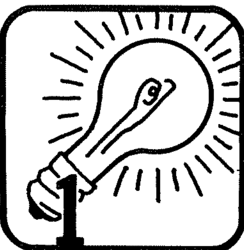
Our program showed the interest for each day. This is a good way to see the effect of compounding daily. However, if you only want to see the final amount, the fix is simple: just remove the PRINT statement (line 455). If you would like to see the interest at a few intermediate times, say, every 30 days, use the same trick employed in line 443. You can do this by keeping the 455 PRINT statement, but preceding it with the statement

```
452 IF J/30 < > INT(J/30) THEN 460
```

The output will be neater if you take out line 443, and add the statement

```
457 PRINT"--CONTINUE";: INPUT D$
```

## LOAN2



### The Idea

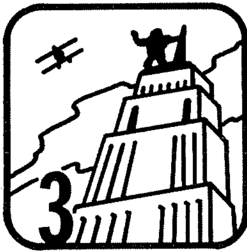
The add-on interest method of paying for "renting" money is usually not used on large loans. Instead, interest is only charged on the unpaid balance of the principal. So if the same interest rate is quoted by two lending institutions, but one uses the first method and the other the second, you will pay much less interest using the second method.

### Sample Run

This program asks for the same input as LOAN1, but, after dividing the principal by the months to get an equal principal payment for each month, it calculates the interest for each month in a loop and prints a table. Since the unpaid balance declines each month, so does the interest payment and the total monthly payment.

```
* INSTALLMENT PAYMENTS WITH INT. ON UNPAID BAL. *
AMOUNT BORROWED (PRINCIPAL)? 3000
NUMBER OF MONTHS TO PAY? 36
INTEREST RATE PER YEAR (6.5% = 6.5)? 9
--CONTINUE? Y

MONTH   PRINCIPAL     INTEREST   +   PRINCIPAL   =   MONTHLY
NUMBER  OWED          PAYMENT        PAYMENT         PAYMENT
-----------------------------------------------------------
  1   $  3,000.00   $    22.50     $    83.33     $  105.83
  2   $  2,916.67   $    21.88     $    83.33     $  105.21
  3   $  2,833.33   $    21.25     $    83.33     $  104.58
  4   $  2,750.00   $    20.63     $    83.33     $  103.96
  5   $  2,666.67   $    20.00     $    83.33     $  103.33
  6   $  2,583.33   $    19.38     $    83.33     $  102.71
  7   $  2,500.00   $    18.75     $    83.33     $  102.08
  8   $  2,416.67   $    18.13     $    83.33     $  101.46
  9   $  2,333.33   $    17.50     $    83.33     $  100.83
 10   $  2,250.00   $    16.88     $    83.33     $  100.21
 11   $  2,166.67   $    16.25     $    83.33     $   99.58
--CONTINUE? Y

 12   $  2,083.33   $    15.63     $    83.33     $   98.96
 13   $  2,000.00   $    15.00     $    83.33     $   98.33
 14   $  1,916.67   $    14.38     $    83.33     $   97.71
 15   $  1,833.33   $    13.75     $    83.33     $   97.08
 16   $  1,750.00   $    13.13     $    83.33     $   96.46
 17   $  1,666.67   $    12.50     $    83.33     $   95.83
 18   $  1,583.33   $    11.88     $    83.33     $   95.21
 19   $  1,500.00   $    11.25     $    83.33     $   94.58
 20   $  1,416.67   $    10.63     $    83.33     $   93.96
 21   $  1,333.33   $    10.00     $    83.33     $   93.33
 22   $  1,250.00   $     9.38     $    83.33     $   92.71
 23   $  1,166.67   $     8.75     $    83.33     $   92.08
--CONTINUE? Y

 24   $  1,083.33   $     8.13     $    83.33     $   91.46
 25   $  1,000.00   $     7.50     $    83.33     $   90.83
 26   $    916.67   $     6.88     $    83.33     $   90.21
 27   $    833.33   $     6.25     $    83.33     $   89.58
 28   $    750.00   $     5.63     $    83.33     $   88.96
 29   $    666.67   $     5.00     $    83.33     $   88.33
 30   $    583.33   $     4.38     $    83.33     $   87.71
 31   $    500.00   $     3.75     $    83.33     $   87.08
 32   $    416.67   $     3.13     $    83.33     $   86.46
 33   $    333.33   $     2.50     $    83.33     $   85.83
 34   $    250.00   $     1.88     $    83.33     $   85.21
 35   $    166.67   $     1.25     $    83.33     $   84.58
--CONTINUE? Y

 36   $     83.33   $     0.63     $    83.33     $   83.96
--CONTINUE? Y

                      TOTAL          TOTAL           TOTAL
                      INTEREST       PRINCIPAL       PAYMENTS
-----------------------------------------------------------
                    $    416.25    $  3,000.00    $  3,416.25
AGAIN (Y = YES)? NO
```

More information is printed out in this table. (Confession: it's made to be similar to the output of the next program, for comparison.) The table shows not only month number, the principal still owed, the interest paid each month, and the total monthly payment, but also the part of the payment which is the payment on the principal. At the end, the summary shows not only total interest and total amount paid back, but the total of the principal payments. These should add up to the amount borrowed within the accuracy of your machine. This is usually about 6 significant figures.



### High-Level Design

1. Get input: principal P, months M, interest rate R.
2. Do calculations: R = R * .01, principal payment P1 = P/M.
3. Calculate and print table: month number J, principal owed P, current month's interest I1, principal payment P1, and the monthly payment P1 + I1.
5. Ask if person wants to use program again; if yes go to 1, otherwise end the program.

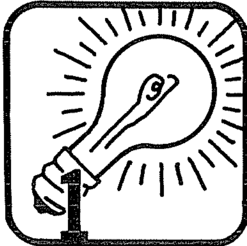
### Coding the Program

This program combines ideas from LOAN1 and SAVE. In fact, about all you must remember is that since the principal payment stays the same it should be calculated outside the loop. And since the program now accumulates three sums inside the loop, TI, SP, and TP, these should all be set equal to zero just before entering the loop.

```
300 '****************************************************
301 '*                        LOAN2                     *
302 '****************************************************
305 CLS: A$=" "
307 PRINT"* INSTALLMENT PAYMENTS WITH INT. ON UNPAID BAL. *"
310 '----------GET INPUT---------------------------------
312 PRINT"AMOUNT BORROWED (PRINCIPAL)";: INPUT P
315 PRINT"NUMBER OF MONTHS TO PAY";: INPUT M
316 IF M <= 0 OR INT(M) <> M THEN 315
317 PRINT"INTEREST RATE PER YEAR (6.5% = 6.5)";: INPUT R
319 PRINT"--CONTINUE";: INPUT D$
320 '----------CALCULATE FIXED QUANTITIES----------------
322 R = R * .01
324 P1 = P/M
325 '----------CALCULATE & PRINT TABLE-------------------
326 TI = 0: TP = 0: SP = 0
327 PRINT"MONTH   PRINCIPAL    INTEREST   +   PRINCIPAL  =   MONTHLY"
329 PRINT"NUMBER  OWED         PAYMENT        PAYMENT        PAYMENT"
330 FOR K = 1 TO 60: PRINT"-";: NEXT K: PRINT
```

```
332 F3$ =" ##   $###,###.##    $#,###.##     $#,###.##    $#,###.##"
340 FOR J = 1 TO M
342   IF J/12 = INT(J/12) PRINT"--CONTINUE";: INPUT D$
345   I1 = P * R / 12
350   PRINT USING F3$; J, P, I1, P1, P1+I1
355   TI = TI + I1
360   TP = TP + P1 + I1
363   SP = SP + P1
365   P = P - P1
370 NEXT J
372 PRINT"--CONTINUE";: INPUT D$
374 '----------SUMMARY-----------------------------------
375 PRINT TAB(21);"TOTAL           TOTAL           TOTAL"
377 PRINT TAB(21);"INTEREST        PRINCIPAL       PAYMENTS"
380 FOR K = 1 TO 60: PRINT"-";: NEXT K: PRINT
382 F4$ ="$##,###.##     $###,###.##     $###,###.##"
383 PRINT TAB(19);:PRINT USING F4$; TI, SP, TP
384 '----------AGAIN?------------------------------------
385 PRINT"AGAIN (Y = YES)";: INPUT A$: IF A$="Y" THEN 305
```

## AMORT



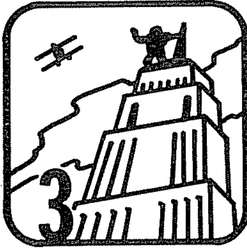
### The Idea

This program will print out the same information as LOAN2, but with a new wrinkle added. Sometimes there is an advantage to having all the payments on a loan be equal, even though interest is being paid on the unpaid balance. This is called an *amortized payment schedule* or an *amortization table.*

A special formula is used to calculate the equal monthly payments. Interest is calculated on the unpaid balance each month, and the principal payment is what is left when this interest payment is subtracted from the monthly payment.

### Sample Run

```
* AMORTIZATION TABLE *
AMOUNT BORROWED (PRINCIPAL)? 3000
NUMBER OF MONTHS TO PAY? 36
INTEREST RATE PER YEAR (6.5% = 6.5)? 9
--CONTINUE? Y

MONTH   PRINCIPAL      INTEREST   +   PRINCIPAL   =   MONTHLY
NUMBER  OWED           PAYMENT        PAYMENT         PAYMENT
----------------------------------------------------------------
  1    $  3,000.00     $    22.50     $    72.90     $    95.40
  2    $  2,927.10     $    21.95     $    73.45     $    95.40
  3    $  2,853.65     $    21.40     $    74.00     $    95.40
  4    $  2,779.65     $    20.85     $    74.55     $    95.40
  5    $  2,705.10     $    20.29     $    75.11     $    95.40
  6    $  2,629.99     $    19.72     $    75.68     $    95.40
  7    $  2,554.31     $    19.16     $    76.24     $    95.40
  8    $  2,478.07     $    18.59     $    76.82     $    95.40
  9    $  2,401.25     $    18.01     $    77.39     $    95.40
 10    $  2,323.86     $    17.43     $    77.97     $    95.40
 11    $  2,245.89     $    16.84     $    78.56     $    95.40
--CONTINUE? Y

 12    $  2,167.33     $    16.25     $    79.15     $    95.40
 13    $  2,088.19     $    15.66     $    79.74     $    95.40
 14    $  2,008.45     $    15.06     $    80.34     $    95.40
 15    $  1,928.11     $    14.46     $    80.94     $    95.40
 16    $  1,847.17     $    13.85     $    81.55     $    95.40
 17    $  1,765.62     $    13.24     $    82.16     $    95.40
 18    $  1,683.46     $    12.63     $    82.77     $    95.40
 19    $  1,600.69     $    12.01     $    83.40     $    95.40
 20    $  1,517.29     $    11.38     $    84.02     $    95.40
 21    $  1,433.27     $    10.75     $    84.65     $    95.40
 22    $  1,348.62     $    10.11     $    85.29     $    95.40
 23    $  1,263.33     $     9.47     $    85.93     $    95.40
--CONTINUE? Y

 24    $  1,177.41     $     8.83     $    86.57     $    95.40
 25    $  1,090.84     $     8.18     $    87.22     $    95.40
 26    $  1,003.62     $     7.53     $    87.87     $    95.40
 27    $    915.74     $     6.87     $    88.53     $    95.40
 28    $    827.21     $     6.20     $    89.20     $    95.40
 29    $    738.01     $     5.54     $    89.87     $    95.40
 30    $    648.15     $     4.86     $    90.54     $    95.40
 31    $    557.61     $     4.18     $    91.22     $    95.40
 32    $    466.39     $     3.50     $    91.90     $    95.40
 33    $    374.49     $     2.81     $    92.59     $    95.40
 34    $    281.89     $     2.11     $    93.29     $    95.40
 35    $    188.61     $     1.41     $    93.99     $    95.40
--CONTINUE? Y

 36    $     94.62     $     0.78     $    94.62     $    95.40
--CONTINUE? Y

                       TOTAL          TOTAL           TOTAL
                       INTEREST       PRINCIPAL       PAYMENTS
----------------------------------------------------------------
                     $    434.43    $  3,000.00     $  3,434.43
AGAIN (Y = YES)? NO
```



### High-Level Design

1. Get input: (same as LOAN2).
2. Do calculations: R = R* .01/12, (For E see below, **Coding the Program.**)
3. Calculate and print table: (same variables as LOAN2, different calculations.)
4. Print summary: TI, SP, TP.
5. Ask if person wants to use program again; if yes go to 1, otherwise end the program.

### Coding the Program

```
500 '*****************************************************
501 '*                    AMORT                          *
502 '*****************************************************
505 CLS: A$=" "
507 PRINT"* AMORTIZATION TABLE *"
510 '----------GET INPUT-------------------------------
515 PRINT"AMOUNT BORROWED (PRINCIPAL)";: INPUT P
517 PRINT"NUMBER OF MONTHS TO PAY";: INPUT M
518 IF M <= 0 OR INT(M) <> M THEN 517
519 PRINT"INTEREST RATE PER YEAR (6.5% = 6.5)";: INPUT R
520 PRINT"--CONTINUE";: INPUT D$
522 '----------CALCULATE FIXED QUANTITIES-----------------
523 R = R * .01 / 12
524 E = (P * R * (1 + R)[M) / ((1 + R)[M-1)
525 '----------CALCULATE & PRINT TABLE--------------------
526 TI = 0: TP = 0: SP = 0
527 PRINT"MONTH   PRINCIPAL    INTEREST   +   PRINCIPAL =    MONTHLY"
529 PRINT"NUMBER  OWED         PAYMENT        PAYMENT        PAYMENT"
530 FOR K = 1 TO 60: PRINT"-";: NEXT K: PRINT
532 F3$=" ##   $###,###.##    $#,###.##      $#,###.##    $#,###.##"
545 FOR J = 1 TO M
550   IF J/12 = INT(J/12) PRINT"--CONTINUE";: INPUT D$
555   I1 = P * R
560   P1 = E - I1
565   IF J = M THEN P1 = P: I1 = E - P1
570   PRINT USING F3$; J, P, I1, P1, E
575   TI = TI + I1
580   TP = TP + P1 + I1
584   SP = SP + P1
587   P = P - P1
590 NEXT J
600 PRINT"--CONTINUE";: INPUT D$
602 '----------SUMMARY-----------------------------------
605 PRINT TAB(21);"TOTAL          TOTAL          TOTAL"
610 PRINT TAB(21);"INTEREST       PRINCIPAL      PAYMENTS"
615 FOR K = 1 TO 60: PRINT"-";: NEXT K: PRINT
620 F4$="$##,###.##     $###,###.##     $###,###.##"
630 PRINT TAB(19);: PRINT USING F4$; TI, SP, TP
635 '----------AGAIN?-------------------------------------
640 PRINT"AGAIN (Y = YES)";: INPUT A$: IF A$="Y" THEN 505
```

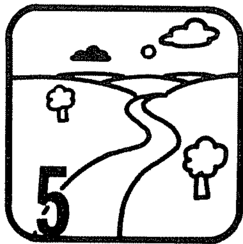
Line 524 contains the formula for finding the equal monthly payment that will take care of both the principal payments and interest on the unpaid balance. The formula uses the "↑" symbol to mean "raise to the power." Mathematicians would write the formula as follows:

$$E = \frac{P * R * (1 + R)^M}{(1 + R)^M - 1} \quad \text{where R is the monthly interest rate.}$$

For example, if R = .09/12 = .0075, P = 3000, and M = 36,

$$E = \frac{3000 * .0075 * 1.0075^{36}}{1.0075^{36} - 1} = \frac{22.5 * 1.30864}{1.30864 - 1}$$

$$= \frac{29.444}{0.3086} = \$95.40$$




**Future Extensions and Revisions**

These short programs are handy to have around. Even more convenient would be a larger program that contained all four shorter programs in a way that made them easy to use.

**Project *FINANCE***

Write a program to accomplish the above improvements. A high-level design for such a larger program would be:

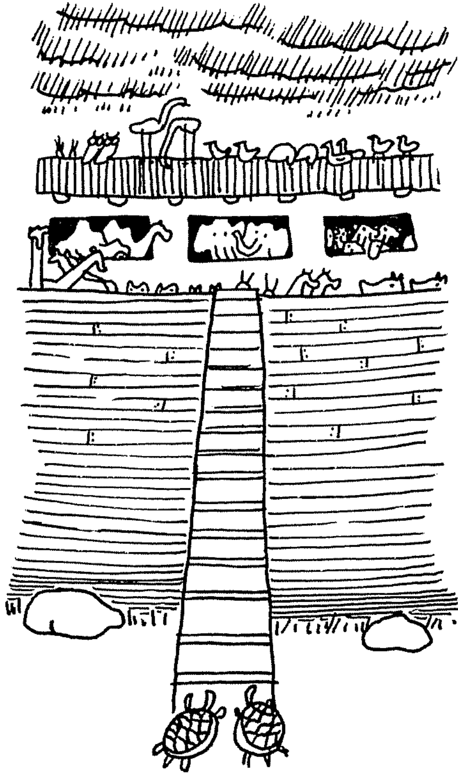
1. Present the user with information on four different programs, and instructions on how to choose one. This is called a *menu*.
2. Obtain the user's input; go to steps 3, 4, 5, 6, or 7.
3. Do LOAN1; when finished go to 1.
4. Do LOAN2; when finished go to 1.
5. Do SAVE; when finished go to 1.
6. Do AMORT; when finished go to 1.
7. End

Use lines 100 to 200 for the menu section. If you have a MERGE command, use it to bring each earlier program into memory and then modify the whole to make FINANCE. If you don't have MERGE, see if there is another way to combine programs on your system; otherwise you will have to re-type each program.

### Project *FINAN2*

Study the listing of your *FINANCE* program. Look for repeated lines or blocks of code which could be made into subroutines. Write a program (modify *FINANCE*) so that some or all repeated code is placed in subroutines. *Note:* You may find that for some short repeated statements it is almost as much trouble to make it into a subroutine as to type the code repeatedly each time it is needed. Keep in mind that the main reason for using subroutines is to organize your thinking. So if making part of a program into a subroutine helps you to design, read, or understand a program, do it. Otherwise just leave the repeated code where it is.

## 3.3 DATA BASED BUSINESS APPLICATIONS



The word *data* is the plural of *datum* which means a piece of information. For example, the age of one person in a group is a datum while information about the ages of all the people in the group is data. Similarly, one temperature measured by a scientist in a lab is a datum, while the set of all temperatures measured in an experiment is data.

Most data comes in the form of what are called "n-tuples." This is a fancy way of saying that data can come singly (1-tuples), in pairs (2-tuples), triples (3-tuples), quadruples (4-tuples), and so on. For example, data on ages might be given in the form of pairs (John, 32), (Sue, 17), and (Rover, 7). Data from a lab might come in the form of triples like pressure, temperature, volume. These n-tuples are also called *data points* because they are sometimes recorded as small marks on graph paper. A collection of many data points is called a *data base.*

What has all this to do with business applications? The answer is "A lot!" Most of the record keeping in both personal and business systems actually amounts to collecting and using large amounts of data. There is also a growing interest in this area on the part of computer scientists, and there is now a specialization called "data base management." The next three programs illustrate some of the simpler ways in which business programs can both create and use data bases. Chapter 4 examines several additonal examples which utilize disks and printers to do even fancier things with data. These include MONYLIST, SNAPSHOT, FILEBOX, and EDIT1000. Longer programs that extend these ideas are introduced as projects, with complete solutions given in Chapter 5.

## 3.4 SMALL DATA BASED PROGRAMS. ESTIMATE, *(CATERER)*

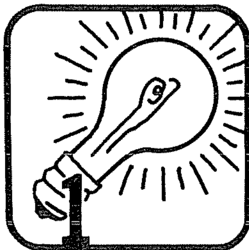
Our first data based program will use only simple BASIC, but it will illustrate the distinction between *transaction* data and *master file* data. This distinction applies to most data based applications, and it will reappear in all the remaining business programs.

### Program ESTIMATE

#### The Idea

Suppose you have a part-time job painting house interiors. People have heard about your work, and the phone calls are starting to come in: "I'd like my house painted. What do you charge?" Instead of just guessing, you decide to write a program that will provide an immediate and fairly accurate answer for each inquiry.

This is a good program to study even if you don't paint houses. That's because the basic idea applies to many other businesses—to any enterprise where a customized estimate is needed for each job. No matter what the specific application, the same three basic operations are involved:



1. Special data unique to the customer is first gathered. In our painting example we input this data as the number of rooms, their dimensions, and the kind of paint desired. Data that is entered at the time a specific customer is handled is called *transaction data.*
2. Additional "older" data is then read from a *master file.* In our example the master file will consist of data statements that store the price per gallon for each kind of paint, the number of square feet covered by a gallon, and some conversion factors that estimate the cost of labor. This data may have to be updated occasionally, but probably not more than once a month.
3. All of this information is then put together with formulas that calculate the total cost of labor, materials, fixed expenses, and applicable tax. (If the total is too high you could then run a loan program from FINANCE to give the customer figures on financing the job!)

#### Sample Run

Here's what the ESTIMATE program might look like during a run. You should imagine that this program is being run on the painter's desk-top computer while he is talking to a client on the telephone. This way instant estimates can be provided, and quotes for several alternate schemes given by simply rerunning the program.

```
   <*>   INTERIOR PAINTING ESTIMATE   <*>
---------------------------------------------
HOW MANY ROOMS TO BE PAINTED (1 TO 10)? 3

NAME OF ROOM # 1 ? LIVING ROOM
L, W, H OF LIVING ROOM IN FT. (E.G. 30,20,8)? 25,20,10
TYPE OF PAINT FOR LIVING ROOM (1, 2, OR 3)? 2

NAME OF ROOM # 2 ? BEDROOM
L, W, H OF BEDROOM IN FT. (E.G. 30,20,8)? 20,15,8
TYPE OF PAINT FOR BEDROOM (1, 2, OR 3)? 1

NAME OF ROOM # 3 ? KITCHEN
L, W, H OF KITCHEN IN FT. (E.G. 30,20,8)? 15,15,8
TYPE OF PAINT FOR KITCHEN (1, 2, OR 3)? 3

ROOM         AREA      GALS       $ PAINT    HOURS      $ LABOR
==============================================================
LIVING ROOM  1400      3.11111    36.5556    18.6667    140
BEDROOM      860       2.15       16.1035    8.6        64.5
KITCHEN      705       1.28182    20.3809    8.97273    67.2955
PRESS 'ENTER' FOR TOTALS -- READY?



*** TOTAL ESTIMATE FOR ALL 3 ROOMS ***
TOTAL AREA (SQ. FT.) = 2965
TOTAL HOURS OF LABOR = 36.2394
TOTAL COST OF PAINT = $  73.04
TOTAL COST OF LABOR = $ 271.80

SUB-TOTAL PAINT AND LABOR = $ 344.84
PLUS 10% TRAVEL & SETUP   = $  34.48
PLUS  6% LOCAL TAX        = $  20.69
.........................................
ESTIMATE FOR TOTAL JOB    = $ 400.01
```

## The High-Level Design

In addition to specifying the three operations mentioned earlier (input the transaction data, read the master file data, and calculate the final estimate) our high-level design will want to include a new feature. It should specify the organization of the data by showing what is often called the *data layout*.

For the ESTIMATE program, we'll specify the data layout as follows:

1. Transaction Data: This will be entered during a run via 3 input statements.
   N = number of rooms to be painted
   L(I), W(I), H(I) = dimensions for room #I in feet
   T(I) = Type of paint for room #I (entered as
   1, 2, or 3 for flat latex, semi-gloss, or special).

2. Master Data File: This will be read during a run from data statements. There will be three data statements in our file, each containing the quadruple; paint type, dollars per gallon, square feet covered per gallon, hours of labor per gallon.

```
1001 DATA 1, 7.49, 400, 4
1002 DATA 2, 11.75, 450, 7
1003 DATA 3, 15.90, 600, 12
```

For example, statement 1001 contains data that says paint of type 1 (latex) costs 7.49 per gallon, it covers 400 square feet per gallon, and it takes about 4 hours per gallon to apply.

Using data statements like this implies that the data must be organized and used *sequentially.* A run of a BASIC program that uses sequential data always begins reading at the first item, and then goes through all the data in the order shown. It is as though the data were recorded on a tape which is always rewound to the start for each program run. As a matter of fact, this data could be stored on a data tape instead of in data statements. The only change in the final program would be to change the READ statement to something like INPUT #−1 (which on the TRS-80 means read from tape machine #1; for the exact tape statement on any other machine you will have to check its manual).

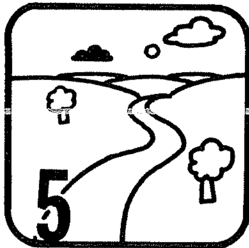
The reason we emphasize that this data is stored sequentially is to alert you to the fact that your program may have to read through some unwanted data in order to arrive at the information actually desired. One technique for doing this was explained in section 2.1 under project DATARIT4. Another approach is to avoid the problem by first reading the master data into arrays. We'll use this approach in the block beginning at line 300. Since arrays can be accessed *randomly,* we can then use the master data any way we want. Here's a high-level design based on this approach.

```
100 '  INITIALIZE VARIABLES: PRINT HEADING
200 '  INPUT TRANSACTION DATA INTO ARRAYS
300 '  READ MASTER FILE DATA INTO ARRAYS
400 '  CALCULATE MATERIAL AND LABOR COSTS FROM
       DATA
500 '  PRINT SUMMARY OF ESTIMATED COSTS
1000 ' PUT MASTER FILE DATA STATEMENTS HERE
```

**Coding the Program**

```
10 ' ***************************************************
20 ' *     ESTIMATE  (ESTIMATES COST OF PAINTING)      *
30 ' ***************************************************
110 CLEAR 300: CLS
120 ' DEFAULT DIM FOR ALL ARRAYS IS 10
130 PRINT "   <*>   INTERIOR PAINTING ESTIMATE   <*>"
140 PRINT STRING$(44,"-")
200 '-------------------------------------------------
201 '     INPUT TRANSACTION DATA FOR EACH ROOM
202 '-------------------------------------------------
210 PRINT"HOW MANY ROOMS TO BE PAINTED (1 TO 10)";:INPUT R
211 IF INT(R)<>R OR R<1 OR R>10 THEN 210
215 PRINT
220 FOR J=1 TO R
230   PRINT"NAME OF ROOM #";J;:INPUT N$(J)
240   PRINT"L, W, H OF ";N$(J);" IN FT. (E.G. 30,20,8)";
245   INPUT L(J),W(J),H(J)
250   PRINT"TYPE OF PAINT FOR ";N$(J);" (1, 2, OR 3)";
252   INPUT P(J)
255   IF INT(P(J))<>P(J) OR P(J)<1 OR P(J)>3 THEN 250
260   PRINT
270 NEXT J
300 '-------------------------------------------------
301 '     READ MASTER FILE DATA
302 '-------------------------------------------------
310 FOR K=1 TO 3
320   READ D, C(K), A(K), T(K)
330 NEXT K
400 '-------------------------------------------------
401 '     CALCULATE AND PRINT COSTS FOR EACH ROOM
402 '-------------------------------------------------
410 CLS:PRINT"ROOM";TAB(12);"AREA";TAB(22);"GALS";
411 PRINT TAB(32);"$ PAINT";TAB(43);"HOURS";TAB(54);"$ LABOR"
415 PRINT STRING$(60,"=")
420 W=7.5      'CURRENT HR. WAGE
425 FOR J=1 TO R
430   RA(J)=2*H(J)*(L(J)+W(J))+L(J)*W(J)
435   TA=TA+RA(J)
440   RG(J)=RA(J)/A(P(J))
445   RM(J)=RG(J)*C(P(J))
450   TM=TM+RM(J)
455   RH(J)=RG(J)*T(P(J))
460   TH=TH+RH(J)
465   RL(J)=RH(J)*W
470   TL=TL+RL(J)
475   PRINT LEFT$(N$(J),11);TAB(12);RA(J);TAB(22);RG(J);
476   PRINT TAB(32);RM(J);TAB(43);RH(J);TAB(54);RL(J)
480 NEXT J
485 PRINT "PRESS 'ENTER' FOR TOTALS -- READY";:INPUT D$
500 '-------------------------------------------------
501 '     PRINT SUMMARY OF TOTAL ESTIMATED COSTS
502 '-------------------------------------------------
510 PRINT:PRINT"*** TOTAL ESTIMATE FOR ALL";R;"ROOMS ***"
520 PRINT "TOTAL AREA (SQ. FT.) =";TA
530 PRINT "TOTAL HOURS OF LABOR =";TH
540 PRINT USING "TOTAL COST OF PAINT = $####.##";TM
550 PRINT USING "TOTAL COST OF LABOR = $####.##";TL
555 T=TL+TM: PRINT
560 PRINT USING "SUB-TOTAL PAINT AND LABOR = $####.##";T
565 PRINT USING "PLUS 10% TRAVEL & SETUP   = $####.##";.1*T
570 PRINT USING "PLUS  6% LOCAL TAX        = $####.##";.06*T
575 PRINT STRING$(50,".")
580 PRINT USING "ESTIMATE FOR TOTAL JOB    = $####.##";T*1.16
```

```
999 '
1000 '========== MASTER DATA STATEMENTS ===========
1001 DATA 1,7.49,400,4
1002 DATA 2,11.75,450,6
1003 DATA 3,15.90,550,7
9999 END
```



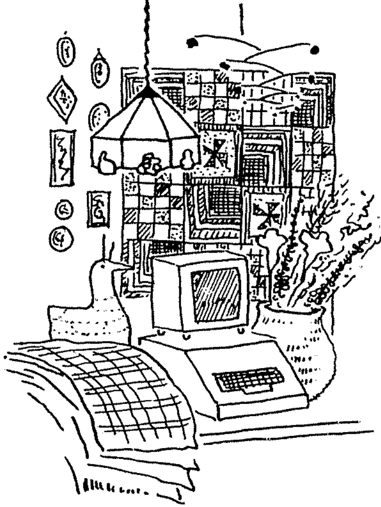

**Future Extensions and Revisions**

The principal extension to an estimating program is to revise the model (formulas) used, on the basis of experience. There are many factors that go into making a good estimate, and your first program may miss some of them. An experienced person should continually evaluate the results of the computer program to see if and what changes ought to be made. This is a good example of why there is no substitute for human judgment; the computer should always be viewed as a tool for supporting that judgment, not replacing it.

**Superproject** (*CATERER*)

Write a program that allows a caterer to quickly give estimates for serving various kinds of meals. Allow the customer to specify such things as choice and quantity of several entrees, with or without the customer's own dishes, buffet or sit down, and any other special features you deem desirable. It would be a good idea to research this by first talking to a few caterers. Do not be surprised if you find that some of them haven't developed definite procedures for calculating estimates. In situations like this, the programmer's most difficult task is to *define* the problem before attempting to design a solution. If you ever start working as a computer consultant, defining your client's problem will become as time consuming as solving it.

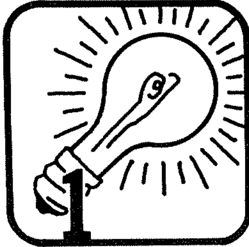
## 3.5 SCREEN-ORIENTED TRANSACTIONS. SALESLIP, *HARDSALE*



Being able to put transaction data into a computer quickly makes all the difference in certain business applications. The BASIC input statement is a big help here. It can be made even more useful by combining it with the TAB function, and other special features that place the prompting symbol (?) in more natural places.

The next program shows how this works for a computer "sales slip" program that is a much more powerful business tool than it may seem at first. This is because it gathers transaction data at the time of a sale which can later be used for all sorts of other things. For example, this same data can be used to produce invoices, inventory lists, profit reports, tax records, and mailing lists.

### Program SALESLIP



### The Idea

Let's imagine a small retail store (say, a shop that sells hand made products produced by local artists) where all sales are recorded on a sales slip. So, at the time of a sale, a clerk records such things as name and address of the customer, quantity and description of each article bought, and unit price. The sales clerk also has to do the arithmetic needed to calculate total price, tax, total amount due, and the change due to the customer after payment is made.

Our idea is to place a small computer with a video screen at the sales counter and do all of the above electronically. So we will want the printing on the screen to prompt the sales clerk to enter the needed information. There should also be provisions for making changes or correcting typing mistakes. Thus a simple *editor* should be built into the program. Finally, there should be "hooks" to the future. By this we mean that the program should have provision for such things as printing invoices on paper (in case the owner buys a printer), or saving charge account information on a magnetic disk file (in case the owner later decides to add disk drives of the type described in Chapter 4).

### Sample Run

The first part of the run is shown in photograph A. The clerk types in the data shown after each question mark. Everything else is printed by the program.

```
SALES SLIP # 4857          DATE: OCT. 23 , 1982
.................................................
CUSTOMER NAME? JOHN G. SMITH
STREET ADDRESS? 2385 E. MAIN ST.
ZIP CODE? MIDVILLE 18324
TYPE      QTY   CAT#     DESCRIPTION  UNIT PRICE    (0 = EXIT)
ITEM 1 :? 2   ? AC-48  ? WHITE RUG   ? 18.95
ITEM 2 :? 12  ? TH-03  ? GOBLET      ? 6.25
ITEM 3 :? 0
TYPE F IF FINISHED, C TO CHANGE, R TO RESUME, V TO VOID? C
WHICH ITEM # TO BE CHANGED? 1
ITEM 1 :? 2   ? AC-49  ? BROWN RUG   ? 18.95
TYPE F IF FINISHED, C TO CHANGE, R TO RESUME, V TO VOID? F_
```

*When SALESLIP is first run, the clerk enters sales data as shown. To stop entering data, a zero is entered. To change an entry, the command C is used. When finished, the command F is entered, producing the final sales slip as shown in the next photograph.*

The second part of the run is shown in the photograph on the next page. This output appears right after the clerk types F (for finished) at the end of the first part of the run as shown above.

```
              <*> MADE BY HAND SHOPS <*>

INVOICE # 4857                        DATE: OCT. 23 , 1982
SOLD TO: JOHN G. SMITH
         2385 E. MAIN ST.
         CITY MIDVILLE 18324

          QTY   CAT#     DESCRIPTION     PRICE      TOTAL
**********************************************************
ITEM  1:   2    AC-49    BROWN RUG       18.95      37.90
ITEM  2:  12    TH-03    GOBLET           6.25      75.00

                                      SUM =  112.90
                                      TAX =    6.77
TYPE P TO PRINT, V TO VOID? P_      TOTAL =  119.67
```

*This is the electronic sales slip that is produced when F is typed. All the arithmetic is done, giving the final sum owed. The formula for sales tax will have to be modified for different localities.*

### High-Level Design

We will organize our program around seven blocks, with two of them (the printer routine at line 4000 and the charge account section at line 500) reserved for future use.

100′ INITIALIZE DATA AND SALES SLIP #
200′ INPUT SALES TRANSACTION DATA, USING SUBROUTINE 2000 TO DO FANCY "CURSOR CONTROL" OF THE INPUT PROMPT SYMBOL (?)
300′ ALLOW CORRECTIONS TO BE MADE OR SALES SLIP TO BE VOIDED
400′ PRINT A NICELY FORMATTED SALES SLIP ON THE SCREEN WITH ALL THE CALCULATIONS COMPLETED
500′ FUTURE CHARGE ACCOUNT SECTION
2000′ SPECIAL SUBROUTINE FOR MANIPULATING THE INPUT ? PROMPT SYMBOL (THIS IS ALSO CALLED "CURSOR CONTROL")
4000′ FUTURE SUBROUTINE FOR PRINTING THE SALES SLIP ON PAPER

### Coding the Program

The technique used by this program to place the INPUT question mark at several places on the same line (see photograph A) is shown in lines 2010 to 2040. In line 2010, the middle of the statement says INPUT Q(N). For N = 1, this produces one ? as follows:

```
ITEM 1:?
```

Now if line 2020 said

```
PRINT TAB(14);:INPUT C$(N)
```

there would be a second ? placed 14 spaces to the right. But this would be on a new line so we'd have

```
ITEM 1:?
              ?
```

Fortunately, on the TRS-80, there is a special control character that says go back up one line. Its ASCII code is 27. So you can use the CHR$ function in the following way:

```
2010 PRINT "ITEM";N;":";:INPUT Q(N)
2020 PRINT TAB(14); CHR$(27);:INPUT C$(N)
```

This produces, for N = 1,

```
ITEM 1:? 12     ? AJ-42
```

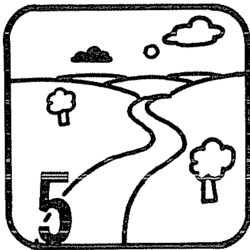
where the 12 is typed by the clerk (and stored in Q(1)) as the quantity bought of item 1, and AJ-42 is typed by the clerk (and stored in C$(1)) as the catalog number of item 1.

Here's a table of the variables and arrays used in SALESLIP, followed by a listing of the program.

| Variable | Meaning |
|---|---|
| M, D, Y | Month, day, year of sale. |
| M$ | Month name. |
| S | Sales slip number |
| Q( ) | Array to hold quantities of items sold. |
| C$( ) | Array to hold catalog (or stock) numbers. |
| D$( ) | Array to hold description of items sold. |
| P( ) | Array to hold prices of items sold. |
| N$ | Customer name. |
| S$ | Customer street address. |
| Z$ | Customer city and zip code. |
| N | Current item number being processed. |
| L | Largest item number processed. |
| C | Item number to be changed. |
| A$ | Answer to program branch question. |
| E | Extended price for each item. |
| T | Total cost of all items. |
| X | Sales tax at 6%. |

```
10 '*****************************************************
20 '*   SALESLIP  (COLLECTS TRANSACTIONS & PRINTS BILL)   *
30 '*****************************************************
40 CLS: CLEAR 500
100 '----------------------------------------------------
101 '                    INITIALIZATION
102 '----------------------------------------------------
110 DATA"JAN.","FEB.","MAR.","APRIL","MAY","JUNE","JULY","AUG."
120 DATA "SEPT.","OCT.","NOV.","DEC."
125 PRINT "DATE (mm,DD,YY)";: INPUT M,D,Y
130 FOR K=1 TO M: READ M$: NEXT K
140 PRINT "TYPE LAST SALES SLIP # USED";: INPUT S: S=S+1
160 DIM Q(25),C$(25),D$(25),P(25)
200 '----------------------------------------------------
201 '                    SALES DATA ENTRY
202 '----------------------------------------------------
210 CLS: PRINT "SALES SLIP #";S; TAB(27);"DATE: ";M$;" ";D;",";1900 + Y
220 PRINT "..............................................."
225 L=0
230 PRINT "CUSTOMER NAME";:INPUT N$
240 PRINT "STREET ADDRESS";:INPUT S$
250 PRINT "ZIP CODE";:INPUT Z$
255 PRINT "TYPE     QTY   CAT#     DESCRIPTION  UNIT PRICE   (0 = EXIT)
260 L=L+1
280 N=L : GOSUB 2000
290 GOTO 260
300 '----------------------------------------------------
301 '                  CORRECTION ROUTINE
302 '----------------------------------------------------
310 PRINT "TYPE F IF FINISHED, C TO CHANGE, R TO RESUME, V TO VOID";
315 INPUT A$
320 IF A$="F" THEN 400
330 IF A$="C" THEN 360
340 IF A$="V" THEN 210
345 IF A$="R" THEN 280
350 GOTO 310
360 PRINT "WHICH ITEM # TO BE CHANGED";:INPUT C
365 IF C > L THEN 360
370 N=C: GOSUB 2000
375 IF C=L THEN L=L+1
380 GOTO 310
400 '----------------------------------------------------
401 '                  VIDEO SALES INVOICE
402 '----------------------------------------------------
410 CLS: PRINT TAB(15); "<*> MADE BY HAND SHOPS <*>"
415 PRINT: PRINT "INVOICE #";S, TAB(39); "DATE:  ";M$;" ";D;",";1900+Y
420 PRINT "SOLD TO: "; N$
425 PRINT TAB(9);S$
430 PRINT TAB(9);"CITY ";Z$
440 PRINT
445 PRINT "         QTY    CAT#      DESCRIPTION    PRICE      TOTAL"
450 PRINT "........................................................."
455 T=0
460 FOR K=1 TO L-1
465 E=P(K)*Q(K)
468 F$="ITEM ##:  ###    %      %  %              %  ###.##     #####.##"
470 PRINT USING F$;K,Q(K),C$(K),D$(K),P(K),E
475 T=T+E
480 NEXT K
485 PRINT
486 X=.06*T
487 PRINT TAB(47);:PRINT USING "  SUM = ####.##";T
```

```
488 PRINT TAB(47);:PRINT USING "  TAX = ####.##";X
489 PRINT TAB(47);:PRINT USING "TOTAL = ####.##";T+X
490 PRINT CHR$(27);"TYPE P TO PRINT, V TO VOID";:A$="":INPUT A$
494 IF A$="V" THEN 210
495 IF A$="P" THEN 497
496 PRINT "??? PLEASE RETYPE";:INPUT A$: GOTO 494
497 PRINT "IF CASH, TYPE PAYMENT; IF CREDIT TYPE 0";:INPUT A
498 IF A>0 PRINT USING "CHANGE IS ##.##";A-T-X: GOTO 4010
500 '--------------------------------------------------
501 '          CHARGE ACCOUNT SECTION
502 '--------------------------------------------------
510 PRINT "CHARGE ACCT. SECTION NOT WRITTEN YET *** RECORD BY HAND"
590 PRINT "PRESS 'ENTER' TO CONTINUE -- READY";:INPUT A$
595 GOTO 4010
2000 '-------------------------------------------------
2001 '         CURSOR-CONTROLLED INPUT SUBROUTINE
2002 '-------------------------------------------------
2010 PRINT "ITEM";N;":";:INPUT Q(N):IF Q(N)=0 THEN 300
2020 PRINT TAB(14);CHR$(27);:INPUT C$(N):IF C$(N)="0" THEN 300
2030 PRINT TAB(23);CHR$(27);:INPUT D$(N):IF D$(N)="0" THEN 300
2040 PRINT TAB(37);CHR$(27);:INPUT P(N):IF P(N)=0 THEN 300
2050 RETURN
4000 '-------------------------------------------------
4001 '         LINE PRINTER SECTION
4002 '--------------------------------------------------
4010 PRINT "TURN PRINTER ON -- READY";:INPUT A$
4020 PRINT "LINE PRINTER SECTION NOT WRITTEN YET"
4090 PRINT "RESUME(R), PRINT AGAIN(P), OR QUIT(Q)";:A$="":INPUT A$
4092 IF A$="R" THEN S=S+1: GOTO210
4094 IF A$="P" THEN 4010
4096 IF A$="Q" THEN 9999
4099 GOTO 4090
9999 END
```



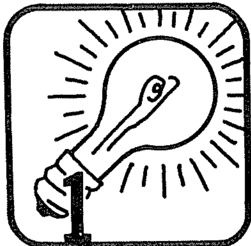

## Future Extensions

The most obvious extension to this program is to print the final sales slips on paper to make a permanent file of these transactions and give a copy to the customer. Other extensions would depend on the use of disk storage, and as mentioned earlier, there are all kinds of possibilities ranging from inventory control to maintaining an accounts receivable file.

## Project *HARDSALE*

Finish subroutine 4000 for use with a hard copy printer. This is a computer-controlled printing device that puts its output on paper. For the TRS-80 machine, you can do this by imitating block 400, but using LPRINT instead of PRINT. (The keyword LPRINT comes from "line printer." This is a high-speed mechanism that puts out a whole line at a time with synchronized impacts of hammers that hit the paper against a rapidly moving belt of metal typefaces. However the printers typically used on small computers print one character at a time from left to right like typewriters. It is more accurate to call them "high-speed sequential character printers," but computer vendors are not always that careful with their terminology.)

## 3.6 DATA BASED FINANCIAL REPORTS. MONEY, *MONEY2, (CHECKBAL)*



In this section we'll concentrate on a home finance program that will seem to be a fancy check balancing system. But as you will soon find out, there's more to it than meets the eye.

### Program MONEY

### The Idea

The simplest kind of personal accounting system is a *single entry* checking account. The main accounting task required is to reconcile the balance (checks or cash deposited minus the checks written) with the bank's monthly statement. Let's use a computer to keep track of these two kinds of transactions: checks written to others and deposits to your account. While we're at it, we will also record cash receipts and payments. By giving each of these transactions a code, we can relate them to various income tax forms. That way it should be possible to automate much of the reporting that is required at tax time.

This suggests that there should eventually be two major parts to our program: the *transaction* (data gathering) part, and the *report generating* part. Each of these parts will, in turn, have several sub-parts. The program will allow us to choose which part and which sub-part we want by using what is called a selection *menu.*

### A Sample Run

In the run that follows, you will see that some of the sub-parts are missing. They will be added later. To follow the run, imagine that the user is a part-time consultant who writes software for clients. It is a small operation, so he or she keeps pertinent records in four file folders (or boxes) corresponding to the four quarters of the year. We will call these the *reporting periods,* with the idea that the computer will be used once each period to gather data and generate reports. During a period, written records, ie: cancelled checks, deposit slips, sales slips for cash purchases, bank statements, credit card bills, etc. are filed in their appropriate folder.

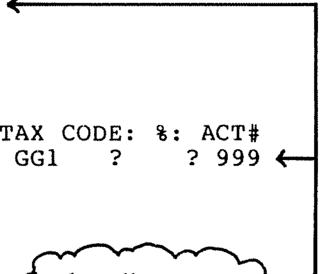
Of course, you may wish to use a different number of periods, say 12, depending on how timely you want your reports to be. The program allows this, but it's up to you to start each period by telling the program what the balance was at the end of the previous period. This is necessary because we will not use tapes or disks for saving data; that is coming later. Here is our sample run:

```
         <*>   FINANCIAL TAX/REPORTING SYSTEM   <*>
------------------------------------------------------------
FOR CALENDAR YEAR 19? 84
FOR PERIOD #? 2
STARTING BALANCE THIS PERIOD (E.G. 589.65)? 350.89
CK. ACCT. OR BANK NAME? UNION NATIONAL
STARTING CK. # THIS PERIOD? 123

                         STAND BY


              <<<    MAIN MENU SELECTION   >>>
------------------------------------------------------------
TRANSACTION (1), REPORT (2), OR EXIT (3)? 1


          <   TRANSACTION MENU SELECTION   >
------------------------------------------------------------
     1 = CK PAYMENTS BY YOU
     2 = CASH PAYMENTS BY YOU
     3 = CK RECEIPTS DEPOSITED BY YOU
     4 = CASH RECEIPTS
     5 = EXIT FROM TRANSACTION SECTION

1, 2, 3, 4, OR 5? 1


***  CURRENT CK BALANCE THIS PERIOD =$  350.89
TO MAKE A CHANGE  OR EXIT TYPE 0,0 FOR MO,DAY
TO RESTART A LINE TYPE -1 IN ANY COLUMN
------------------------------------------------------------
CK #  MO,DAY: AMT:     VENDOR:       FOR:       TAX CODE: %: ACT#
 123 ? 10,22? 42.59  ? AJAX BOOKS? COMPUTER B? CC26   ? 100? 680
 124 ? 10,30? 189.60 ? MUTUAL CO.? INSURANCE ? AA5    ? 0   ? 740
 125 ? 11,4 ? 250    ? BROKEN ARM? RENT       ? CC16   ? 20  ? 320
 126 ? 0,0
EXIT(E), CHANGE(C), RESUME(R), OR LIST(L)? E

  < EXIT FROM TRANSACTION SECTION >
>>> CURRENT BALANCE =-131.3
*** WARNING: CURRENT CHECKING BAL. IS NEGATIVE ***
SAVE(S), RESUME(R), OR EXIT(E)? E
```

*This was typed as "Broken Arms APT" but overwritten by the next item. However it is all in memory for later printing.*

```
            <   TRANSACTION MENU SELECTION    >
-----------------------------------------------------------
     1 = CK PAYMENTS BY YOU
     2 = CASH PAYMENTS BY YOU
     3 = CK RECEIPTS DEPOSITED BY YOU
     4 = CASH RECEIPTS
     5 = EXIT FROM TRANSACTION SECTION

1, 2, 3, 4, OR 5? 2




-----------------------------------------------------------
PAY#  MO,DAY: AMT:       VENDOR:      FOR:        TAX CODE: %: ACT#
 1    ? 9,14 ? 12      ? YELLOW CAB? TAXI       ? EB3    ? 100? 210
 2    ? 11,23? 37.50   ? LA FONDUE ? BUS. MEAL ? CC21   ? 50 ? 720
 3    ? 0,0
EXIT(E), CHANGE(C), RESUME(R), OR LIST(L)? L
LIST OF PAY TRANSACTIONS NOW IN G ARRAY
  1  9/14 $   12.00  YELLOW CAB     TAXI            EB3   100 210
  2 11/23 $   37.50  LA FONDUE      BUS. MEAL       CC21   50 720
EXIT(E), CHANGE(C), RESUME(R), OR LIST(L)? E

   < EXIT FROM TRANSACTION SECTION >
>>> CURRENT BALANCE =-131.3
*** WARNING: CURRENT CHECKING BAL. IS NEGATIVE ***
SAVE(S), RESUME(R), OR EXIT(E)? E


            <   TRANSACTION MENU SELECTION    >
-----------------------------------------------------------
     1 = CK PAYMENTS BY YOU
     2 = CASH PAYMENTS BY YOU
     3 = CK RECEIPTS DEPOSITED BY YOU
     4 = CASH RECEIPTS
     5 = EXIT FROM TRANSACTION SECTION

1, 2, 3, 4, OR 5? 3




-----------------------------------------------------------
DEP#  MO,DAY: AMT:      VENDOR:      FOR:        TAX CODE: %: ACT#
 1    ? 8,4  ? 821    ? GLOBAL INT? -1
 1    ? 8,4  ? 812    ? GLOBAL INT? AUG. PAY  ? AA1    ?     ? 410
 2    ? 11,23? 150    ? BYTE       ? ARTICLE   ? CC4    ?     ? 430
 3    ? 0,0
EXIT(E), CHANGE(C), RESUME(R), OR LIST(L)? L
LIST OF DEP TRANSACTIONS NOW IN G ARRAY
  1  8/ 4 $ 812.00  GLOBAL INTL.   AUG. PAY        AA1    100 410
  2 11/23 $ 150.00  BYTE           ARTICLE         CC4    100 430
EXIT(E), CHANGE(C), RESUME(R), OR LIST(L)? E
```

```
  < EXIT FROM TRANSACTION SECTION >
>>> CURRENT BALANCE = 830.7
SAVE(S), RESUME(R), OR EXIT(E)? E



          <   TRANSACTION MENU SELECTION   >
------------------------------------------------------------
     1 = CK PAYMENTS BY YOU
     2 = CASH PAYMENTS BY YOU
     3 = CK RECEIPTS DEPOSITED BY YOU
     4 = CASH RECEIPTS
     5 = EXIT FROM TRANSACTION SECTION

1, 2, 3, 4, OR 5? 4



CSH#  MO,DAY: AMT:     VENDOR:     FOR:        TAX CODE: %: ACT#
 1    ? 11,14? 20     ? UNKLE BEN ? GYFT      ? GG1    ?    ? 506
 2    ? 0,0
EXIT(E), CHANGE(C), RESUME(R), OR LIST(L)? C
CHANGE DATA FOR WHICH #? 3
CSH# # TOO HIGH
EXIT(E), CHANGE(C), RESUME(R), OR LIST(L)? C
CHANGE DATA FOR WHICH #? 1
CSH#  MO,DAY: AMT:     VENDOR:     FOR:        TAX CODE: %: ACT#
 1    ? 11,14? 20     ? UNCLE BEN ? GIFT      ? GG1    ?    ? 999
EXIT(E), CHANGE(C), RESUME(R), OR LIST(L)? E

  < EXIT FROM TRANSACTION SECTION >
>>> CURRENT BALANCE = 830.7
SAVE(S), RESUME(R), OR EXIT(E)? E
```




```
          <   TRANSACTION MENU SELECTION   >
------------------------------------------------------------
     1 = CK PAYMENTS BY YOU
     2 = CASH PAYMENTS BY YOU
     3 = CK RECEIPTS DEPOSITED BY YOU
     4 = CASH RECEIPTS
     5 = EXIT FROM TRANSACTION SECTION

1, 2, 3, 4, OR 5? 5
```

```
              <<<    MAIN MENU SELECTION    >>>
------------------------------------------------------------
TRANSACTION (1), REPORT (2), OR EXIT (3)? 3
```

*Typing 2 would produce the message "report section not written yet." This is a "hook" to a future expansion.*

```
*** END OF PERIOD 2 TRANSACTIONS FOR 1984 ***
..................................................

STARTING CHECKING ACCOUNT BALANCE WAS    $ -131.30
CLOSING CHECKING ACCOUNT BALANCE IS      $  830.70

STARTING CK # THIS PERIOD WAS   123
HIGHEST CK # THIS PERIOD WAS    125

>>> THE NEXT TIME THIS PROGRAM IS RUN RESPOND AS FOLLOWS:
FOR PERIOD # ? 3
STARTING BALANCE THIS PERIOD ? 830.7
STARTING CK # THIS PERIOD ? 126

PROGRAM IS ABOUT TO TERMINATE.  VERIFY O.K. (Y/N)? Y
```

## High-Level Design

This program is going to be the most ambitious one so far. We've shown that the trick to designing large programs is to organize them as a collection of smaller programs sometimes called program *modules*. For a more complex program, the high-level design will want to say not only what the program modules are, but also describe how they interconnect.

A good tool for showing the interconnections between program modules is a *macro* flow diagram. This is similar to a flowchart, except that only the major ideas are shown. Detail is suppressed so that the designer can "think big thoughts," postponing the description of details as long as possible.

The next figure shows what a macro flow diagram for our MONEY program might look like. You'll notice that it contains a hook for the *TAXRPT* program that will be coming along later on. For the present, we'll concentrate on the transaction part of the program since that's where all the data is gathered. The job of generating reports (in *TAXRPT)* is properly postponed as a second stage project. Incidentally, what we've called a hook is also referred to as a *program stub* (since it is incomplete).

100 INITIALIZATION

210 MAIN MENU SELECTION

9910 EXIT

EXIT

1010 SUB-MENU FOR MONEY TRANSACTIONS

5010 "HOOK" TO REPORT SUB-MENU AND TAXRPT

1410 SETUP G PARAME-TERS FOR CHECK PAYMENTS

1070

1510 SETUP G PARAME-TERS FOR CASH PAYMENTS

1070

1610 SETUP G PARAME-TERS FOR CHECK ACCT DEPOSITS

1070

1710 SETUP G PARAME-TERS FOR CASH RECEIPTS

1070

1110 G(ENERAL) TRANSACTION MODULE

1150 PROCESS ONE TRANSACTION; PUT IN G ARRAY
1165 IF MO/DAY= 0, 0 GOTO 1210 ELSE GOTO 1150
1210 CHOOSE ONE OF FIVE OPTIONS (E, R, L, V, C)

1230 E(XIT)

1222 R(ESUME) GOTO 1135

1260 L(IST) LOOP THROUGH G ARRAY GOTO 1210

1225 V(OID)"HOOK" GOSUB 1910 GOTO 1210

1245 C(HANGE) COLLECT CHANGED DATA GOTO 1210

3010 SUBROUTINE TO COLLECT DATA AND STORE IT IN THE G ARRAY

4010 CLEANUP MODULE

4030 "HOOK" TO DISK SAVE ROUTINE 4110 GOTO 4020

4040 RESUME DOING SAME KIND OF TRANSACTION (GOTO 1134)

4050 EXIT

4510 SAVE G PARAMETERS. GOTO 1010

1910 SUBROUTINE TO VOID TRANSACTIONS

## Coding the Program

The best way to study the listing of MONEY is in conjunction with the macro flow diagram. The BASIC line numbers in each macro box correspond to actual line numbers in the final program.

The trickiest part of the program is the way in which a single *procedure* (called the GENERAL TRANSACTION MODULE) at line 1110 handles four different kinds of transactions. These are: check payments; cash payments; checking account deposits* and cash receipts. This is possible because the four transactions are similar, differing mainly in the names they bear, and in whether they are added or subtracted from the final balance (the explanations for TR$ and M1 given below explain how these differences are handled).

*In our program we call either checks or cash put in the checking account deposits (DEP). If a check received is not deposited, it will be treated as a cash receipt (CSH).

The following list of variable names and data structures will also help you read and follow the final program listing. Incidentally, line 112 (DEFINT C-P) says "all variable names beginning with C up to (and including) P will be integers," while line 114 (CLEAR 2000) means "set aside 2000 locations (bytes) for storing string (character) information in arrays like GV$(I), GD$(I), GF$(I), and GL$(I)."

| Variables | Meaning |
|---|---|
| IY | Last two digits of the year. |
| P | Reporting period number. |
| BS | Balance in checking account at the start of the period. |
| BC | Current balance in checking account. |
| CA$ | Name of checking account bank. |
| CS | First check # at the start of the period. |
| CK | Current check # about to be processed. |
| CH | Highest check # processed in this period. |
| DS | First deposit # at the start of the period. |
| DK | Current deposit # about to be processed. |
| DH | Highest deposit # processed in this period. |
| PS | First cash payment # at the start of the period (=1). |
| PK | Current cash payment # about to be processed. |
| PH | Highest cash payment # processed in this period. |
| HS | First cash receipt # at the start of the period (=1). |
| HK | Current cash receipt # about to be processed. |
| HH | Highest cash receipt # processed in this period. |
| F$, D$, F1$, F2$ and LP$ are printing formats. | |
| GD( ) | General array to hold dates. |
| GA( ) | General array to hold amounts. |
| GV$( ) | General array to hold vendors names. |
| GD$( ) | General array to hold descriptions ('for ...). |
| GL$( ) | General array to hold tax form and line number. |
| GP( ) | General array to hold % of GA for tax purposes. |
| GC( ) | General array to hold accounting category number. |
| MS | Integer which expresses menu selection. |
| TR$ | General transaction identifier for screen display ("CK#", "PAY#", "DEP#", or "CSH#"). |
| GS | General transaction starting number. |
| GK | General transaction current number. |
| GH | General transaction highest number. |
| XX | Starting number for voiding transactions. |
| M | Month of transaction. |
| D | Day of transaction. |
| AM | Variable to temporarily hold amount of transaction. |
| M1 | Multiplier for calculating balance (+1 for DEP, −1 for CK, 0 otherwise). |
| A$ | Inputted answer to question. |

| | |
|---|---|
| C1 | Parameter for transforming the transaction numbers into a selected range of G array indices, I. This technique is explained further in the *Department of Further Explanation,* below. |
| MAX | The maximum array index for each type of transaction. |

*Note:* In line 3025, inputting −1 is a signal to re-do the line.
*Note:* In line 3027, MM means "multiple payees (or payers)" for a transaction. This feature is not implemented in the present program.
*Note On TR$:* the identifier CK# is used for checks you wrote to others; PAY# refers to cash payments you made to others; DEP# means deposits made to your checking account (it could mean checks or cash, but for some people it will be synonymous with "checks deposited"); CSH# refers to money you received from others that was not deposited in your checking account.

## Department of Further Explanation

One way to design this program would be to use four different subprograms to handle the four different kinds of transactions, (check paid, cash paid, deposits made, and cash receipts not deposited). But these are very similar kinds of transactions, so a more efficient approach would be to write one general transaction subprogram, and then adapt it to the four specific transactions. This is the approach our program uses.

The general transaction subprogram is found in lines 1110 to 1290. To show how it works, here's a picture of the data structure it uses, along with some typical data that might be stored in it. The structure consists of 7 arrays used as follows:

| | Date | Amount | Vendor | Description | Tax Form | Tax Percent | Account # |
|---|---|---|---|---|---|---|---|
| I | GD(I) | GA(I) | GV$(I) | GD$(I) | GF$(I) | GP(I) | GC(I) |
| 0 | −1 | | | | | | |
| 1 | 1022 | 42.59 | AJAX BOOKS | COMPUTER BK | CC26 | 100 | 680 |
| 2 | 1030 | 189.60 | MUTUAL CO. | INSURANCE | AA5 | 0 | 740 |
| 3 | 1104 | 250.00 | BROKEN ARMS | RENT | CC16 | 20 | 320 |
| 4 | −1 | | | | | | |
| . | . | | | | | | |
| . | . | | | | | | |
| . | . | | | | | | |
| 99 | −1 | | | | | | |
| 100 | −1 | | | | | | |
| 101 | 914 | 12.00 | YELLOW CAB | TAXI | EB3 | 100 | 210 |
| 102 | 1123 | 37.50 | LA FONDUE | BUS. MEAL | CC21 | 50 | 720 |
| 103 | −1 | | | | | | |
| . | . | | | | | | |
| . | . | | | | | | |
| . | . | | | | | | |
| 149 | −1 | | | | | | |
| 150 | −1 | | | | | | |
| 151 | 804 | 812.00 | GLOBAL INTL | AUG. PAY | AA1 | 100 | 410 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 152 | 1123 | 150.00 | BYTE | ARTICLE | CC4 | 100 | 430 |
| 153 | −1 | | | | | | |
| . | . | | | | | | |
| . | . | | | | | | |
| . | . | | | | | | |
| 189 | −1 | | | | | | |
| 190 | −1 | | | | | | |
| 191 | 1114 | 20.00 | UNCLE BEN | GIFT | GG1 | 100 | 506 |
| 192 | −1 | | | | | | |
| . | . | | | | | | |
| . | . | | | | | | |
| . | . | | | | | | |
| 200 | −1 | | | | | | |

The −1 is put in each date entry during initialization. It is overwritten when actual data is entered. It will prove useful later on when we want to print out just those contents of the array that contain real data, since our print routine can say IF GD(I) < 0 THEN "don't print." The spaces at positions 0, 100, 150, and 190 are reserved for possible future use as *headers* for each sub-section of the array.

Each of the arrays is divided into four parts, with each part reserved for a specific kind of transaction. For example, although array elements for the date are all called GD(I), for "checks paid" I goes from 1 to 99, for "cash paid" I goes from 101 to 149, for "deposits made" I goes from 151 to 189, and for "cash received" I goes from 191 to 200.

How does the general transaction subprogram know which of these parts of the array to use? The answer is in the formula

$$I = (GK - GS) - C1$$

The quantities GK, GS, and C1 are changed for each kind of transaction. GK means the general current transaction number, GS means the general starting transaction number, and C1 means the place in the array where the particular kind of transaction starts. For "checks paid" C1 = 1, for "cash paid" C1 = 101, for "deposits made" C1 = 151, and for "cash received" C1 = 191.

C1 is called a *parameter* (a constant that depends on the type of transaction). It is "passed on" to the general transaction subprogram in one of the subroutines you see at lines 1410, 1510, 1610, or 1710. You will also see that a number of other parameters are passed by these subroutines, and that's what customizes the general transaction subprogram for each kind of special transaction. For example, for a checking account deposit with a current deposit number of DK = 20, where the deposit numbers started at DS = 15, line 1610 makes GS = 15, and GK = 20. It also makes C1 = 151, so our formula for I becomes

$$I = (GK - GS) + C1 = 20 - 15 + 151 = 156$$

Thus check deposit number 20 has all its information (date, amount, vendor, etc.) stored in the 156th location of each G array (in GD(156), GA(156), GV$(156), and so on).

Here's a listing of MONEY. It's a long program, but if you read it in conjunction with the macro flow diagram we gave earlier, and with a pic-

ture of how the data is stored in the various G arrays, it should start to make sense after a few days (and nights) of study.

```
10 ' ***************************************************
15 ' *                    MONEY                        *
20 ' *   COMPUTERIZED FINANCIAL/TAX REPORTING SYSTEM   *
40 ' ***************************************************
100 ' -------------------------------------------------
101 '      INITIALIZE VARIABLES; DECLARE DATA TYPES
102 ' -------------------------------------------------
105 CLS
110 PRINT "          <*>   FINANCIAL TAX/REPORTING SYSTEM   <*>
112 DEFINT C-P
114 CLEAR 2000
115 PRINT STRING$(60,"-")
117 '
120 INPUT "FOR CALENDAR YEAR 19";IY
130 IF IY<70 OR IY>99  PRINT "INVALID YEAR" : GOTO 120
140 INPUT "FOR PERIOD #"; P
150 IF P<1 OR P>366 PRINT "INVALID PERIOD #" : GOTO 140
160 INPUT "STARTING BALANCE THIS PERIOD (E.G. 589.65)"; BS
165 BC=BS              'INITIALIZE CURRENT CK BALANCE
170 INPUT "CK. ACCT. OR BANK NAME"; CA$
175 INPUT "STARTING CK. # THIS PERIOD"; CS
176 CK=CS              'CK IS CURRENT CK ABOUT TO BE PROCESSED
177 CH=CS-1            'CH IS HIGHEST CK # ALREADY PROCESSED
178 '
179 PRINT:PRINT TAB(25);"STAND BY"
180 DS=1 : PS=1 : HS=1 'STARTING DEP, CASH PAY, & CASH RCVD #S
181 DK=DS : DH=DS-1    'CURRENT & HIGHEST DEP #
183 PK=PS : PH=PS-1    'CURRENT AND HIGHEST CASH PAY #
184 HK=HS : HH=HS-1    'CURRENT AND HIGHEST CASH RCVD #
185 F$="### ##/## $####.##  %            % %         % %   % ### ###
186 D$="$#####.##"
187 F1$="####: ##/##/##  $#####.##   %              %"
188 F2$=" %              % %   % ###  ###"
189 LP$=F1$+F2$
190 DIM GD(200),GA(200),GV$(200),GD$(200),GF$(200),GP(200),GC(200)
198 FOR J=1 TO 200: GD(J)=-1: NEXT J
200 ' -------------------------------------------------
201 '                MAIN MENU SELECTION
202 ' -------------------------------------------------
210 CLS
220 PRINT "                <<<    MAIN MENU SELECTION   >>>
225 PRINT STRING$(60,"-")
230 INPUT "TRANSACTION (1), REPORT (2), OR EXIT (3)";  MS
240 IF MS<1 OR MS>3 THEN 230
250 ON MS GOTO 1010, 5010, 9910
1000 ' =================================================
1001 '                TRANSACTION PROGRAMS
1003 ' -------------------------------------------------
1004 '        TRANSACTION  SUB-MENU SELECTION
1005 ' -------------------------------------------------
1010 CLS
1020 PRINT "          <   TRANSACTION MENU SELECTION   >"
1030 PRINT STRING$(60,"-")
1040 PRINT"      1 = CK PAYMENTS BY YOU"
1042 PRINT"      2 = CASH PAYMENTS BY YOU"
1044 PRINT"      3 = CK RECEIPTS DEPOSITED BY YOU"
1046 PRINT"      4 = CASH RECEIPTS"
1048 PRINT"      5 = EXIT FROM TRANSACTION SECTION"
```

```
1050 PRINT
1055 INPUT"1, 2, 3, 4, OR 5";  MS
1060 IF MS<1 OR MS>5 THEN 1055
1065 IF MS=5 THEN 210
1070 ON MS GOSUB 1410, 1510, 1610, 1710
1100 ' -----------------------------------------------
1101 '              MAIN TRANSACTION PROGRAM
1102 ' -----------------------------------------------
1110 CLS
1115 PRINT "*** ";LEFT$(TR$,3);" TRANSACTIONS FOR PERIOD";P;"OF";1900+IY
1120 PRINT "***STARTING CK BALANCE THIS PERIOD =";TAB(36); USING D$;BS
1125 PRINT "***CURRENT CK BALANCE THIS PERIOD =";TAB(36); USING D$;BC
1130 PRINT "TO MAKE A CHANGE  OR EXIT TYPE 0,0 FOR MO,DAY"
1132 PRINT "TO RESTART A LINE TYPE -1 IN ANY COLUMN"
1134 PRINT STRING$(60,"-")
1135 PRINT TR$;"  MO,DAY: AMT:      VENDOR:      ";
1140 PRINT "FOR:        TAX CODE: %: ACT#"
1150 GK=GH+1         'GK IS TR. # ABOUT TO BE PROCESSED
1155 I=(GK-GS)+C1      'INDEX TO TR. RECORDS IN G ARRAY
1157 IF I>MAX PRINT "OUT OF SPACE" : GOTO 1210
1160 GOSUB 3010     'COLLECT TR. DATA
1165 IF M=0 AND D=0 THEN 1210
1170   GH=GH+1
1175   GOTO 1150
1200 '
1201 '.............TRANSACTION EXIT ROUTINE.............
1210 PRINT "EXIT(E), CHANGE(C), RESUME(R), OR LIST(L)";
1215 INPUT A$
1222 IF A$="R" THEN 1135
1230 IF A$="E" THEN 4010
1235 IF A$="L" THEN 1258
1237 IF A$="C" THEN 1245
1240 GOTO 1210
1241 '
1242 '.............CHANGE ROUTINE.............
1245 PRINT "CHANGE DATA FOR WHICH #";:INPUT GK
1247 IF GK>GH PRINT TR$" # TOO HIGH": GOTO 1210
1248  IF GK<GS PRINT "TOO LOW" : GOTO 1210
1250   I=(GK-GS)+C1 : BC=BC-M1*GA(I)*.01
1253   PRINT TR$;"  MO,DAY: AMT:      VENDOR:      ";
1254   PRINT "FOR:        TAX CODE: %: ACT#"
1256   GOSUB 3010 : GOTO 1210
1257 '
1258 '.............LIST ROUTINE................
1259 PRINT "LIST OF ";LEFT$(TR$,3);" TRANSACTIONS NOW IN G ARRAY"
1260 FOR J=GS TO GH
1261   K=(J-GS)+C1
1263   M=INT(GD(K)/100) : D=GD(K)-M*100
1265   PRINT USING F$;J,M,D,GA(K)*.01,GV$(K),GD$(K),GF$(K),GP(K),GC(K)
1270 NEXT J
1290 GOTO 1210
1400 '---------------------------------------------
1401 '    SUBROUTINES TO PASS TRANSACTION PARAMETERS
1402 '---------------------------------------------
1410 TR$="CK #" : GS=CS : GH=CH : GK=CK : MAX=99 : C1=1 : M1=-1
1490 RETURN
1500 '
1510 TR$="PAY#" : GS=PS : GH=PH : GK=PK : MAX=149 : C1=101 : M1=0
1590 RETURN
1600 '
1610 TR$="DEP#" : GS=DS : GH=DH : GK=DK : MAX=189 : C1=151 : M1=1
1690 RETURN
```

```
1700 '
1710 TR$="CSH#" : GS=HS : GH=HH : GK=HK : MAX=199 : C1=191 : M1=0
1790 RETURN
3000 ' ------------------------------------------------
3001 '      SUBROUTINE TO COLLECT TRANSACTION DATA
3002 ' ------------------------------------------------
3010 PRINT GK;TAB(5);: INPUT M,D: IF M=0 AND D=0 THEN 3090
3012 IF M<0 OR D<0 THEN 3010
3015 GD(I)=100*M + D
3016 AM=0: GV$(I)="NO ENTRY": GD$(I)="NO ENTRY"
3017 GF$(I)="NONE": GP$=" ": GC(I)=0
3018 PRINT TAB(12);CHR$(27);:INPUT AM:IF AM<0 THEN 3010
3025 PRINT TAB(21);CHR$(27);:INPUT GV$(I):IF GV$(I)="-1"THEN 3010
3030 PRINT TAB(33);CHR$(27);:INPUT GD$(I):IF GD$(I)="-1"THEN 3010
3045 PRINT TAB(45);CHR$(27);:INPUT GF$(I):IF GF$(I)="-1"THEN 3011
3055 PRINT TAB(53);CHR$(27);:INPUT GP$:IF GP$="-1"THEN 3010
3057 BC=BC+AM*M1
3058 GA(I)=INT(100*AM)
3060 IF GP$="" OR GP$=" " THEN GP(I)=100 ELSE GP(I)=VAL(GP$)
3065 PRINT TAB(58);CHR$(27);:INPUT GC(I): IF GC(I)<0 THEN 3010
3090 RETURN
4000 '-------------------------------------------,--------
4001 '    TRANSACTION "CLEAN-UP" PROGRAM; DISK SAVE
4002 '---------------------------------------------------
4010 PRINT: PRINT"  < EXIT FROM TRANSACTION SECTION >"
4012 PRINT ">>> CURRENT BALANCE =";BC
4015 IF BC<0 PRINT "*** WARNING: CURRENT CHECKING BAL.IS NEGATIVE ***"
4020 PRINT"SAVE(S), RESUME(R), OR EXIT(E)";:INPUT A$
4030 IF A$="S" THEN 4110
4040 IF A$="R" THEN 1134
4050 IF A$="E" THEN 4510
4060 GOTO 4020
4100 '
4101 '.............DISK SAVE ROUTINE.............
4110 PRINT "NO DISK SAVE YET": GOTO 4020
4510 '
4530 CK=CH+1
4550 IF TR$="CK #":CS=GS:CH=GH:CK=GK: GOTO 1010
4555 IF TR$="DEP#":DS=GS:DH=GH:DK=GK: GOTO 1010
4560 IF TR$="PAY#":PS=GS:PH=GH:PK=GK: GOTO 1010
4565 IF TR$="CSH#":HS=GS:HH=GH:HK=GK: GOTO 1010
4570 PRINT "ERROR IN TR$ -- IT = ";TR$: STOP
5000 ' ================================================
5001 '              REPORT GENERATING PROGRAMS
5002 '
5010 PRINT "REPORT SECTION NOT WRITTEN YET"
5055 FOR J=1 TO 500: NEXT J     'DELAY LOOP
5090 GOTO 210
9000 ' ================================================
990( ' ------------------------------------------------
9901 '               FINAL EXIT MESSAGES
9902 ' ------------------------------------------------
9910 CLS: PRINT "*** END OF PERIOD";P;"TRANSACTIONS FOR";1900+IY;"***"
9915 PRINT STRING$(50,"."): PRINT
9920 PRINT USING "STARTING CHECKING ACCOUNT BALANCE WAS $#####.##";BS
9930 PRINT USING "CLOSING CHECKING ACCOUNT BALANCE IS   $#####.##";BC
9935 PRINT
9940 PRINT "STARTING CK # THIS PERIOD WAS  ";CS
9950 PRINT "HIGHEST CK # THIS PERIOD WAS   ";CH: PRINT
9960 PRINT ">>> THE NEXT TIME THIS PROGRAM IS RUN RESPOND AS FOLLOWS:"
9965 PRINT "FOR PERIOD # ?";P+1
```

```
9970 PRINT "STARTING BALANCE THIS PERIOD ?";BC
9975 PRINT "STARTING CK # THIS PERIOD ?";CH+1 : PRINT
9980 PRINT "PROGRAM IS ABOUT TO TERMINATE.  VERIFY O.K. (Y/N)";
9985 INPUT Z$:  IF LEFT$(Z$,1)<>"Y" THEN 210
9999 END
```

## Project *MONEY2*

The high-level flow diagram for MONEY shows a subroutine beginning at line 1910 to be used for voiding a group of transactions. This subroutine is missing from MONEY, having been left as a "low-level"refinement to be added later on.

Write the code for a void subroutine, and add it to MONEY to make *MONEY2*. Also add the command V (void) as part of the instructions in line 1210. Insert a new line as follows:

1225 IF A$ = "V" THEN GOSUB 1910 : GOTO 1210

Thus when a user types "V" in response to the question at line 1210, he should be sent to subroutine 1910. There he should be asked at which transaction number the voiding should start. All transactions at and above this number should then be cancelled. As part of a void, don't forget to reset the highest transaction number processed (GH), and readjust the check balance (BC) where appropriate.

## Superproject *(CHECKBAL)*

The check balance reported by the program MONEY is based on all the checks you have written and entered into the program for a given period, and all the deposits you have made. The statement of your checking balance you receive from the bank may be for an earlier date, and so it will be lacking the latest information. As a result, it will be higher than the latest balance your program calculates by the sum of the checks you have written that were not yet returned to the bank. It will be lower than your balance by the sum of the deposits you made that have not yet been recorded by the bank.

Write a program module for MONEY that asks you for the amounts of the missing checks and missing deposits, and then prints a balance that compares with the bank's statement. Write this module as a report program that starts at line 6010. You will also want to write a menu subprogram for selecting reports (starting at line 5010) which has as its initial choice this "checking account balance" report.

**Note:** There will be other reports added to this program in chapter 4. So far, MONEY is mainly a data gathering instrument that will seem to waste a lot of the information you type into it. Do not despair—it will become increasingly useful as more and more report modules are added. If you need some help with *(CHECKBAL)*, take a look at the program MONYLIST is section 4.2. This shows how to write the menu subprogram for selecting reports at line 5010, and it contains an example of a report program at line 7010.

## Exercises for Chapter 3

The following exercises contain several useful programming techniques connected with the idea of sorting. They are more advanced than the previous exercises, and will require more time to complete.

```
10 REM %%% EXERCISE 8 (INSERTION SORT).  SIMULATE A RUN %%%
15 PRINT "NUMBERS TO BE SORTED ARE:"
18 N=10
20 FOR K=1 TO N: READ R(K): PRINT R(K): NEXT K
25 DATA 23,67,12,90,2,52,86,20,13,88
30 FOR J=2 TO N
70     I=J-1: R=R(J)
80     IF I<=0 THEN 120
85     IF R>=R(I) THEN 120
90       R(I+1)=R(I)
100      I=I-1
110      GOTO 80
120    R(I+1)=R
130 NEXT J
180 PRINT "*** READY TO SEE SORTED NUMBERS ***";:INPUT D$
190 FOR K=1 TO N: PRINT R(K): NEXT K
200 END
```

```
10 REM %%% EXERCISE 9.  MODIFY EXERCISE 8 SO THAT IT FILLS
11 REM %%% THE R() ARRAY WITH 100 RANDOM NUMBERS, AND THEN
12 REM %%% SORTS THE ARRAY IN ASCENDING ORDER.
```

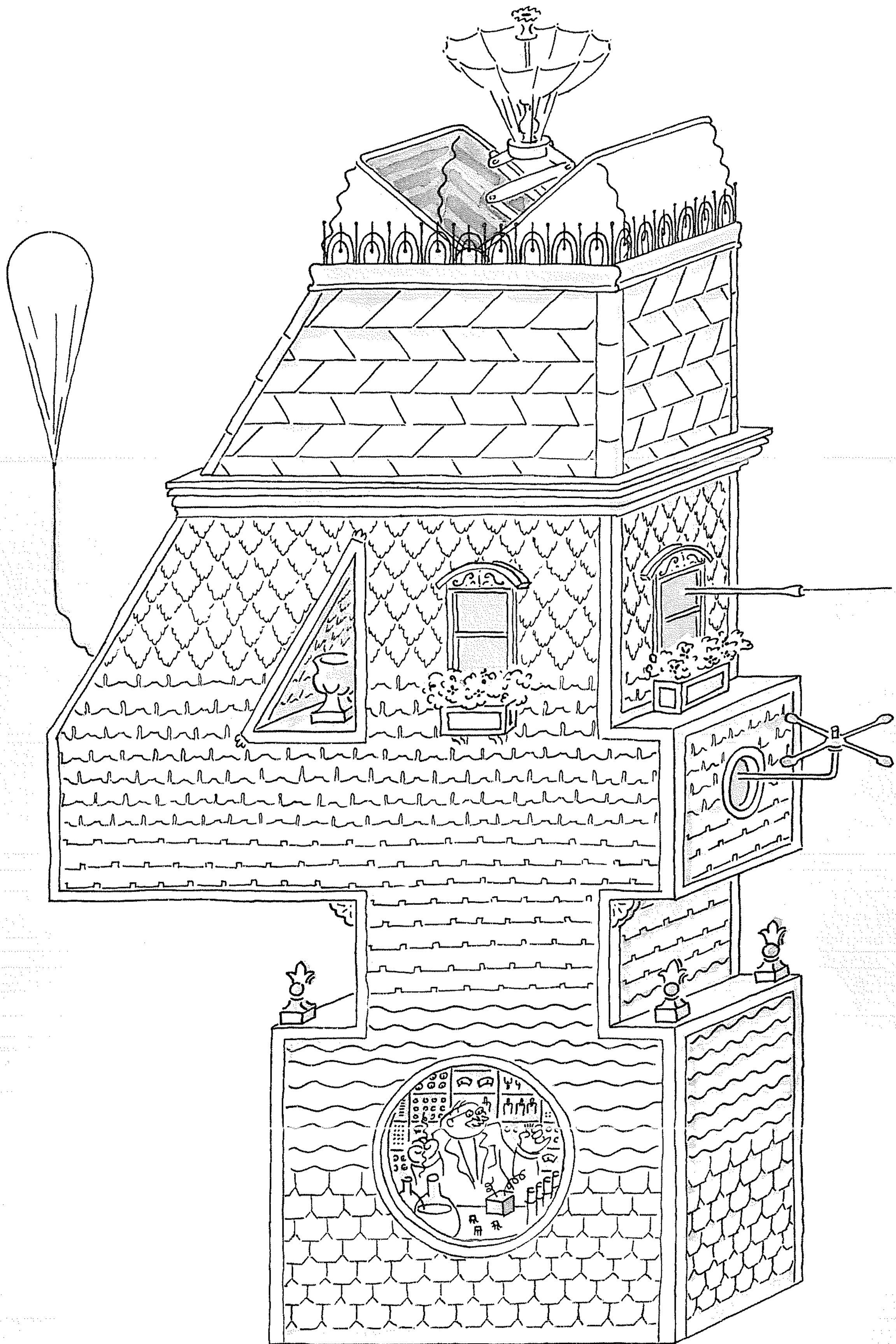
```
9 REM  %%% ----------------- EXERCISE 10 --------------------
10 REM %%% MODIFY EXERCISE 8 SO THAT IT READS STRINGS INTO AN
11 REM %%% ARRAY R$(), AND THEN SORTS AND PRINTS THE ARRAY IN
12 REM %%% ASCENDING ORDER.  FOR SOME HELP SEE 'BIT OF BASIC'
13 REM %%% BY T. DWYER & M. CRITCHFIELD (ADDISON WESLEY CO.)
```

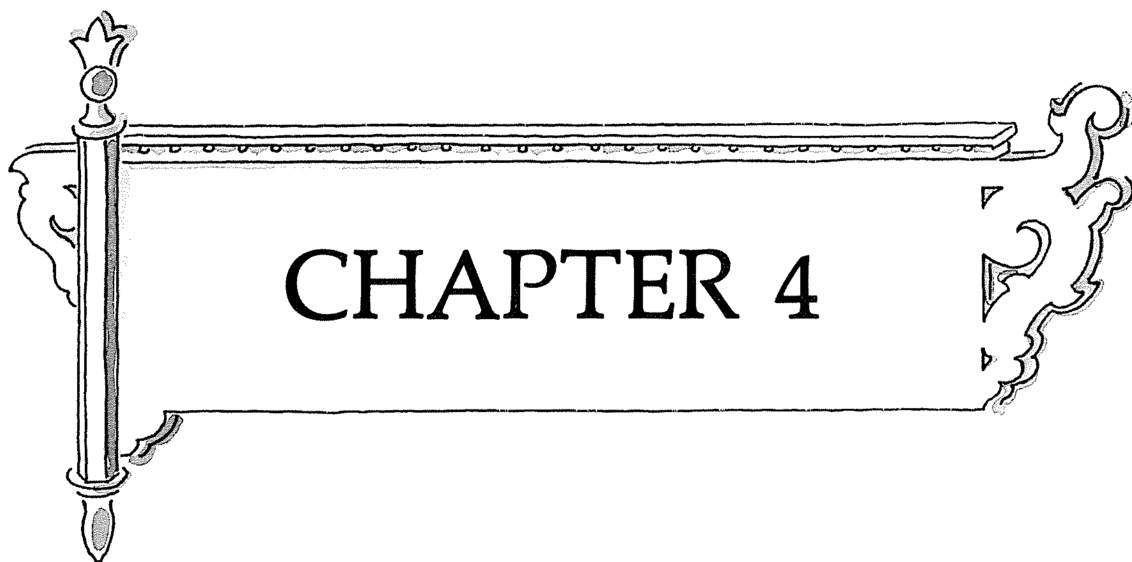
## SOURCES OF FURTHER INFORMATION FOR CHAPTER 3

Business application programs often appear in the magazines and books mentioned at the end of Chapter 1. Two additional magazines that publish business applications are:

1. *Interface Age* (P.O. Box 1234, Cerritos, CA 90701).
2. *Microcomputing* (Peterborough, NH 03458)

There are a number of application areas that are similar to business programs even though they don't involve financial record keeping. Data based diet information programs would fall into this category. Several examples of diet information programs can be found in Chapter 8 of the book *BASIC and the Personal Computer* (referenced at the end of Chapter 1). This chapter also explains a business invoicing program.






# HOW TO TAKE THE PLUNGE: UPGRADING TO A BIGGER COMPUTER SYSTEM

## 4.0 INTRODUCTION

The applications shown up to this point were designed for use on personal computers with a consumer price tag —machines costing about the same as a hi-fi system or video tape recorder. In this chapter we'll look at what's possible if and when you decide to expand your computer into a more sophisticated system, particularly by adding professional-grade peripherals. A *peripheral* is any device that can be connected to a computer. The two most valuable peripherals for serious computing are *printers* and *disk drives,* and we'll give most of our attention to discussing how these can be used effectively.

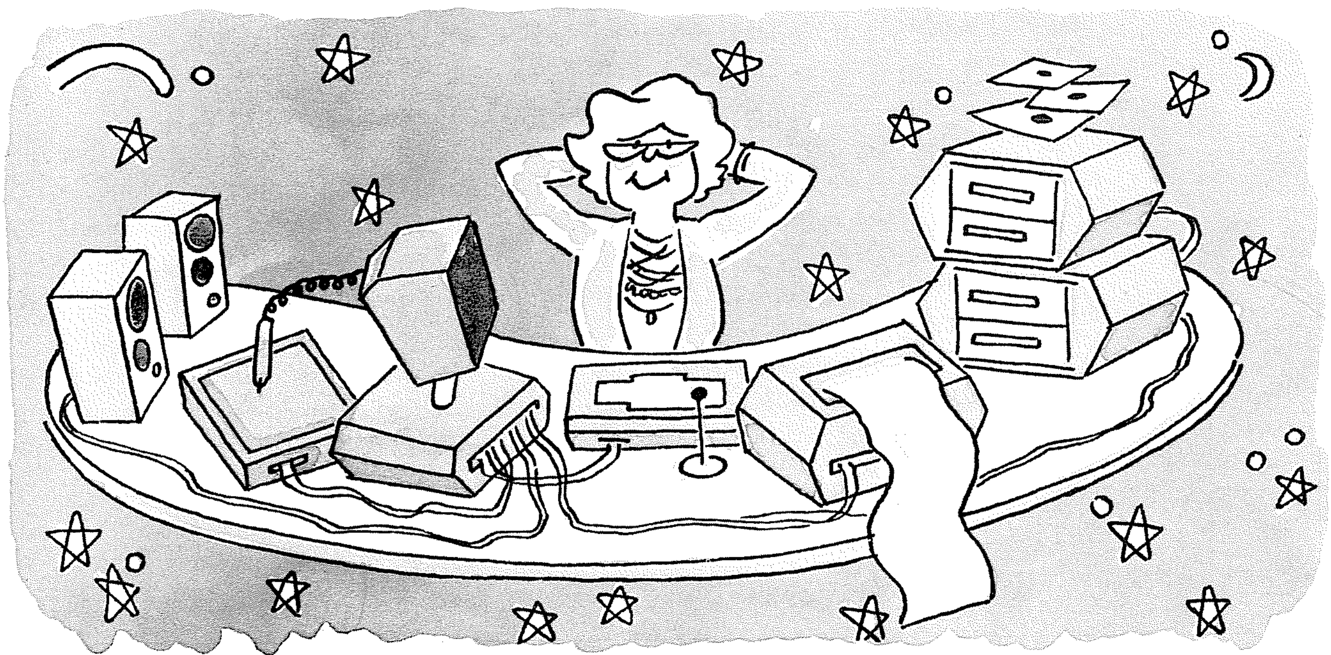
We'll also spend some time explaining the features of what is usually called *disk-extended BASIC,* basing our discussion on the version developed by the Microsoft company. Microsoft disk-extended BASIC is available for most personal computers, and it includes features that are very similar to those found in BASIC PLUS, a professional-level language used on Digital Equipment Company computers. Our examples were all run on a Radio Shack TRS-80 system using the *disk operating system* called TRSDOS (TRS-80 Disk Operating System). When this disk system is

added to a Level II TRS-80 computer, it gives you all the important features of Microsoft disk-extended BASIC.

The third thing we will do in this chapter is introduce some new techniques from the world of computer science. In particular, we will discuss several file handling techniques in section 4.3, including something called *hashing,* and explain the use of *linked list* data structures in section 4.4. This will then open the door to writing a text editing program which will serve as the basis for a useful word processing system.

## 4.1 WHY MORE EQUIPMENT? EXAMPLES OF WHAT'S POSSIBLE

A computer system can be expanded in two ways: by adding external devices (peripherals), and by expanding the circuitry within the computer itself. New circuitry can often be added by plugging additional circuit boards into empty *expansion slots.* Another possibility on some machines is to plug small plastic enclosures called *integrated circuit* (IC) chips into sockets left open for that purpose. The two most popular circuit additions are *memory* boards, or memory chips, and *interface* boards, (or in some cases, interface expansion boxes.)



### Why Add More Memory?

Microcomputers usually have two kinds of memory: read only memory (usually abbreviated ROM), and programmable memory (also called main memory, user memory, or simply memory.) Often the term RAM, or random access memory, is used in the microcomputer literature for programmable memory. The word random means that each byte can be accessed directly without running through all the preceding bytes in sequence. ROM memory also has this property, so you can think of ROM as meaning "random read only memory".

ROM is used to store permanent information, such as the BASIC interpreter needed to translate your programs into detailed machine code. The user cannot change the contents of ROM memory. User memory, on the other hand, is used to hold the latest BASIC program you have written or loaded from tape or disk. It's also used to hold data—often transaction

data of the kind we have seen in the programs ESTIMATE, SALESLIP, and MONEY. For example, in MONEY, the transactions were stored in seven arrays of 200 locations each, so there were 1400 locations of RAM reserved (dimensioned) for this program. Some of these locations held numbers while others held strings. Let's see how much memory is actually required for holding numbers and strings.

The basic unit of memory storage is one *byte*. A byte consists of 8 binary digits called *bits*. A bit can only have one of two values, called 0 and 1. If you think about all the ways 8 bits can be arranged, you discover that there are 256 possibilities, so one byte can store this many different kinds of information.

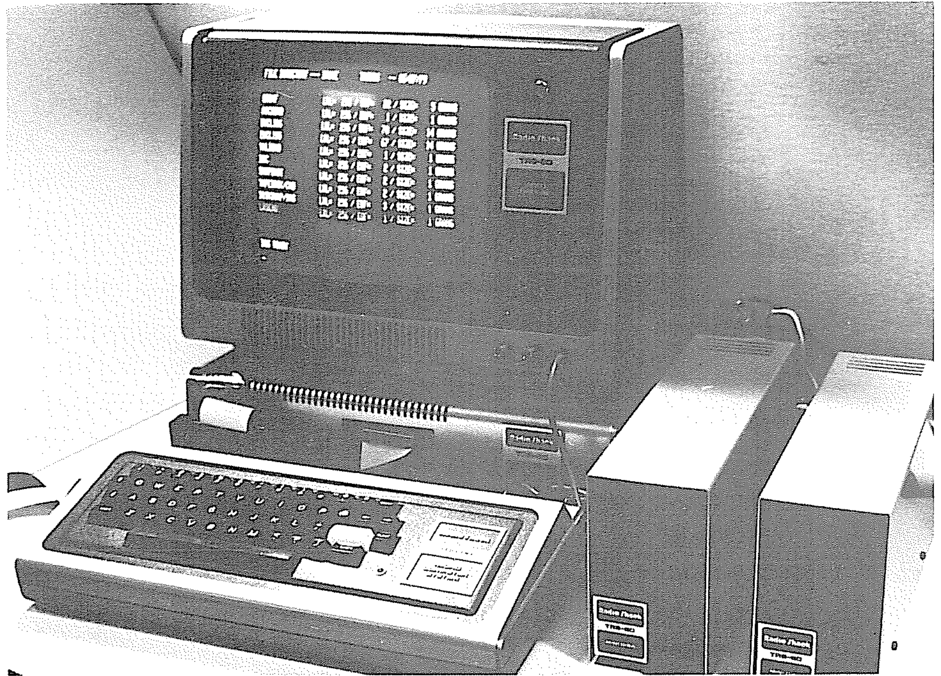
Bytes are sometimes used to store characters. For example, the character A is stored as the byte 01000001. Bytes can also be used to represent numbers. For example, the pattern 01000001 can also be interpreted as the number 65 (because it represents $2^6 + 2^0 = 64 + 1$). Since most applications require numbers greater then 256, two bytes are usually used to represent integers, and four bytes are used to represent *floating point numbers,* numbers with decimals that are moved around automatically, like 3.14159 or 314.159. Strings take as many bytes for storage as there are characters in the string. For example, "GOOD MORNING" takes 12 bytes. Your BASIC program also takes one byte for each character used in typing it.

The conclusion to all this is that if you want to write long programs, or use lots of numbers or strings, you'll need plenty of programmable memory. A small system may have 4,000 bytes of memory (abbreviated 4 K), a medium system may have 16 K, and a large system 32 K to 64 K. All the programs in the previous two chapters will work on a machine with 16 K bytes of memory; the programs in chapter 1 require less than 4 K bytes.

### Why Disks?

Although 48 K bytes of user memory is a lot, there are many applications where even more storage is needed. The solution to this program is to use magnetic disks as a form of *external* or *mass* storage to hold the data for such applications. We'll explain how this is done in section 4.3.

Another use of disks is to store your BASIC programs. You do this by giving each program a file name (for example, BABYQ) and then typing SAVE "BABYQ". When you want to use this program at a later date, you simply type LOAD "BABYQ", and a copy of the program will be transferred from disk into main memory. The file names are limited to 8 characters on many systems (now you know why some of our program names were spelled in abbreviated form—like *MONYDISK).* There are many special features which can make a file name more meaningful or protect the file from accidental damage. The disk manual for your system should be consulted to see what these are, and how they are used.



*This is the model I TRS-80 computer equipped with an expansion interface box (under the video display), and two minidisk drives (on the right). The display on the screen is a directory of the programs (or "files") stored on one of the disks.*

Another reason for adding disks is that you can then use what's called a disk operating system (DOS). This is a collection of system software programs. One of the most important of these is an extension of BASIC called disk-extended BASIC. This includes many additional language features, and we'll discuss a number of these in section 4.3. One you should know about now is LINE INPUT, used as follows:

```
130 LINE INPUT A$
```

This differs from INPUT in two ways. First, it does not cause a ? to be printed. Second, it allows you to type a string for input that includes commas as part of the string. This last feature will be very important for the text editor program described later in section 4.4.

## Why a Printer?

Printers enable you to get a permanent record on paper, called *hard copy* of program listings and program output. Several kinds of printers are available. The distinction between line printers and sequential character printers was explained at the end of section 3.5 as part of project *HARD-SALE.* Most printers used with microcomputers are of the sequential character type, printing one character at a time. Another factor that must be considered is the type of *interface* used. The word interface describes the circuitry used to transmit signals between the computer and its peripherals—in this case to transmit characters from computer to printer.

There are two kinds of printer interfaces: *parallel* (information sent 8 bits at a time), and *serial* (information sent 1 bit at a time). Serial interfaces are a little more expensive, but they provide more flexibility. For example, only serial information can be sent over telephone lines. The most common serial interface is called the RS-232-C standard. We won't go into further detail, but advise that before you buy a printer you should make sure your computer can interface to it properly. The best way is to work with a local dealer who will guarantee that the combination you select will work.

The other decision you have to make is choosing between a *dot matrix* and a *letter-quality impact* printer. Below is an example showing the difference. As you can see, the letter-quality printer is preferable for such applications as letter writing, contract preparation, or in general, situations where quality and clarity are important.

```
>RUN
** COMPARISON OF SURVEYS **
TOTAL VOTERS =  10189
PERCENT REPUBLICANS =   44.8327 %
PERCENT DEMOCRATS   =   50.368 %
PERCENT OTHERS      =   4.79929 %
.................................
TOTAL VOTERS =  10189
PERCENT REPUBLICANS =   44.4793 %
PERCENT DEMOCRATS   =   50.3877 %
PERCENT OTHERS      =   5.13299 %
.................................
OUT OF DATA IN 10
```

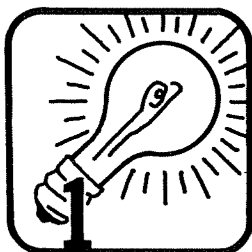
```
>RUN
** COMPARISON OF SURVEYS **
TOTAL VOTERS =  10189
PERCENT REPUBLICANS =  44.8327 %
PERCENT DEMOCRATS   =  50.368 %
PERCENT OTHERS      =  4.79929 %
.................................
TOTAL VOTERS =  10189
PERCENT REPUBLICANS =  44.4793 %
PERCENT DEMOCRATS   =  50.3877 %
PERCENT OTHERS      =  5.13299 %
.................................
OUT OF DATA IN 10
```

## 4.2 USING A PRINTER. MONYLIST, *TAXRPT*, SNAPSHOT

The programs in this section produce output that can be displayed on either a dot matrix or an impact printer. Some of these printers are designed for use with parallel interfaces of the type built into the TRS-80 expansion interface box. However, a number of impact printers can only be connected to a serial RS-232-C interface. Some computers have this interface built in, while others require the addition of a special RS-232-C circuit board.

*An RS-232-C interface board is shown here installed in the TRS-80 expansion box (top center of the photograph). A flat ribbon cable connected to this board can be seen coming out of the front of the expansion box. This cable is then connected to the letter-quality printer/terminal shown on the left. The keyboard on the printer/terminal is not used in this arrangement; all control of input and output comes from the TRS-80.*

### Program MONYLIST



### The Idea

The program MONEY in section 3.6 concentrated on the gathering of transaction data: checks written, cash paid, deposits made, cash received. It also allowed the user to input tax classification information: a tax-form

code, the percent used for taxes, and an account number. However, the program never used this information—in fact the only thing it did was to keep track of the checking account balance! It's time to correct this deficiency by expanding MONEY to make better use of all that data.

In MONYLIST we'll expand MONEY (or MONEY2) by first adding a menu selection routine at line 5010. This will allow the user to choose one of several report-generating programs, in the same way that the routine starting at line 1010 allows the user to choose between different kinds of transaction-gathering programs. Then we'll write the code for generating one of these reports, starting at line 7010. This report will be a listing of all the transaction data that is currently in the various G arrays: GD( ), GA( ), GV$( ), GD$( ), GF$( ), GP( ), and GC( ).

We'll assume that the report is to be listed on a printer. This is done on the TRS-80 by using LPRINT instead of PRINT in output statements. On other machines there is a special command to switch output from the video screen to the printer. On the Apple, for example, you type PR#1. On these machines you don't have to change PRINT to LPRINT—all output is automatically directed to the printer connected at the given interface slot (#1 in the Apple example).

### A Sample Run

Part of the run below is shown on the video display. This is followed by showing the actual *hard copy* produced on the printer.

```
          <   REPORT MENU SELECTION   >
------------------------------------------------------------
     0 = EXIT FROM REPORT SECTION
     1 = CHECKING ACCOUNT BALANCE
     2 = PRINTED LIST OF ALL TRANSACTIONS
     3 = TAX FORM REPORTS

0, 1, 2, OR 3? 2
TURN PRINTER ON -- READY? YEP
```

```
LIST OF ALL TRANSACTIONS FOR PERIOD 2 OF 1988
CHECKING ACCOUNT IS AT SEVENTH RESERVE BANK
STARTING BALANCE WAS $  500.00


*** CHECKING ACCOUNT PAYMENTS ***

CK#  DATE:  AMOUNT:    TO:              FOR:              FORM: TX%: ACCT:
123  11/12  $ 350.00   HAPPY PLAZA      SEPT. RENT        CC16   30  980
124  12/13  $ 230.00   COUNTY           TAXES             DD12  100  890
-------------------------------------------------------------------------
```

```
*** CASH PAYMENTS ***

PAY# DATE:  AMOUNT:    TO:               FOR:              FORM: TX% ACCT:
  1  12/15  $  20.00   TAXI              NO ENTRY          NONE  100    0
------------------------------------------------------------------------

*** CHECKING ACCOUNT DEPOSITS ***

DEP# DATE:  AMOUNT:    FROM:             FOR:              FORM: TX% ACCT:
  1  12/18  $ 200.00   BOSS              PAY               AA    100  340
  2  12/23  $ 300.00   TRUE CONFESSION   ARTICLE           CC16E 100  440
------------------------------------------------------------------------

*** CASH RECEIPTS NOT DEPOSITED ***

CSH# DATE:  AMOUNT:    FROM:             FOR:              FORM: TX% ACCT:
  1  12/ 1  $  23.00   NO ENTRY          NO ENTRY          NONE  100    0
------------------------------------------------------------------------

CURRENT CHECKING ACCT. BALANCE IS $  420.00
```

### High-Level Design

Since we planned on adding both these new features back in the design of MONEY (remember our *hooks*?), there's no need for much of a high-level design—it's already been taken care of. The main thing to do is to stick to the line numbers we reserved for the report menu selection and for the actual reports. When a report is finished, the program should always branch back to the report menu section. When EXIT is chosen from this menu, the program should then branch back to the main menu section.

### Coding the Program

The only new feature of BASIC needed for coding this program is the LPRINT statement which is used in several places starting at line 7015. Most of the printing is done by the subroutine in lines 7510 to 7590. Each part of the G array is printed in turn by calling on this subroutine. Notice that the subroutine contains a test to see if the data is −1 (IF GD(K)<0). This way no time will be wasted printing out rows of G that do not yet contain transaction data. Here's a complete listing of MONYLIST that includes all the features of *MONEY2* and MONEY:

```
10 ' ****************************************************
15 ' *                   MONYLIST                       *
20 ' *   COMPUTERIZED FINANCIAL/TAX REPORTING SYSTEM    *
40 ' ****************************************************
100 ' -------------------------------------------------
101 '      INITIALIZE VARIABLES; DECLARE DATA TYPES
102 ' -------------------------------------------------
105 CLS
110 PRINT "         <*>   FINANCIAL TAX/REPORTING SYSTEM   <*>
112 DEFINT C-P
114 CLEAR 2000
115 PRINT STRING$(60,"-")
120 INPUT "FOR CALENDAR YEAR 19";IY
130 IF IY<70 OR IY>99  PRINT "INVALID YEAR" : GOTO 120
140 INPUT "FOR PERIOD #"; P
150 IF P<1 OR P>366 PRINT "INVALID PERIOD #" : GOTO 140
160 INPUT "STARTING BALANCE THIS PERIOD (E.G. 589.65)"; BS
165 BC=BS               'INITIALIZE CURRENT CK BALANCE
170 INPUT "CK. ACCT. OR BANK NAME"; CA$
175 INPUT "STARTING CK. # THIS PERIOD"; CS
176 CK=CS               'CK IS CURRENT CK ABOUT TO BE PROCESSED
177 CH=CS-1             'CH IS HIGHEST CK # ALREADY PROCESSED
179 PRINT:PRINT TAB(25);"STAND BY"
180 DS=1 : PS=1 : HS=1 'STARTING DEP, CASH PAY, & CASH RCVD #S
181 DK=DS : DH=DS-1     'CURRENT & HIGHEST DEP #
183 PK=PS : PH=PS-1     'CURRENT AND HIGHEST CASH PAY #
184 HK=HS : HH=HS-1     'CURRENT AND HIGHEST CASH RCVD #
185 F$="### ##/## $####.## %             % %           % %   % ### ###
186 D$="$#####.##"
187 F1$="####: ##/##/##  $#####.##   %            %"
188 F2$=" %              % %   % ###  ###"
189 LP$=F1$+F2$
190 DIM GD(200),GA(200),GV$(200),GD$(200),GF$(200),GP(200),GC(200)
198 FOR J=1 TO 200: GD(J)=-1: NEXT J
200 ' -------------------------------------------------
201 '               MAIN MENU SELECTION
202 ' -------------------------------------------------
210 CLS
220 PRINT "              <<<    MAIN MENU SELECTION   >>>
225 PRINT STRING$(60,"-")
230 INPUT"EXIT (0), TRANSACTIONS (1), OR REPORTS (2)"; MS
240 IF MS<0 OR MS>2 THEN 230
250 ON MS+1  GOTO 9910, 1010, 5010
999 '
1000 ' ================================================
1001 '               TRANSACTION PROGRAMS
1002 '
1003 ' ------------------------------------------------
1004 '         TRANSACTION  SUB-MENU SELECTION
1005 ' ------------------------------------------------
1010 CLS
1020 PRINT "          <   TRANSACTION MENU SELECTION   >"
1030 PRINT STRING$(60,"-")
1035 PRINT"     0 = EXIT FROM TRANSACTION SECTION"
1040 PRINT"     1 = CK PAYMENTS BY YOU"
1042 PRINT"     2 = CASH PAYMENTS BY YOU"
1044 PRINT"     3 = DEPOSITS TO YOUR CHECKING ACCOUNT"
1046 PRINT"     4 = CASH RECEIPTS NOT DEPOSITED"
1050 PRINT
1055 INPUT"0, 1, 2, 3, OR 4"; MS
1060 IF MS<0 OR MS>4 THEN 1055
1065 IF MS=0 THEN 210
1070 ON MS GOSUB 1410, 1510, 1610, 1710
```

```
1099 '
1100 ' ----------------------------------------------------
1101 '            MAIN TRANSACTION PROGRAM
1102 ' ----------------------------------------------------
1110 CLS
1115 PRINT "***  ";LEFT$(TR$,3);" TRANSACTIONS FOR PERIOD";P;"OF";1900+IY
1120 PRINT "***  STARTING CK BALANCE THIS PERIOD =";TAB(36); USING D$;BS
1125 PRINT "***  CURRENT CK BALANCE THIS PERIOD =";TAB(36); USING D$;BC
1130 PRINT "TO MAKE A CHANGE  OR EXIT TYPE 0,0 FOR MO,DAY"
1132 PRINT "TO RESTART A LINE TYPE -1 IN ANY COLUMN"
1134 PRINT STRING$(60,"-")
1135 PRINT TR$;"  MO,DAY: AMT:     VENDOR:      ";
1140 PRINT "FOR:        TAX FORM: %: ACT#"
1150 GK=GH+1        'GK IS TR. # ABOUT TO BE PROCESSED
1155 I=(GK-GS)+C1      'INDEX TO TR. RECORDS IN G ARRAY
1157 IF I>MAX PRINT "OUT OF SPACE" : GOTO 1210
1160 GOSUB 3010     'COLLECT TR. DATA
1165 IF M=0 AND D=0 THEN 1210
1170   GH=GH+1
1175   GOTO 1150
1200 '
1201 '.............TRANSACTION EXIT ROUTINE.............
1210 PRINT "EXIT(E), CHANGE(C), RESUME(R), LIST(L), OR VOID(V)";
1215 INPUT A$
1222 IF A$="R" THEN 1135
1225 IF A$="V" THEN : GOSUB 1910: : GOTO 1210
1230 IF A$="E" THEN 4010
1235 IF A$="L" THEN 1258
1237 IF A$="C" THEN 1245
1240 GOTO 1210
1241 '
1242 '.............CHANGE ROUTINE.............
1245 PRINT "CHANGE DATA FOR WHICH #";:INPUT GK
1247 IF GK>GH PRINT TR$" # TOO HIGH": GOTO 1210
1248  IF GK<GS PRINT "TOO LOW" : GOTO 1210
1250   I=(GK-GS)+C1 : BC=BC-M1*GA(I)*.01
1253   PRINT TR$;"  MO,DAY: AMT:     VENDOR:      ";
1254   PRINT "FOR:        TAX CODE: %: ACT#"
1256   GOSUB 3010 : GOTO 1210
1257 '
1258 '.............LIST ROUTINE................
1259 PRINT "LIST OF ";LEFT$(TR$,3);" TRANSACTIONS NOW IN G ARRAY"
1260 FOR J=GS TO GH
1261   K=(J-GS)+C1
1263   M=INT(GD(K)/100) : D=GD(K)-M*100
1265   PRINT USING F$;J,M,D,GA(K)*.01,GV$(K),GD$(K),GF$(K),GP(K),GC(K)
1270 NEXT J
1290 GOTO 1210
1400 '-----------------------------------------------
1401 '    SUBROUTINES TO PASS TRANSACTION PARAMETERS
1402 '-----------------------------------------------
1410 TR$="CK #" : GS=CS : GH=CH : GK=CK : MAX=99 : C1=1 : M1=-1
1490 RETURN
1500 '
1510 TR$="PAY#" : GS=PS : GH=PH : GK=PK : MAX=149 : C1=101 : M1=0
1590 RETURN
1600 '
1610 TR$="DEP#" : GS=DS : GH=DH : GK=DK : MAX=189 : C1=151 : M1=1
1690 RETURN
1700 '
1710 TR$="CSH#" : GS=HS : GH=HH : GK=HK : MAX=199 : C1=191 : M1=0
1790 RETURN
```

```
1900 ' ---------------------------------------------------
1901 '        SUBROUTINE TO VOID TRANSACTIONS
1902 ' ---------------------------------------------------
1910 PRINT "  * HIGHEST TRANSACTION # PROCESSED SO FAR IS";GH
1920 PRINT "  * VOID REMOVES DATA FROM # XX ON UP. WHAT VALUE"
1925 PRINT "  * OF XX DO YOU WISH TO VOID FROM (-1 MEANS NONE)";
1930 INPUT XX
1950 IF XX<0 THEN 1990
1955 IF XX<GS PRINT "TR # TOO LOW.  XX =";:GOTO 1930
1957 IF XX>GH PRINT "TR # TOO HIGH.   XX =";:GOTO 1930
1960 FOR J=XX TO GH
1961   K=(J-GS)+C1
1962   BC=BC-M1*GA(K)*.01
1964 NEXT J
1968 GH=XX-1
1970   PRINT "ALL TRANSACTIONS FROM #";XX;"ON UP ARE  VOID"
1980   PRINT "HIGHEST TRANSACTION # PROCESSED IS NOW";GH
1990 RETURN
3000 ' ---------------------------------------------------
3001 '       SUBROUTINE TO COLLECT TRANSACTION DATA
3002 ' ---------------------------------------------------
3010 PRINT GK;TAB(5);: INPUT M,D: IF M=0 AND D=0 THEN 3090
3012 IF M<0 OR D<0 THEN 3010
3015 GD(I)=100*M + D
3016 AM=0: GV$(I)="NO ENTRY": GD$(I)="NO ENTRY"
3017 GF$(I)="NONE": GP$=" ": GC(I)=0
3018 PRINT TAB(12);CHR$(27);:INPUT AM:IF AM<0 THEN 3010
3025 PRINT TAB(21);CHR$(27);:INPUT GV$(I):IF GV$(I)="-1"THEN 3010
3030 PRINT TAB(33);CHR$(27);:INPUT GD$(I):IF GD$(I)="-1"THEN 3010
3045 PRINT TAB(45);CHR$(27);:INPUT GF$(I):IF GF$(I)="-1"THEN 3011
3055 PRINT TAB(53);CHR$(27);:INPUT GP$:IF GP$="-1"THEN 3010
3057 BC=BC+AM*M1
3058 GA(I)=INT(100*AM)
3060 IF GP$="" OR GP$=" " THEN GP(I)=100 ELSE GP(I)=VAL(GP$)
3065 PRINT TAB(58);CHR$(27);:INPUT GC(I): IF GC(I)<0 THEN 3010
3090 RETURN
4000 '--------------------------------------------,--------
4001 '    TRANSACTION "CLEAN-UP" PROGRAM; DISK SAVE
4002 '----------------------------------------------------
4010 PRINT: PRINT"  < EXIT FROM TRANSACTION SECTION >"
4012 PRINT ">>> CURRENT BALANCE =";BC
4015 IF BC<0 PRINT "*** WARNING: CURRENT CHECKING BAL. IS NEGATIVE ***"
4020 PRINT"SAVE(S), RESUME(R), OR EXIT(E)";:INPUT A$
4030 IF A$="S" THEN 4110
4040 IF A$="R" THEN 1134
4050 IF A$="E" THEN 4510
4060 GOTO 4020
4100 '
4101 '.............DISK SAVE ROUTINE.............
4110 PRINT "NO DISK SAVE YET": GOTO 4020
4510 '
4530 CK=CH+1
4550 IF TR$="CK #":CS=GS:CH=GH:CK=GK: GOTO 1010
4555 IF TR$="DEP#":DS=GS:DH=GH:DK=GK: GOTO 1010
4560 IF TR$="PAY#":PS=GS:PH=GH:PK=GK: GOTO 1010
4565 IF TR$="CSH#":HS=GS:HH=GH:HK=GK: GOTO 1010
4570 PRINT "ERROR IN TR$ -- IT = ";TR$: STOP
5000 ' ===================================================
5001 '               REPORT GENERATING PROGRAMS
5002 '
5010 CLS
5020 PRINT "           <   REPORT MENU SELECTION   >"
5030 PRINT STRING$(60,"-")
5035 PRINT "      0 = EXIT FROM REPORT SECTION"
5040 PRINT "      1 = CHECKING ACCOUNT BALANCE"
```

```
5042 PRINT "      2 = PRINTED LIST OF ALL TRANSACTIONS"
5044 PRINT "      3 = TAX FORM REPORTS"
5050 PRINT
5055 INPUT "0, 1, 2, OR 3"; MS
5060 IF MS<0 OR MS>3 THEN 5055
5070 IF MS=0 THEN 210
5080 ON MS GOTO 6010, 7010, 8010
5090 GOTO 210
6010 GOTO5010
7000 '-------------------------------------------
7001 '   PRINTED LIST OF ALL TRANSACTIONS ENTERED
7002 '-------------------------------------------
7010 PRINT "TURN PRINTER ON -- READY";:INPUT Z$
7015 LPRINT "LIST OF ALL TRANSACTIONS FOR PERIOD";P;"OF";IY+1900"
7016 LPRINT "CHECKING ACCOUNT IS AT ";CA$;" BANK"
7017 LPRINT USING "STARTING BALANCE WAS $#####.##";BS
7025 LPRINT" "
7030 PF$="###  ##/##  $####.##  %                %"
7031 PF$=PF$+" %              % %   % ### ####"
7110 LPRINT" ":LPRINT"*** CHECKING ACCOUNT PAYMENTS ***":LPRINT" "
7120 LPRINT"CK#  DATE:  AMOUNT:   TO:                FOR";
7121 LPRINT":              FORM: TX%: ACCT:"
7130 LO=1: HI=99: S=CS: C1=1: GOSUB7510
7210 LPRINT" ":LPRINT"*** CASH PAYMENTS ***":LPRINT" "
7220 LPRINT"PAY# DATE:  AMOUNT:   TO:                FOR";
7221 LPRINT":              FORM: TX% ACCT:"
7230 LO=101: HI=149: S=PS: C1=101: GOSUB 7510
7310 LPRINT" ":LPRINT"*** CHECKING ACCOUNT DEPOSITS ***":LPRINT" "
7320 LPRINT"DEP# DATE:  AMOUNT:   FROM:              FOR";
7321 LPRINT":              FORM: TX% ACCT:"
7330 LO=151: HI=189: S=DS: C1=151: GOSUB 7510
7410 LPRINT" ":LPRINT"*** CASH RECEIPTS NOT DEPOSITED ***":LPRINT" "
7420 LPRINT"CSH# DATE:  AMOUNT:   FROM:              FOR";
7421 LPRINT":              FORM: TX% ACCT:"
7430 LO=191: HI=200: S=HS: C1=191: GOSUB 7510
7480 LPRINT" ":LPRINT "CURRENT CHECKING ACCT. BALANCE IS ";
7481 LPRINT USING "$#####.##";BC
7490 GOTO 5010
7499 '
7500 '      PRINT SUBROUTINE
7501 '
7510 FOR K=LO TO HI
7520   IF GD(K)<0 THEN 7580
7530   J=K+S-C1
7540   M=INT(GD(K)/100): D=GD(K)-M*100: A=GA(K)/100
7550   LPRINT USING PF$;J,M,D,A,GV$(K),GD$(K),GF$(K),GP(K),GC(K)
7560 NEXT K
7580 LPRINT STRING$(70,"-")
7590 RETURN
8010 GOTO5010
9000 ' ==========================================
9900 ' ------------------------------------------
9901 '              FINAL EXIT MESSAGES
9902 ' ------------------------------------------
9910 CLS: PRINT "*** END OF PERIOD";P;"TRANSACTIONS FOR";1900+IY;"***"
9915 PRINT STRING$(50,"."): PRINT
9920 PRINT USING "STARTING CHECKING ACCOUNT BALANCE WAS   $#####.##";BS
9930 PRINT USING "CLOSING CHECKING ACCOUNT BALANCE IS     $#####.##";BC
9935 PRINT
9940 PRINT "STARTING CK # THIS PERIOD WAS  ";CS
9950 PRINT "HIGHEST CK # THIS PERIOD WAS   ";CH: PRINT
9960 PRINT ">>> THE NEXT TIME THIS PROGRAM IS RUN RESPOND AS FOLLOWS:"
```

```
9965 PRINT "FOR PERIOD # ?";P+1
9970 PRINT "STARTING BALANCE THIS PERIOD ?";BC
9975 PRINT "STARTING CK # THIS PERIOD ?";CH+1 : PRINT
9980 PRINT "PROGRAM IS ABOUT TO TERMINATE.  VERIFY O.K. (Y/N)";
9985 INPUT Z$:  IF LEFT$(Z$,1)<>"Y" THEN 210
9999 END
```

## Project *TAXRPT*

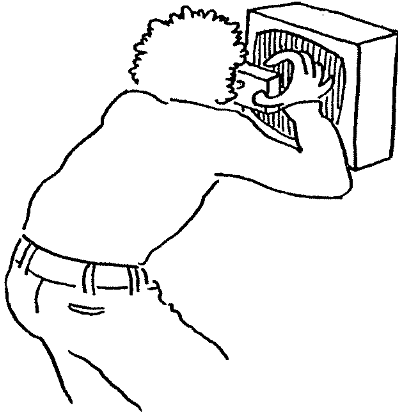
Write a program that extends MONYLIST by adding another report subprogram starting at line 8010. This report should also be done on a printer, and it should consist of summaries of all transactions that correspond to a given tax code. For example, if you have used the code CC to mean "tax form C," then the dialogue could look something like this:

```
REPORT FOR WHICH TAX FORM CODE?CC
***TRANSACTIONS APPLICABLE TO TAX FORM WITH CODE CC***

CHECK PAYMENTS MADE BY YOU:
CK #:  DATE:     AMOUNT:  TO:           FOR:          TAX AMT:  ACCT #:
123    10/22/84  42.59    AJAX BOOKS    COMPUTER BK   42.59     680
125    11/04/84  250.00   BROKEN ARMS   RENT          50.00     320
                            SUM OF FORM CC CHECK PAYMENTS = 92.59

CHECK RECEIPTS DEPOSITED BY YOU:
DP #:  DATE:     AMOUNT:  FROM:         FOR:          TAX AMT:  ACCT #:
2      11/23/84  150.00   BYTE          ARTICLE       150.00    430
                            SUM OF FORM CC CHECK RECEIPTS = 150.00

CASH PAYMENTS MADE BY YOU:
CH #:  DATE:     AMOUNT   TO:           FOR:          TAX AMT:  ACCT #:
2      9/14/84   37.50    LA FONDUE     BUS. MEAL     18.75     720
                            SUM OF FORM CC CASH PAYMENTS = 18.75
```

Of course these reports would normally be much longer. In this case, it would also be useful to break each report down by line number on a given tax form. A tax code like CC16 would mean line 16 on form C. Since the tax form codes were entered as strings, you can use the LEFT$ and MID$ functions to separate the two parts as follows:

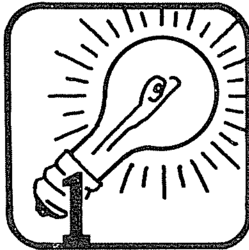
```
FORM$ = LEFT$(GF$(K), 2)
LINE$ = MID$(GF$(K),3,LEN(GF$(K)−2))
```

Other reports based on account numbers could be handled in a similar manner. With a little ingenuity, you can extend this program in many directions.



## Program SNAPSHOT

### The Idea



The preceding examples were illustrated on a TRS-80 computer, using the BASIC statement LPRINT (instead of PRINT) to direct the program's output to a printer. But suppose you want to print everything that appears on the video screen out on paper. For example, suppose you want to show the responses typed in by someone after an INPUT statement as follows:

```
10 PRINT "WHAT'S YOUR NAME";
20 INPUT A$
30 PRINT "HI";A$
RUN
WHAT'S YOUR NAME? PASCAL
HI PASCAL
```

All of the above will appear on video screen, but how do you get it on paper? To show the program part (lines 10, 20, 30) on a printer, you type LLIST on the TRS-80. To show the output, you can try changing PRINT to LPRINT, but all you'll see on paper is this:

```
WHAT'S YOUR NAME
HI PASCAL
```

The word RUN, the ?, and the user's response will be missing.

It would be a lot better in applications that use the INPUT statement if everything shown on the screen could be printed on paper. The program SNAPSHOT will do this, using a trick that involves the key word PEEK.

PEEK(L) is a BASIC function that returns the numeric value of the *contents* of any byte in memory, provided you supply the *address* L of that byte. The memory addresses in most microcomputers go from 0 to 65535. If you type

```
PRINT PEEK(15360)
```

what you'll see printed on the video screen is the numeric value of the byte at memory location 15360. If you type

```
PRINT CHR$(PEEK(15360))
```

then you'll see the ASCII character corresponding to the number stored in location 15360. Both of these examples also work with LPRINT, putting the results on the printer.

The next thing you need to know is that there are 1024 characters, including spaces, displayed on the screen of a TRS-80 computer and that the numeric codes for these are stored in memory at locations 15360 up to 16383. So if we write a program that PEEKs in each of these locations, and then LPRINTs the corresponding ASCII characters, we'll be able to print, on paper, a "snapshot" of exactly what's on the screen.

### Coding the Program

We'll skip right to the code since this is a short program. It uses line numbers starting at 32000, so that you can easily add this subroutine at the end of another main program. We've used variable names ending in 9 so that conflict with the variables in the main program will be unlikely.

The outer loop in SNAPSHOT (line 32010) runs through the 16 lines on the TRS-80 screen. The loop in 32030 to 32050 finds out how many *non-blank* characters are in each video screen line by counting backwards from the right until an ASCII code greater than 32 is found (32 is the code for space; codes 33 to 126 mean a printable character). Then the loop in 32060 to 32080 concatenates (strings together) these characters into L9$. Line 32090 prints this string on paper, and 32100 calculates the starting address for the next line of the video screen.

```
1 CLEAR 500
31999 STOP      'SNAPSHOT SUBROUTINE
32000 N9=15359
32010 FOR L9=1 TO 16
32020 L9$=""
32030 FOR W9=64 TO 1 STEP -1
32040 IF PEEK(N9+W9) > 32 THEN 32060
32050 NEXT W9
32060 FOR K9=1 TO W9
32070 L9$=L9$+CHR$(PEEK(N9+K9))
32080 NEXT K9
32090 LPRINT L9$
32100 N9=N9+64
32110 NEXT L9
32120 RETURN
```

To use SNAPSHOT, it should be added to the main program with which you want to use it. This can be done by loading the main program, and then typing in SNAPSHOT. A better way, if you have a disk system, is to first type in SNAPSHOT and then save it as an ASCII file by typing SAVE "SNAPSHOT", A. To attach it to a main program which was previously saved on disk (for example SAVE "POEM") do this:

> LOAD "POEM"

> MERGE "SNAPSHOT"

If you now type LIST, you'll see that you have both programs merged.

The last thing to do is put the statement GOSUB 32000 at those places in the main program where you want to capture video output on paper. If there are several places, use GOSUB 32000: CLS so that the screen gets cleared after each snapshot is taken. Since SNAPSHOT stores quite a few

characters in the variable L9$, it is also necessary to put the statement 1 CLEAR 500 at the beginning of the program, which sets aside extra space needed for the strings.

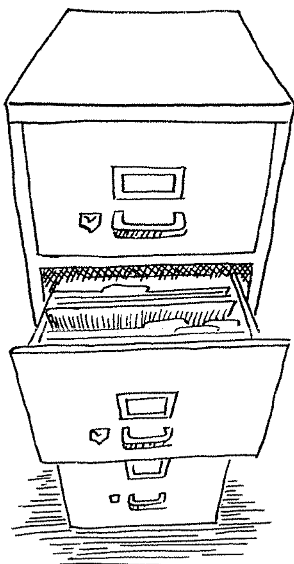
All the sample runs in this book that show responses to INPUT statements were printed by using SNAPSHOT. Here's an example showing how we used it to display what the video screen looked like for Project *POEM*. (To see a run of this program, see section 1.7.)

```
1 CLEAR500
10 REM --- POEM ---
15 CLS
20 PRINT"WITH YOUR HELP, THIS PROGRAM WILL"
30 PRINT"WRITE AN 'ORIGINAL' POEM."
40 PRINT
50 PRINT"WHAT'S A GOOD WORD FOR A PLACE--"
60 PRINT"LIKE DESERT, MOON, FOREST, ALLEY, ETC.   ";
65 INPUT P$
70 PRINT"THANKS."
80 PRINT"GIVE ME A PLURAL NOUN FOR SOME"
85 PRINT"THINGS THAT BELONG ON LAND.              ";
86 INPUT Q$
90 PRINT"THANKS."
100 PRINT"NOW I NEED A PLURAL NOUN FOR SOME"
110 PRINT"THINGS THAT LIVE IN THE SEA.             ";
115 INPUT R$
120 PRINT"THANKS."
130 PRINT
135 GOSUB 32000:CLS
150 PRINT"          A RIDDLE"
160 PRINT
170 PRINT"THE MAN IN THE ";P$;" ASKED OF ME,"
180 PRINT"HOW MANY ";Q$;" GROW IN THE SEA?"
190 PRINT"I ANSWERED HIM AS I THOUGHT GOOD,"
200 PRINT"AS MANY AS ";R$;" GROW IN THE WOOD."
205 PRINT
206 PRINT
210 PRINT"WANT TO PRINT IT AGAIN (Y = YES)";
220 A$ = " "
225 INPUT A$
226 IF A$ = "Y" THEN 130
230 PRINT"WANT TO WRITE ANOTHER (Y = YES)";
235 A$ = " "
240 INPUT A$
245 GOSUB 32000
250 END
32000 N9=15359
32010 FOR L9=1 TO 16
32020 L9$=""
32030 FOR W9=64 TO 1 STEP -1
32040 IF PEEK(N9+W9) > 32 THEN 32060
32050 NEXT W9
32060 FOR K9=1 TO W9
32070 L9$=L9$+CHR$(PEEK(N9+K9))
32080 NEXT K9
32090 LPRINT L9$
32100 N9=N9+64
32110 NEXT L9
32115 RETURN
```

*These cause SNAPSHOT to transfer all of the preceding dialogue from screen to printer (two screenfulls in this example).*

*This is the snapshot subroutine (see also line 1).*

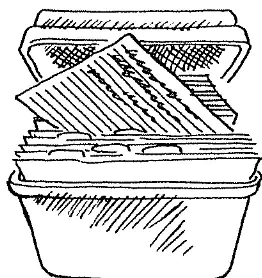
## 4.3 ADDING A DISK SYSTEM: FILEBOX, FILEDEMO, *FILEBOX2, FILEBOX3,* HASHDEMO, *(HASHFILE), (MONEYDISK)*



Of all the ways to extend and improve a personal computer, adding disk storage (along with system software called a disk operating system) should rank high on anyone's priority list. It's an addition that's guaranteed to make computing more fascinating, opening up areas of application equal to anything done by the "pros."

The cost of adding a disk system has lessened quite a bit in recent years, and it's comparable to the cost of adding a printer. However, unlike learning to use a printer, learning to use a disk system to its full potential will take time. Mastering all the features of a good disk system is analogous to learning to use all the power of BASIC. In both cases there's a forest of technical foliage to be cut away before one can get to the "good stuff."

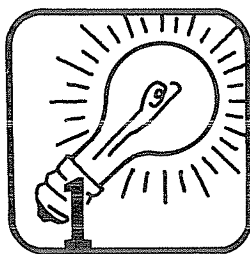
In this section we'll try to get to the good stuff rather quickly, while advising you not to be afraid to go back and beat the path down again and again until it feels comfortable. Our approach will be based on a human-oriented description of the use of *disk files* for *data storage.* As will be seen in subsequent sections, this use of disks is at the heart of most modern information and data processing applications.



The word *file* refers to a collection of pieces of information called *records,* where each record is made up of individual data *items.* For example, a recipe file is a collection of records, each one of which contains several items that tell a cook how to prepare a dish. A library card catalog is a file in which each record is also a card. In this case a card record contains items such as catalog number, book title, author, publisher, and so on. An insurance company file could be a collection of records in the form of manila folders, each one containing such items as customer account number, name, address, medical history, and insurance policies issued.

### Program FILEBOX

#### The Idea



The idea behind the program FILEBOX is to use a magnetic disk instead of paper for storing data files. The computer will be used for quickly storing records of data items on this disk file, for quickly retrieving selected portions of these records, and for changing and updating records.

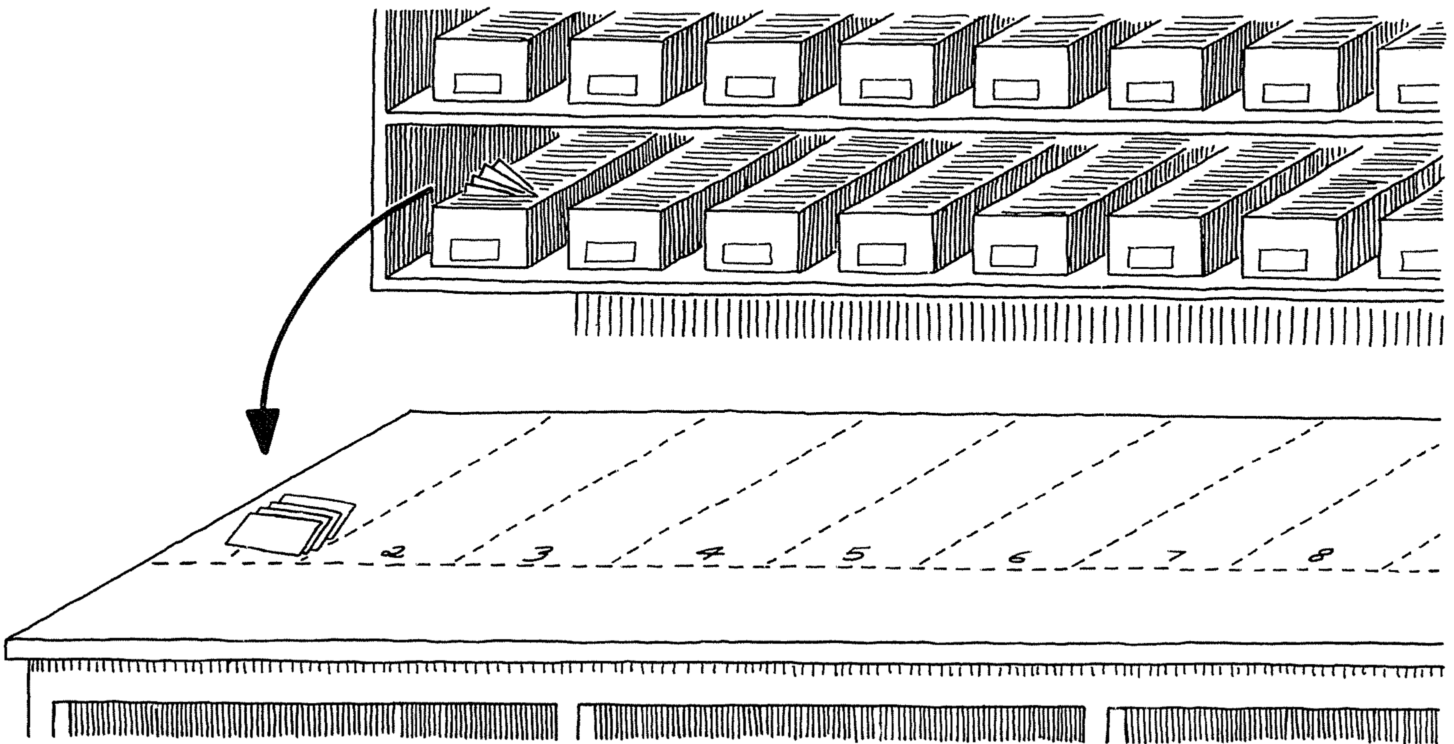
A good way to describe what we want the program to do is to compare its operation with a manual file card system. We can see that there are seven main operations to be considered:

## MANUAL FILE CARD SYSTEM

1. Select (by name) one of the file boxes sitting on a shelf above your desk. Set it on the desk in one of several *numbered work places* reserved for this purpose.

## COMPUTER DISK FILE SYSTEM

1. Select (by name) one of the files stored on the computer's magnetic disk. Associate it with a *channel/buffer number* in the computer by using the OPEN statement.




2. If this is a new file, make sure there are nice clean cards in the box, and number them, say from 1 to 25.

3. Decide how many items you want to put on each card, and in what position they should go.

4. Choose a card and pull it from the box. Hold it close enough to see what's on it.

5. If it's blank, or if it contains outdated information, write new data on it if you wish.

6. To make later access of records by number easier, hire a clerk to type a list of all the reference numbers, followed by the key data item associated with each number.

2. If this is a new disk file, magnetically record *initial* data on it, associating each record in the file with a *physical record number* called PR%.

3. Use the FIELD statement to describe the positions of data items in each record, and how large they can be.

4. Use the GET statement to bring a record from disk into main memory. Print it on the video display to see what's on it.

5. If the record is marked "blank," or if it contains outdated information, use the PUT statement to write new data on the disk record if you wish.

6. To make access of records easier, add a routine to your program which can print a list of all the record numbers, followed by the key data item associated with each number.

7. When finished, put all cards back and return the box to its shelf.

7. Use the CLOSE statement to "detach" the disk file from your program.

We'll return to these seven steps in the high-level design, but let's first see what a run of a computer version of FILEBOX looks like.

### A Sample Run

We've decided to put three items on each record in this program: a key word, information related to the key, and the number of times the record has been accessed. For example, the information on record #14 might be pictured as follows:

Record #14

| CIGARS | MANILLA ROPE & TOBACCO, 487-2315 | 5 |
|---|---|---|

The 5 at the end means that this record has been retrieved 5 times since it was first put on file. In our program, this record would be referenced by letting PR% = 14

In the first sample run, a file called MEMO is used to hold these records. At this time it is a new file, so the program initializes it with blanks for the data, and a 0 (zero) for "number of times accessed." Then the user stores new data in some of its records (these can be chosen in random order). Each time the user decides to look at any of these newly created records, the access number is increased by 1. If the data is changed on an old record, the access number is set back to 1.

```
THIS PROGRAM ALLOWS YOU TO SAVE AND/OR RETRIEVE PAIRS
OF DATA OF THE FORM (KEY, INFO).      EXAMPLES:
KEY? SAUERBRATEN   INFO? OLD HEIDELBERG, 555-1234
KEY? ACE BONDS     INFO? SAFE DEP BOX 24, MARGINAL TRUST CO.

NAME OF DATA FILE? MEMO
>>> WARNING: DO NOT EXIT THIS PROGRAM BY PRESSING BREAK <<<
THIS IS A NEW FILE.  STAND BY FOR INITIALIZATION

-------------------------------------------------
RECORD # (0 = EXIT)? 1
* RECORD # 1 IS BLANK *
......... DO YOU WISH TO .........
CHANGE(C), DELETE(D), OR RESUME(R)? C
TYPE NEW DATA FOR RECORD # 1
KEY : SAUERBRATEN
INFO: ALPINE VILLAGE, 555-9876, MUSIC AFTER 8
```

```
------------------------------------------------
RECORD # (0 = EXIT)? 3
* RECORD # 3 IS BLANK *
......... DO YOU WISH TO .........
CHANGE(C), DELETE(D), OR RESUME(R)? C
TYPE NEW DATA FOR RECORD # 3
KEY : VEGETARIAN
INFO: CORNUCOPIA REST., 334 ATWOOD ST., 234-9987
------------------------------------------------
RECORD # (0 = EXIT)? 1
DATA FOR RECORD # 1
KEY  : SAUERBRATEN
INFO : ALPINE VILLAGE, 555-9876, MUSIC AFTER 8
THIS RECORD HAS BEEN ACCESSED 1 TIME(S)
......... DO YOU WISH TO .........
CHANGE(C), DELETE(D), OR RESUME(R)? R
------------------------------------------------
RECORD # (0 = EXIT)? 5
* RECORD # 5 IS BLANK *
......... DO YOU WISH TO .........
CHANGE(C), DELETE(D), OR RESUME(R)? C
TYPE NEW DATA FOR RECORD # 5
KEY : AAA
INFO: TOWING SERVICE AT 555-4567
------------------------------------------------
RECORD # (0 = EXIT)? 0
QUIT(Q), PRINT KEYS(P), OR RESUME(R)? P
PRINT ROUTINE MISSING AT PRESENT
QUIT(Q), PRINT KEYS(P), OR RESUME(R)? Q
* END OF PROGRAM -- ALL FILES ARE CLOSED *
```

The next run shows what happens when this program is used at a later date. The file MEMO is now an "old" one, so it is not initialized. Previously stored information can be retrieved or changed. And of course new information can still be stored on the unused records of the file.

```
THIS PROGRAM ALLOWS YOU TO SAVE AND/OR RETRIEVE PAIRS
OF DATA OF THE FORM (KEY, INFO).      EXAMPLES:
KEY? SAUERBRATEN  INFO? OLD HEIDELBERG, 555-1234
KEY? ACE BONDS    INFO? SAFE DEP BOX 24, MARGINAL TRUST CO.

NAME OF DATA FILE? MEMO
>>> WARNING: DO NOT EXIT THIS PROGRAM BY PRESSING BREAK <<<
NOTE: THIS IS AN OLD FILE
```

```
-------------------------------------------------
RECORD # (0 = EXIT)? 1
DATA FOR RECORD # 1
KEY  : SAUERBRATEN
INFO : ALPINE VILLAGE, 555-9876, MUSIC AFTER 8
THIS RECORD HAS BEEN ACCESSED 2 TIME(S)
......... DO YOU WISH TO .........
CHANGE(C), DELETE(D), OR RESUME(R)? R
-------------------------------------------------
RECORD # (0 = EXIT)? 3
DATA FOR RECORD # 3
KEY  : VEGETARIAN
INFO : CORNUCOPIA REST., 334 ATWOOD ST., 234-9987
THIS RECORD HAS BEEN ACCESSED 1 TIME(S)
......... DO YOU WISH TO .........
CHANGE(C), DELETE(D), OR RESUME(R)? D
-------------------------------------------------
RECORD # (0 = EXIT)? 3
* RECORD # 3 IS BLANK *
......... DO YOU WISH TO .........
CHANGE(C), DELETE(D), OR RESUME(R)? R
-------------------------------------------------
RECORD # (0 = EXIT)? 1
DATA FOR RECORD # 1
KEY  : SAUERBRATEN
INFO : ALPINE VILLAGE, 555-9876, MUSIC AFTER 8
THIS RECORD HAS BEEN ACCESSED 3 TIME(S)
......... DO YOU WISH TO .........
CHANGE(C), DELETE(D), OR RESUME(R)? X
CHANGE(C), DELETE(D), OR RESUME(R)? R
-------------------------------------------------
RECORD # (0 = EXIT)? 0
QUIT(Q), PRINT KEYS(P), OR RESUME(R)? Q
* END OF PROGRAM -- ALL FILES ARE CLOSED *
```

### High-Level Design

The seven steps given in our earlier comparison of a manual filing system with computer files can also be used for the design of FILEBOX. These can be restated more fully as follows:

1. Print instructions, input name of data file, and open it for use with a *channel/buffer number* from 1 to 15. This number will be used in the rest of the program to refer to this file. This is like using a channel number in a TV set to refer to one of the thousands of possible station names. Just as channel 2 can refer to WCBS in one place, but to WXYZ somewhere else, channel/buffer 2 can refer to the file MEMO at one time, but to STARTREK at another. You determine the correspondence, using the OPEN statement explained below.

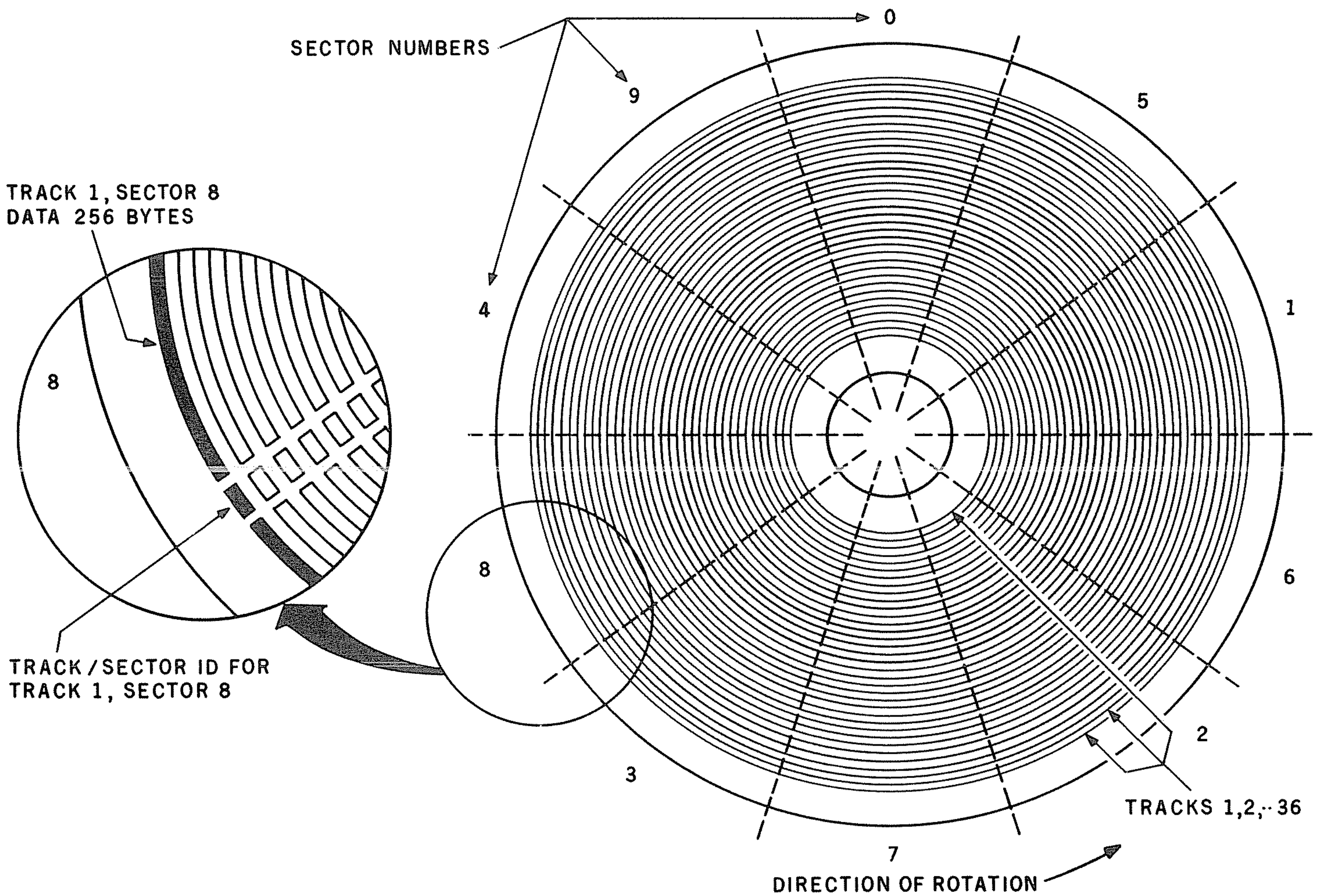
2. If the file is new, initialize it with blank records and set the access numbers to zero.
3. Ask the user which record number is desired. Then take care of a number of technical matters:
   (a) Convert the record number supplied by the user into a physical record number. In FILEBOX the physical record number PR% will simply be made equal to the user's number.
   (b) Use the FIELD statement to arrange the file *buffer area* in a way that matches the way you want data items stored in each record.
4. GET the desired record from the disk file, and display its contents on the screen.
5. Allow the user to PUT new or revised data on the record just displayed, if desired. Also allow the user to delete data.
6. Repeat steps 3, 4, and 5 until user says "exit."
7. Clean up time. Print a list of keys if desired; close the file.

### Coding the Program

Before presenting the code for FILEBOX, we'll explain six new statements found in Microsoft disk-extended BASIC: OPEN, FIELD, LSET, GET, PUT, and CLOSE. Our explanations will correspond to the way these statements are used on a TRS-80 computer using TRSDOS (the TRS-80 Disk Operating System). We'll also explain use of the four special BASIC functions MKS, MKI, CVS, and CVI.

### How Disk Files Are Accessed

A mini-disk can hold about 90,000 bytes of information, where each byte can represent one character (like A, B, Z, 3, =, +, and so on). The information is recorded on 35 concentric tracks, with each track broken down into 10 sectors. Each sector holds 256 bytes, so you can see the actual total is 10*256*35 = 89,600 bytes. One sector is used for each physical record in a data file, so that means 256 characters can be stored on a physical record. However, since one byte is not available to the user, from now on we'll use the number 255. You should also be aware that some of the tracks on a disk are reserved for system use, so the space available for user files will be less than the disk's maximum capacity.



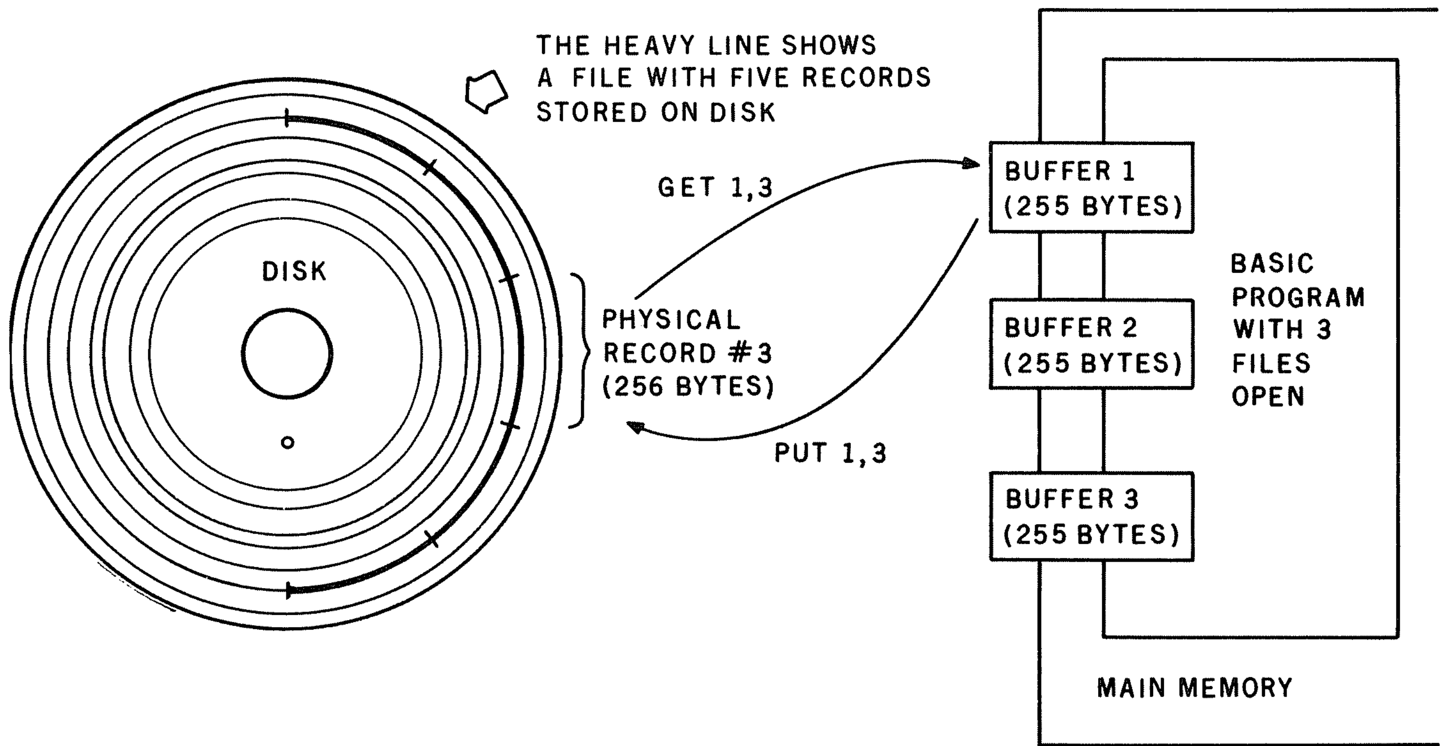

Disk-extended BASIC allows you to move exactly one record at a time from disk to user memory, and vice versa. The record is stored in a portion of memory called a disk I/O *buffer*. There can be up to fifteen of these buffers, and each one holds 255 bytes. The BASIC key words GET and PUT are used to move things from disk to buffer and vice versa, while the key word OPEN is used to say which buffer will be used for a given file.

*An Example:* Suppose the file MEMO has five physical records stored on a disk. You decide to work on some of these records at random using disk I/O buffer 1. So you need the BASIC statement

```
110 OPEN "R", 1, "MEMO"
```

The "R" means that this file will be used randomly. You can ask for any record in any order—you don't have to go through the records sequentially, as with tape.

Here's a picture of what happens when you transfer a record, (for example, #3) from disk and store it in buffer 1, and vice versa:




To make the transfer of record #3 from disk to memory, your program must use a statement like this:

```
310 GET 1,3
```

To reverse the operation, transferring what's in the buffer back to disk (presumably because you've made some changes in the data), your program needs a statement like this:

```
410 PUT 1, 3
```

So far so good. However, once the data for the desired record has been moved into the buffer, the big question is how does your program get at it? The answer is that you use the FIELD statement to set up *pointer names*. These look like string variable names (such as A$, D$, K$(I) ), but they refer only to the contents of the fielded buffer. For example,

```
210 FIELD 1, 11 AS A$, 50 AS B$, 2 AS C$
```

can be pictured as doing the following:

| Buffer 1: | A$ 11 bytes | B$ 50 bytes | C$ 2 bytes | 192 bytes not made accessible |
|---|---|---|---|---|

If you next use the statements

```
410 GET 1, 3
420 PRINT A$, B$, C$
```

you will then see what's in the third record at the parts of the buffer these names point to, with 11, 50, and 2 bytes printed for A$, B$, and C$, respectively.

The reverse procedure is a little trickier. For example, suppose you want to store an 11-character word, followed by a 50-character definition,

followed by a 2-letter code, putting all three items on record #4 of the file DATA opened for use with buffer 1. To do this, you must first field the buffer, and then load it up with data by using LSET statements as follows:

```
150 FIELD 1, 11 AS A$, 50 AS B$, 2 AS C$
160 LSET A$ = "PELICAN"
170 LSET B$ = "HIS BILL HOLDS MORE THAN HIS
               BELLYCAN"
180 LSET C$ = "XX"
190 PUT 1, 4
```

LSET is a special *buffer assignment* statement that must be used when placing data into a buffer. It means "left set," that is, place the left-most part of the string given in each statement at the position specified by the FIELD statement, using exactly the number of bytes specified. If any bytes are unused, space characters (ASCII 32) are inserted as needed. If there aren't enough bytes, the string will be truncated, ie: chopped off, on the right.

### Saving and Retrieving Numbers

The use of the dollar sign ($) in buffer pointer names is to remind you that only string (ASCII character) information can be stored on disk data files. To store numbers, you must first make them into a string, using the special MKS$( ) and MKI$( ) functions. When these strings are later retrieved from disk, they are converted into numbers with the CVS( ) and CVI( ) functions. The S in these function names means "single precision decimal number" (like 3.14159), while the I means "integer number" between −32,768 and 32,767, inclusive. It turns out that the single precision numbers require four bytes when stored as strings, while integer numbers take only two bytes.

For example, suppose we wish to store an integer number on record #4 of the file DATA in the position pointed at by C$. The number is to start out as 0, and increase by 1 every time the record is accessed by a GET statement. Here's a BASIC program that does this, printing out the contents of the record each time it's updated. You can see what was PUT into record #4 each time around the loop by drawing a box around all the output to the right of the = signs.

```
100 REM ----- FILEDEMO -----
110 OPEN "R", 1, "DATA"             'USE BUFFER 1 FOR THE RANDOM FILE "DATA"
120 S$="BEGIN: "                    'INITIALIZE THE STRING S$
130 FIELD 1, 25 AS T$, 2 AS C$      'DEFINE 2 POINTER POSITIONS FOR BUFFER 1
140 LSET T$=S$                      'PLACE S$ IN  BUFFER  AT POSITION T$
150 LSET C$=MKI$(0)                 'MAKE 0 INTO A STRING AND PLACE AT C$
155 LPRINT TAB(25);"T$";TAB(51);"C$"
156 LPRINT TAB(25); "/";TAB(51);"/"
160 FOR C%=1 TO 10                  '========== BEGIN LOOP ==========
170   PUT 1, 4                      'PUT CONTENTS OF BUFFER ON DISK RCD #4
180   GET 1, 4                      'BRING DISK RECORD #4 BACK INTO BUFFER
190   LPRINT "DATA ON RECORD #4 ";   'PRINT OUT T$ PART OF BUFFER AS A
195   LPRINT "NOW =  ";T$;CVI(C$)    'STRING AND C$ PART AS AN INTEGER
200   S$=S$+CHR$(64+C%)             'ADD AN ALPHABETIC CHARACTER TO S$
210   LSET C$=MKI$(C%)              'MAKE C% INTO A STRING AND PUT AT C$
220   LSET T$=S$                    'PLACE LATEST S$ IN BUFFER AT POSITION T$
230 NEXT C%                         '=========== END LOOP ===========
240 CLOSE: END                      'CLOSE ALL FILES; DISCONNECT BUFFERS

>RUN

                          T$                          C$
                          /                           /
DATA ON RECORD #4 NOW =  BEGIN:                       0
DATA ON RECORD #4 NOW =  BEGIN: A                     1
DATA ON RECORD #4 NOW =  BEGIN: AB                    2
DATA ON RECORD #4 NOW =  BEGIN: ABC                   3
DATA ON RECORD #4 NOW =  BEGIN: ABCD                  4
DATA ON RECORD #4 NOW =  BEGIN: ABCDE                 5
DATA ON RECORD #4 NOW =  BEGIN: ABCDEF                6
DATA ON RECORD #4 NOW =  BEGIN: ABCDEFG               7
DATA ON RECORD #4 NOW =  BEGIN: ABCDEFGH              8
DATA ON RECORD #4 NOW =  BEGIN: ABCDEFGHI             9
```

*This is the data that was put on record #4 (and retrieved using GET) in filedemo*

The preceding is a complete, but not very useful program that demonstrates the most important features of disk BASIC. Notice that the last statement is CLOSE (or CLOSE 1). The CLOSE statement should always be at the end of a program that uses disk files. It should also be used any place an opened file will not be accessed for a while.

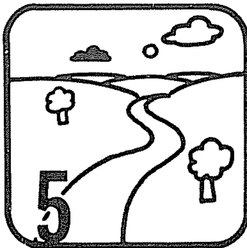
WARNING!!! If a file program you are testing gets interrupted because of an error or a BREAK, the CLOSE statement won't be executed. In this case you should immediately type "CLOSE" in direct mode.

Let's now look at the program FILEBOX which uses all the techniques just discussed, but in a more useful application. The name of the data file

is supplied by the user in line 170 and stored in F$. Line 180 connects this file to buffer 1. Line 190 uses the special "last on file" function LOF(1) to tell us the number of the last physical record recorded on the buffer 1 file. If LOF(1) = 0 it's a new file that needs to be initialized; otherwise it's an old file with previously recorded data.

```
10 '*******************************************************
20 '*    FILEBOX (SAVES AND RETRIEVES DATA USING DISKS)     *
30 '*******************************************************
110 CLEAR 500 : CLS
115 ON ERROR GOTO 989
120 PRINT "THIS PROGRAM ALLOWS YOU TO SAVE AND/OR RETRIEVE PAIRS"
130 PRINT "OF DATA OF THE FORM (KEY, INFO).      EXAMPLES:"
150 PRINT "KEY? SAUERBRATEN  INFO? OLD HEIDELBERG, 555-1234"
160 PRINT "KEY? ACE BONDS    INFO? SAFE DEP BOX 24, MARGINAL TRUST CO."
170 PRINT:PRINT"NAME OF DATA FILE";:INPUT F$
172 IF F$="" THEN 999
175 IF LEN(F$)>8 OR ASC(F$)<65 OR ASC(F$)>90 PRINT "BAD NAME":GOTO 170
177 PRINT ">>> WARNING: DO NOT EXIT THIS PROGRAM BY PRESSING BREAK <<<"
180 OPEN "R", 1, F$
190 IF LOF(1)>0 PRINT "NOTE: THIS IS AN OLD FILE": GOTO 310
195 PRINT "THIS IS A NEW FILE.  STAND BY FOR INITIALIZATION"
200 '---------------------------------------------
201 '       INITIALIZE 25 PHYSICAL DISK RECORDS
202 '---------------------------------------------
210 C%=0
216 FIELD 1, 61 AS T$, 2 AS C$
217 LSET T$=" ": LSET C$=MKI$(C%)
220 FOR PR%=1 TO 25
230   PUT 1, PR%
240 NEXT PR%
300 '---------------------------------------------
301 '   DETERMINE RECORD #'S;  FIELD THE I/O BUFFER
302 '---------------------------------------------
310 PRINT "-----------------------------------------------"
320 PRINT "RECORD # (0 = EXIT)";: INPUT KN%
330 IF KN%=0 THEN 910
350 GOSUB 810     'CALCULATE PHYSICAL RECORD # & SUBRECORD #
360 IF PR%>25 PRINT "FILE SIZE EXCEEDED": GOTO 310
390 FIELD 1, 11 AS A$, 50 AS B$, 2 AS C$
400 '---------------------------------------------
401 '     RETRIEVE DATA FROM DISK AND DISPLAY IT
402 '---------------------------------------------
410 GET 1, PR%
415 C%=CVI(C$): IF C%<>0 THEN 420
417   PRINT"* RECORD #";KN%;"IS BLANK *": GOTO 455
420 PRINT "DATA FOR RECORD #"; KN%
430 PRINT "KEY  : ";A$
440 PRINT "INFO : ";B$
450 PRINT "THIS RECORD HAS BEEN ACCESSED";C%;"TIME(S)":C%=C%+1
451 LSET C$=MKI$(C%)
452 PUT 1, PR%          'PUT NEW # OF ACCESSES ON RECORD
455 PRINT "......... DO YOU WISH TO ........."
460 PRINT"CHANGE(C), DELETE(D), OR RESUME(R)";:R$="":INPUT R$
461 IF R$="R" OR R$="" THEN 310
462 IF R$="C" THEN 510
463 IF R$="D" LSET A$=" ":LSET B$=" ":C%=0:GOTO 540
464 GOTO 460
500 '---------------------------------------------
```

```
501 '         INPUT NEW DATA AND SAVE IT ON DISK
502 '-------------------------------------------------
510 C%=1: PRINT "TYPE NEW DATA FOR RECORD #";KN%
520 PRINT "KEY : ";:LINEINPUT K$ : LSET A$=K$
530 PRINT "INFO: ";:LINEINPUT I$ : LSET B$=I$
540 LSET C$=MKI$(C%)
550 PUT 1, PR%
590 GOTO 310
800 '-------------------------------------------------
801 '    CALCULATE PHYSICAL RECORD # AND SUBRECORD #
802 '-------------------------------------------------
810 PR%=KN%      'PR IS SAME AS NUMBER INPUT BY USER IN
820              'THIS SIMPLIFIED VERSION OF FILEBOX
890 RETURN
900 '-------------------------------------------------
901 '  EXIT ROUTINE; LIST ALL KEYS ON FILE IF DESIRED
902 '-------------------------------------------------
910 PRINT "QUIT(Q), PRINT KEYS(P), OR RESUME(R)";: INPUT R$
915 IF R$="Q" THEN 990
920 IF R$="P" THEN 940
925 IF R$="R" THEN 310
930 GOTO 910
940 PRINT"PRINT ROUTINE MISSING AT PRESENT": GOTO 910
989 PRINT "ERROR DETECTED IN PROGRAM"
990 CLOSE
999 PRINT "* END OF PROGRAM -- ALL FILES ARE CLOSED *":END
```




### Future Extensions and Revisions

The program FILEBOX is wasteful of disk space since it only uses 63 bytes for each user record, while a disk physical record can hold up to 255 bytes. This suggests storing several user records in each physical record (PR%). With 63 bytes per user record, we can put four user records in each physical record. Some books call the user record a *logical record* because its *number* is in the user's head, not in the physical machine. If we give each of the user logical records a key number, KN% = 1, 2, 3, 4, ..., etc., and divide each physical record PR% into four subrecords labelled SR% = 0, 1, 2, 3, here's a picture showing how we can store four times as much information in a file:

| | SR% = 0 | SR% = 1 | SR% = 2 | SR% = 3 |
|---|---|---|---|---|
| PR% = 1 | KN% = 1 | KN% = 2 | KN% = 3 | KN% = 4 |
| PR% = 2 | KN% = 5 | KN% = 6 | KN% = 7 | KN% = 8 |
| PR% = 3 | KN% = 9 | KN% = 10 | KN% = 11 | KN% = 12 |
| . | | | | |
| . | | | | |
| . | | | | |
| etc. | | etc. | | |
| . | | | | |
| . | | | | |
| . | | | | |

Diagram showing a file layout with 4 subrecords on each physical record.

All the variables have had the percent sign (%) added to indicate that BASIC should treat these as integers. In this picture, it is understood that each of the logical records, (the small boxes with the key numbers KN% inside,) is analogous to one file card of information. The data stored in the logical records will be in the form described by the user with the FIELD statement.

### Project *FILEBOX2*

Modify FILEBOX so that each logical record still consists of 11 bytes for the key, 50 bytes for the information, and 2 bytes for the number of times accessed, but so that there will be four logical records stored in each physical record. The trick will be to field the 255-byte buffer as follows:

SR% = 0 SR% = 1 SR % = 3 SR % = 4 Not used

We could field the buffer into this form by using a very long FIELD statement, using a lot of pointer names (12 in our example). But then we'd have the problem of figuring out which pointer names go with which subrecord. One solution to this problem is to use *subscripted* pointer names. For example, the three pointers for subrecord #0 could be called A$(0), B$(0), and C$(0). Then those for subrecord #1 would be called A$(1), B$(1), C$(1), those for subrecord #2, A$(2), B$(2), C$(2), and so on. Here's how buffer 1 can be fielded using this scheme for four subrecords:

```
210 FOR SR% = 0 to 3
220 FIELD 1, (SR% * 63) AS D$, 11 AS A$(SR%), 50 AS
B$(SR%), 2 AS C$(SR%)
230 NEXT SR%
```

The trick is to use a dummy pointer D$ which successively forces the field statement to skip over 0 positions, 63 positions, 126 positions, and 189 positions of the buffer before setting the *real* pointers A$( ), B$( ), and C$( ).

In many applications there is no need to use subscripted pointers since the program deals with only one logical record at a time. This is the case with *FILEBOX2.* Once the user inputs the key number, this can immediately be translated into unique physical record, and subrecord numbers. In this program, we refer to each logical record by the key number KN%, which is input by the user. The physical record and subrecord numbers are calculated, in subroutine 800 by the formulas:

```
PR% = INT((KN% − 1)/4) + 1
SR% = KN% − 4 * (PR% − 1) −1
```

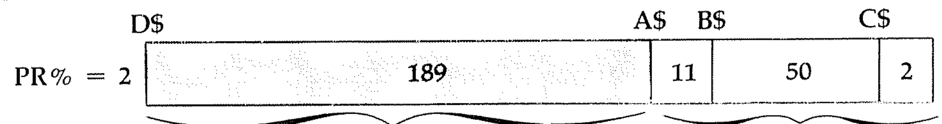
Then the buffer pointers for this subrecord should be set by using the statement

```
FIELD 1, (SR% * 63) AS D$, 11 AS A$, 50 AS B$, 2 AS C$
```

To see how these formulas work, try them out for the key numbers 1, 2, 3, 4, 5, ..., 12 and then compare what you get with the file layout diagram shown earlier. For example, for KN% = 8 you should get:

```
PR% = INT(7/14) + 1 = 1 + 1 = 2
SR% = 8 − 4 * (2 − 1) − 1 = 8 − 4 − 1 = 3
```

Thus when the user gives a key number KN% = 8, which means he wants the 8th logical record in the file, these formulas say that logical record 8 is located on physical record #2, but in the fourth subrecord position (SR% = 3). The field statement locates this logical record by setting up the pointers as follows:




For SR% = 3, the dummy pointer D$ is fielded as 3* 63 = 189.

So here's the fourth subrecord, which is where logical record 8 is stored.

What this all comes down to is that when the user asks for logical record 8 (KN% = 8), the program must do three things:

1. Calculate PR% = 2
2. Calculate SR% = 3
3. Execute the field statement
   FIELD SR% * 63 AS D$, 11 AS A$, 50 AS B$, 2 AS C$
   which for this case is the same as
   FIELD 189 AS D$, 11 AS A$, 50 AS B$, 2 AS C$

These three actions must be taken for every new value of KN% supplied by the user, so they must be handled by statements that are executed

each time around the main loop of the program (lines 310 to 590).

For *FILEBOX2*, use all the above ideas to modify FILEBOX so that it can store a total of 100 logical (user) records using only 25 physical records.

### Project *FILEBOX3*

To use the previous two programs the user must supply the key number (KN%) of the logical record desired. It may be difficult to remember these numbers. To help out someone who'd like a memory jogger, modify *FILEBOX2* so that it produces a printed list of the key numbers KN% followed by the key data item. Here's what the printout should look like:

```
  #: KEY            #: KEY            #: KEY            #: KEY
-----------------------------------------------------------------
  1: SAUERBRATEN    2:                3:                4:
  5: AAA            6: CIGARS         7:                8: SUSIE
  9:               10:               11:               12:
 13: FISH          14: LOANS         15: INSURANCE     16:
 17:               18:               19:               20:
 21:               22:               23:               24:
 25:               26:               27:               28:
 29:               30:               31:               32:
 33:               34: POKER         35:               36:
 37:               38:               39:               40:
 41:               42:               43:               44:
 45:               46:               47:               48:
 49:               50:               51:               52:
 53:               54:               55:               56:
 57:               58:               59:               60:
 61:               62:               63:               64:
 65:               66:               67:               68:
 69:               70:               71:               72:
 73:               74:               75:               76:
 77:               78: CONCERTS      79:               80:
 81:               82:               83:               84:
 85:               86:               87:               88:
 89:               90:               91:               92:
 93:               94:               95:               96:
 97:               98: BUS SKED      99: TAXI         100:
```

### Beyond *FILEBOX3*; Hash Coding

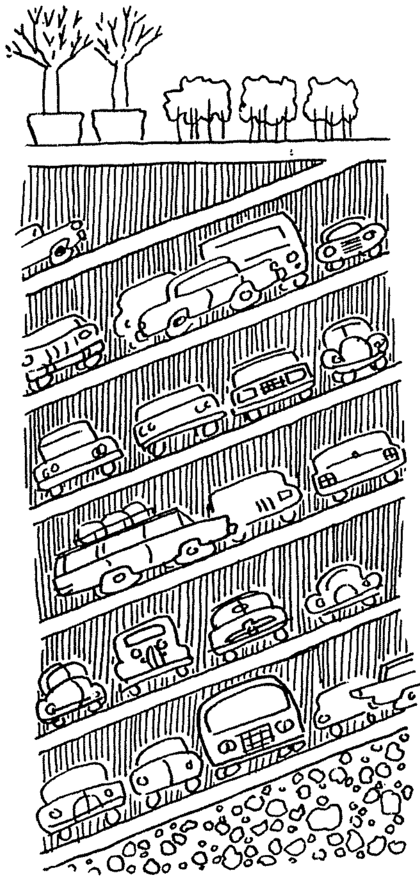
In many applications it's desirable to use more complicated key numbers. For example, a business data system might use 7-digit account numbers (like AN% = 9645234). The problem is that these numbers are too big for use as key numbers. It's out of the question to reserve one logical record for each possible account number (there are millions of them!).

One technique for getting around this difficulty is called *hash coding*.

The idea is to convert a large number like 9645234 into a simple key number KN% that takes on the values 1, 2, 3, 4, 5, ..., and so on, up to the maximum size of your file. A typical hash coding procedure has two parts:

1. Generate a small number KN% from the big number AN%, input by the user, or from a key word input by the user.
2. Make sure KN% hasn't already been used (this is called detecting *collisions*). If it is, set KN% equal to the next unused number.

Step 1 of this procedure can be explained by comparing it to a parking garage which has the innovative feature that when you enter, the attendant converts your license plate number (AN%) into a smaller parking stall number (KN%). The conversion is done by a formula that says "divide AN% by a prime number just a little less than the number of parking stalls, and set KN% equal to 1 + the remainder." For example, if your license plate number is 6358 and there are 100 stalls, we could use the rule "KN% = remainder + 1" after dividing by 97. This will make KN% come out to be 54, since



$$\begin{array}{r} 65 \\ 97\overline{)6358} \\ \underline{582} \\ 538 \\ \underline{485} \\ 53 \end{array}$$

and remainder + 1 = 54.

So you are told to park in stall #54. The reason for the +1 in the formula is that we don't want to have a stall #0.

Step 2 at this garage says, "If there's already a car parked in stall 54, try stalls 55, 56, and so on until you find an empty one." There are better rules that could be used, but this one has the virtue of simplicity. It's also understood that if you get to stall #100 and find it occupied, you continue your search back at stall #1, 2, ..., and so on. If after 100 tries you strike out, there's obviously no room!

Now when you come back to retrieve your car, suppose you find that you forgot your stall number, but you can see your license plate number is 6358. The attendant quickly uses his computer to "hash" this into 53 + 1 = 54 and tells you that you will probably find your car in stall 54, but if it's not there, you should find it with just a small sequential search of stalls 55, 56, and so on.

In our example, we divided by 97 because it is a prime number. It has been found that using a prime divisor "scatters" the data around a lot better. However, it would be a bad idea to try and squeeze 97 cars into a 97-stall garage. There should be more stalls than cars to avoid having too many collisions in a row. In our example we've assumed 100 stalls, corresponding to the number of records in *FILEBOX3*. For more on the subject of hashing, see the BYTE Book, *Program Design*.

The same ideas can be applied to key items expressed as strings. For example, we can take the key items used in FILEBOX and hash them into key numbers KN%, say from 1 to 97.

To do this, we must first change a key string, say CIGARS, into a number. One way is to convert each of the letters in CIGARS into its ASCII code, and then find the sum of these codes. To make the same letter yield a different number in different words, we can multiply the code by the position in the word. Using this scheme, the word BOB would become:

| Letter | ASCII Code | Position | Product |
|---|---|---|---|
| B | 66 | 1 | 66 |
| O | 79 | 2 | 158 |
| B | 66 | 3 | 198 |
| | | AN% = SUM = | 422 |

Next we use the divison by 97 trick, and make KN% = the remainder plus one. A formula for the (remainder + 1) after division by 97 is

KN% = AN% − 97 * INT((AN% − 1)/97)

For example:

When AN% = 1, KN% 1
When AN% = 97, KN% = 97
When AN% = 98, KN% = 1
When AN% = 99, KN% = 2, and so on.

For BOB, KN% = 422 − 97 * INT(421/97) = 422 − 97 * 4 = 422 − 388 = 34.

Here's a test program that converts strings into key numbers using these formulas:

```
10 REM --- HASHDEMO (HASHES STRINGS INTO INTEGERS 1 TO 97) ---
15 LPRINT "STRING =", " AN% =", "KN%="
16 LPRINT "-------------------------------------"
20 READ S$
30 IF S$="XXX" THEN 90
40   GOSUB 210      'CONVERT S$ TO AN%
50 LPRINT S$, AN%, KN%
60 GOTO 20
90 STOP
200 '--- SUBROUTINE TO CONVERT S$ TO INTEGER KN%,  1<=KN%<=97 ---
210 AN%=0
220 FOR K=1 TO LEN(S$)
230   AN%=AN% + K*ASC(MID$(S$,K,1))
240 NEXT K
250 KN%=AN%-97*INT((AN%-1)/97)
260 RETURN
300 '--- SAMPLE STRINGS ---
310 DATA CAT, CIGARS, DOG, SAUERBRATEN, JOHN
320 DATA 621-4839, 123-45-6789, A, AB, ABC
330 DATA BOB, MOZART, GILDA, SHAKESPEARE, AGENT 007
340 DATA BACH, BEETHOVEN, BRAHMS, VERDI, PUCCINI
```

```
350 DATA ALPHA, BRAVO, CHARLIE, M61, M62, M63, M64, M65
355 DATA"ZZZZZZZZZZZ"
390 DATA XXX

>RUN

STRING =         AN% =          KN%=
-------------------------------------
CAT              449            61
CIGARS           1594           42
DOG              439            51
SAUERBRATEN      4948           1
JOHN             760            81
621-4839         1890           47
123-45-6789      3489           94
A                65             65
AB               197            3
ABC              398            10
BOB              422            34
MOZART           1679           30
GILDA            1042           72
SHAKESPEARE      4841           88
AGENT 007        2553           31
BACH             685            6
BEETHOVEN        3437           42
BRAHMS           1596           44
VERDI            1107           40
PUCCINI          2063           26
ALPHA            1070           3
BRAVO            1164           97
CHARLIE          2035           95
M61              332            41
M62              335            44
M63              338            47
M64              341            50
M65              344            53
ZZZZZZZZZZZ      5940           23
```

Notice that there were no collisions until BEETHOVEN came along. For all 28 strings there were 4 collisions: between BEETHOVEN and CIGARS, M63 and 621-4839, ALPHA and AB, M64 and BRAHMS. Using the "next available slot" rule would resolve these collisions by putting BEETHOVEN in 42, M63 in 48, ALPHA in 4, and M64 in 51, all of which happened to be free in this example.

### Superproject *(HASHFILE)*

Modify FILEBOX3 so that the user inputs a key *string* in line 320, instead of a key record number KN%. The program should then hash this into KN% (from 1 to 100), increasing KN% as needed to avoid collisions, and then calculate the proper PR% and SR% values for storage or retrieval.

> *Note:* Your program will have to know whether you want to *save* or *retrieve* information before doing this. The handling of collisions is different for these two cases (think about the garage analogy). It will also be necessary to allow the user to *delete* records when the file gets too full (this is like having cars leave the garage.)

**Superproject** *(MONYDISK)*

The program MONEY, explained at the end of Chapter 3, allows the user to input all kinds of financial data. But when the computer is turned off, this data all disappears. It would be much better if the data were saved on a disk file for future use and/or modification.

Write a program called *MONYDISK* that saves the data in G on disk, where G is shorthand for all the arrays GD( ), ..., GC( ). Use G to store all the data in main memory while the program is in use. But just before ending, the program should save all the G array information on a disk file. The name of this file should be established when the program is first run, and it should be keyed to the financial period given at the beginning of the program (eg: DATA284 for the second period of 1984). This can be done by letting the file name F$ = "DATA" + STR$(P) + STR$(IY).

If the program is later run for that same period, all the data on DATA284 should automatically be moved from disk into the G array before any new transactions are accepted. Then the variables for current check number, deposit number, checking account balance, etc. should be set to their proper new values. A convenient place to store this information is in the records corresponding to the *header* positions we reserved in G at locations 0, 100, 150, and 190.

These headers can also be used for another purpose. You will recall that the G array was rather large; it could hold up to 200 rows of data. To avoid copying unused parts of the G array onto disk, a useful technique is to store the number of locations actually used in each part of G in the corresponding header. These numbers can then be used to control how many logical records are saved on disk. For example, if only rows 1 to 10 are used in the check payment part of G, then the number 10 should be stored as part of the header in row 0. This header, and the ten rows of check payment data should be PUT on the file, but not rows 11 to 99. Later, when the program GETs this information, it can use the 10 in the header to know that the next ten records should be loaded back into locations 1 to 10 of G. It will also know that the very next logical record is the header corresponding to row 100 in G, and be able to use the information in that header to load the next portion of G, and so on down the file.

As you can see, this is a tough one. Take your time, and plan on trying a few smaller experiments to check out your ideas before writing a full blown version of *MONYDISK*. For example, see if you can write a pro-

gram that just saves and retrieves the "used" portions of one small array using the header technique suggested above. Invent other experiments in a similar manner, but keep them simple. Conquering each of these smaller projects will make the final project seem easier. Have fun!

## 4.4 WORD PROCESSING AND LINKED LISTS: EDIT1000, *EDIT2000, EDIT5000, (EDIT7000), (TEXTFORM)*

In this section we're going to put together a program called EDIT1000 that can be applied to *word processing,* one of the most valuable applications of small computers developed to date. As the name suggests, a word processing system is used for creating, modifying, and manipulating documents made up of words, or in general, of any symbols that can be printed on paper. These documents can be short memos, business letters, contracts, reports, term papers, magazine articles, or even books.

The hardware for a word processing system consists of four major components:

1. A video terminal (screen and keyboard) used for entering and modifying text.
2. A letter-quality printer for producing the final output.
3. A disk system for storing documents as data files.
4. The computer itself, including sufficient memory and the needed interface circuits.

In addition to the hardware, there are two kinds of software used:

1. A *text editing* program (usually just called an *editor*) that allows you to enter text, change it, delete it, move it around, save it, and/or retrieve it.
2. A *text formatting* program that takes the text produced by the editor and rearranges margin settings, spacing between lines, paragraph indentations, right justification, page numbering, and so on, in accordance with your instructions.

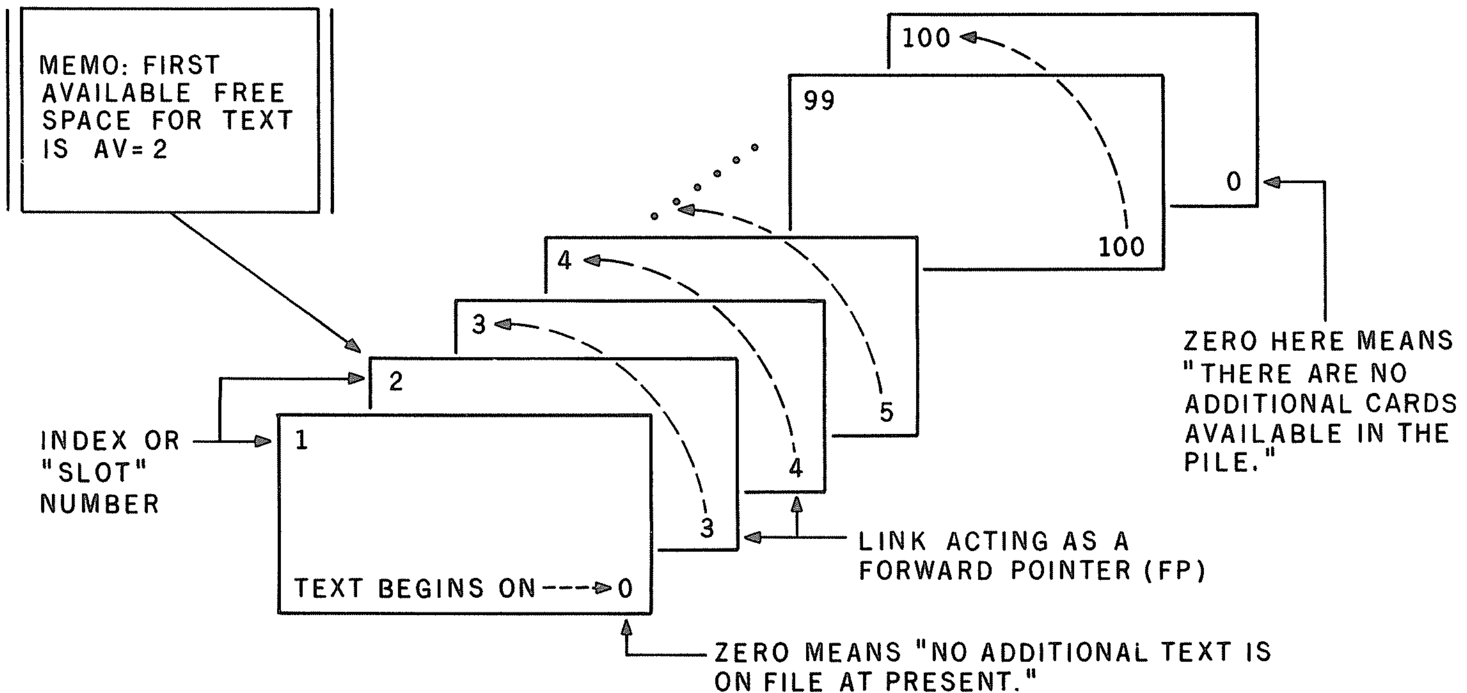
The most important of these two programs is the editor, since you can always format the text to some extent by the way you type it into the editor. In this section we'll show you how to write a simple but effective editor called EDIT1000. The features of EDIT1000 are based on the design specifications of the UNIX editor developed at Bell Laboratories. In section 4.6 we'll suggest additional features, resulting in the *EDIT5000* program.

### Linked Lists

Before designing and coding EDIT1000, it will be helpful if we take time out to discuss a data structure called the *linked list.* Once this is understood, it will be much easier to see how EDIT1000 works.

A non-computer example will help set the scene. Imagine that you are a writer with the following rather strange way of doing things. Instead of using ordinary sheets of paper, you write everything on file cards, putting only one line on each card. The cards are kept in a file box, and the numbers 1, 2, 3, 4, ... are written in the upper left hand corner in ink. The rule is that the cards will always be kept in the file box in the order given by the inked numbers.

Next, using pencil (because you expect to make some changes), you write a number in the lower right hand corner that "points to" or *links* each card to its successor. So when you first start, on card #2 you write a 3 as the link, on card #3 you write a 4 as the link, and so on. The last card has no successor, so you make its link 0 (zero). Here's a picture of what you have so far.




You can see from the diagram that each link is like the phrase "continued on card xx," except on card 1. We have used card #1 for a special purpose: to tell us where the first line of text of our article begins. Since we haven't written anything yet, we initialize the link on card #1 with a 0 (zero) to mean "no text in the file." Later on, as the file fills up with lines of text, a zero link will also be used on the last text-holding card to mean "no additional text in the file."

You'll also see another message, written on a piece of memo paper, that tells the author which is the first card available for writing text. Since card #1 is used for a special purpose, the first available card is initially #2, so we mark AV = 2 on the memo paper.

In our computer editor program, each card will correspond to an entry in a string array T$(I). The card number will correspond to the index I. It will be handy to call T$(I) "the Ith slot in the T$ array." The links will be stored in a separate "forward pointer" array FP(I). This can be pictured as sitting right next to T$(I). Here's what these arrays will look like initially.

AV = 2

| I | T$(I) | FP(I) |
|---|---|---|
| 1 | | 0 |
| 2 | | 3 |
| 3 | | 4 |
| . . . | . . . | . . . |
| 98 | | 99 |
| 99 | | 100 |
| 100 | | 0 |

This structure is called a linked list. The two most important operations in using this structure with an editor are *appending* (or adding) lines of text, and *deleting* lines of text. For example, here's what the arrays will look like after initially appending four lines of text:

AV = 6

| I | T$(I) | FP(I) |
|---|---|---|
| 1 | (not used) | 2 |
| 2 | PERSONAL COMPUTERS ARE GREAT BECAUSE (1) THEY'RE CHEAP, | 3 |
| 3 | (2) THEY KEEP THE KIDS OFF THE STREET, | 4 |
| 4 | (3) THEY EXERCISE YOUR IMAGINATION, | 5 |
| 5 | AND (4) THEY OPEN UP NEW CAREERS. | 0 |
| 6 | | 7 |
| 7 | | 8 |
| . . . | | . . . |
| 100 | | 0 |

To help illustrate a tricky point, we used a text that had numbers in it corresponding to the order in which text was initially entered. For example, the text "AND (4) THEY OPEN UP NEW CAREERS." was initially the fourth line entered. Note that these text line numbers are not the same as the slot numbers given by I. Also note that the zero link has shifted from slot 1 to slot 5.

Now let's see what happens when the author decides to squeeze in a new line right after the current second line. For example, suppose he wants

to make "PREVENTING WASTED HOURS AT THE POOL HALL," the new third line of text. One way to handle this request would be to move the last two lines down in order to "open up" a space. A much easier (and as we'll see more powerful) approach is to put the new text in the first available space, which is presently AV = 6, and then re-link in such a way that the order in which lines of text are to be read can be obtained from the links. Here's what we'll have:

AV = 7

| I | T$(I) | FP(I) |
|---|---|---|
| 1 | (not used) | 2 |
| 2 | PERSONAL COMPUTERS ARE GREAT BECAUSE (1) THEY'RE CHEAP, | 3 |
| 3 | (2) THEY KEEP THE KIDS OFF THE STREET, | 6 |
| 4 | (3) THEY EXERCISE YOUR IMAGINATION, | 5 |
| 5 | AND (4) THEY OPEN UP NEW CAREERS. | 0 |
| 6 | PREVENTING WASTED HOURS AT THE POOL HALL, | 4 |
| 7 | | 8 |
| 8 | | 9 |
| . . . | | . . . |
| 100 | | 0 |

To read the text now, look at the number in FP(1) to find out where the text starts, and then follow the links. Doing this will give you slots 2, 3, 6, 4, 5. If you want to know the implicit line numbers, that is, numbers that say in what order the text will eventually be printed, you'll have to count as you go down the list "1, 2, 3, 4, 5."

Question: Is "(3) THEY EXERCISE YOUR IMAGINATION" still the third line of the text?

Answer: No! It's now the *fourth* line. Simply count as you read in the proper order to see why.

To delete a line, all you do is readjust the links to point around the deleted line, and then put the slot number of the deleted line at the top of the list of available space (in AV). You don't have to erase the contents of the deleted line since that will happen automatically the next time a new line is appended at AV; when something is stored in computer memory the old contents are always erased. For example, here's what the arrays would look like if the second line of text were deleted.

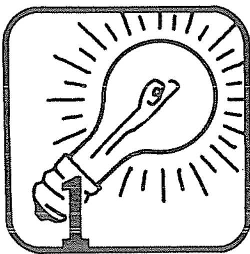
AV = 3

| I | T$(I) | FP(I) |
|---|---|---|
| 1 | (not used) | 2 |
| 2 | PERSONAL COMPUTERS ARE GREAT BECAUSE (1() THEY'RE CHEAP, | 6 |
| 3 | (2) THEY KEEP THE KIDS OFF THE STREET, | 7 |
| 4 | (3) THEY EXERCISE YOUR IMAGINATION, | 5 |
| 5 | AND (4) THEY OPEN UP NEW CAREERS. | 0 |
| 6 | PREVENTING WASTED HOURS AT THE POOL HALL, | 4 |
| 7 | | 8 |
| . . . | | . . . |
| 100 | | 0 |

To read the text, you now follow the sequence 2, 6, 4, 5. Slot #3 is no longer used for text, so we put it back at the top of the list of available space. To follow the list of available space, you start with the slot # in AV, and follow the links. This now gives slots 3, 7, 8, 9, ..., 100. As you can see, there are really two lists intertwined in T$(I), a list of text lines, and a list of available space. Let's now see how these ideas can be applied to an editor program.

### Program EDIT1000

#### The Idea



We want to write an editor program that allows a person to manipulate text. The text will be kept in an array T$( ) which can also be called a *text buffer*. The user will be able to use one-letter commands like A(ppend) text, D(elete) text, C(hange) text, L(ist) text on the video screen, and P(rint) text on paper. There will also be commands that allow the user to G(oto) any line, ask for H(elp), ask for I(nformation) about line numbers, T(ransfer) the contents of the video screen to paper, and V(erify) which line is currently being edited. Blocks of line numbers for later versions of the program which have additional commands such as W(rite) text on a disk file will also be reserved.

#### A Sample Run

In the run below, the EDIT1000 program was used to write a short announcement for use in a newsletter or on a bulletin board. Every place you see CMD? means that the editor program received a command from

the user. The command A is used to add or append lines of text starting after the line number you give. To get started, you append text after line 0, which means starting with line 1. To get out of append mode you type a single period. This puts you back in command mode. The example shows use of several other commands. You should notice in particular that frequent use of the L(ist) command is helpful in keeping track of the latest implicit line numbers. Anything after the symbol ">" is text that was entered by the user. Anything after ":" is text that was printed by the computer to help the user remember what was in a given line.

```
AFTER 'CMD?' TYPE A(PPEND), C(HANGE), D(ELETE), G(O TO)
H(ELP), I(NFO), L(IST), P(RINT), T(RANSFER), OR V(ERIFY).
RESPOND TO OTHER ? PROMPTS WITH A LINE #.  RESPONDING WITH
'ENTER' GIVES CURRENT LINE #. USE I CMD TO FIND CURRENT #.
CURRENT LINE NUMBER NOW = 0.  TO APPEND TEXT
AT LINE # 1 RESPOND   CMD? A   AFTER? 0
.............................................
CMD? A    AFTER? 0
 1>      ***  SOUTH BROOKLYN COMPUTER CLUB   ***
 2>NOTICE OF FORTHCOMING MEETINGS
 3>GOOD NEWS!  WE HAVE 4 GREAT MEETINGS LINED UP.
 4>.
CMD? C    FROM? 2    TO? 2
 2:NOTICE OF FORTHCOMING MEETINGS
 2>          NOTICE OF FORTHCOMING MEETINGS
CMD? L    FROM? 1    TO? 99
 1:      ***  SOUTH BROOKLYN COMPUTER CLUB   ***
 2:          NOTICE OF FORTHCOMING MEETINGS
 3:GOOD NEWS!  WE HAVE 4 GREAT MEETINGS LINED UP.
CMD? A    AFTER? 3
 4>MARCH 1: PROF. T. URING WILL SPEAK ON TAPE MACHINES
 5>APRIL 3: JOHN ROM WILL TALK ABOUT LOSING YOUR MEMORY
 6>MAY   22: MS. FLO DISK WILL SHOW SLIDES OF TERMINALS
 7>.
CMD? D    FROM? 4    TO? 4
CMD? L    FROM? 1    TO? 99
 1:      ***  SOUTH BROOKLYN COMPUTER CLUB   ***
 2:          NOTICE OF FORTHCOMING MEETINGS
 3:GOOD NEWS!  WE HAVE 4 GREAT MEETINGS LINED UP.
 4:APRIL 3: JOHN ROM WILL TALK ABOUT LOSING YOUR MEMORY
 5:MAY   22: MS. FLO DISK WILL SHOW SLIDES OF TERMINALS
CMD? A    AFTER? 4
 5>APRIL 9: JOE RAM WILL DEMONSTRATE HIS ACME III MICRO
 6>I
 7>.
CMD? D    FROM? 6    TO? 6
CMD? I
CURRENT LINE IS # 5     HIGHEST LINE IS # 6
CMD? G    WHERE? 6
 6:MAY   22: MS. FLO DISK WILL SHOW SLIDES OF TERMINALS
CMD? A    AFTER? 6
 7>JUNE 16: BIG ELECTION MEETING. NO SPEAKER.
 8>.
CMD? I
CURRENT LINE IS # 7     HIGHEST LINE IS # 7
```

```
CMD? L    FROM? 1    TO? 7
 1:      ***  SOUTH BROOKLYN COMPUTER CLUB   ***
 2:           NOTICE OF FORTHCOMING MEETINGS
 3:GOOD NEWS!  WE HAVE 4 GREAT MEETINGS LINED UP.
 4:APRIL 3: JOHN ROM WILL TALK ABOUT LOSING YOUR MEMORY
 5:APRIL 9: JOE RAM WILL DEMONSTRATE HIS ACME III MICRO
 6:MAY   22: MS. FLO DISK WILL SHOW SLIDES OF TERMINALS
 7:JUNE 16: BIG ELECTION MEETING. NO SPEAKER.
CMD? A    AFTER? 3
 4>THREE OF THEM WILL FEATURE OUTSTANDING SPEAKERS
 5>I
 6>.
CMD? D    FROM?     TO?
CMD? L    FROM? 1    TO? 99
 1:      ***  SOUTH BROOKLYN COMPUTER CLUB   ***
 2:           NOTICE OF FORTHCOMING MEETINGS
 3:GOOD NEWS!  WE HAVE 4 GREAT MEETINGS LINED UP.
 4:THREE OF THEM WILL FEATURE OUTSTANDING SPEAKERS
 5:APRIL 3: JOHN ROM WILL TALK ABOUT LOSING YOUR MEMORY
 6:APRIL 9: JOE RAM WILL DEMONSTRATE HIS ACME III MICRO
 7:MAY   22: MS. FLO DISK WILL SHOW SLIDES OF TERMINALS
 8:JUNE 16: BIG ELECTION MEETING. NO SPEAKER.
CMD? A    AFTER? 2
 3>
 4>.
CMD? L    FROM? 1    TO? 99
 1:      ***  SOUTH BROOKLYN COMPUTER CLUB   ***
 2:           NOTICE OF FORTHCOMING MEETINGS
 3:
 4:GOOD NEWS!  WE HAVE 4 GREAT MEETINGS LINED UP.
 5:THREE OF THEM WILL FEATURE OUTSTANDING SPEAKERS
 6:APRIL 3: JOHN ROM WILL TALK ABOUT LOSING YOUR MEMORY
 7:APRIL 9: JOE RAM WILL DEMONSTRATE HIS ACME III MICRO
 8:MAY   22: MS. FLO DISK WILL SHOW SLIDES OF TERMINALS
 9:JUNE 16: BIG ELECTION MEETING. NO SPEAKER.
CMD? A    AFTER? 5
 6>
 7>.
CMD? L    FROM? 1    TO? 99
 1:      ***  SOUTH BROOKLYN COMPUTER CLUB   ***
 2:           NOTICE OF FORTHCOMING MEETINGS
 3:
 4:GOOD NEWS!  WE HAVE 4 GREAT MEETINGS LINED UP.
 5:THREE OF THEM WILL FEATURE OUTSTANDING SPEAKERS
 6:
 7:APRIL 3: JOHN ROM WILL TALK ABOUT LOSING YOUR MEMORY
 8:APRIL 9: JOE RAM WILL DEMONSTRATE HIS ACME III MICRO
 9:MAY   22: MS. FLO DISK WILL SHOW SLIDES OF TERMINALS
10:JUNE 16: BIG ELECTION MEETING. NO SPEAKER.
CMD? I
CURRENT LINE IS # 10     HIGHEST LINE IS # 10
CMD? A    AFTER? 10
11>
12>          ***  PLEASE POST  ***
13>.
CMD? P    FROM? 1    TO? 12
TURN PRINTER ON -- READY?
```

*The I in line 5 was a mistake so we type a period to get back into CMD mode. Typing D followed by pressing 'Enter' for line numbers means 'Delete the current line' (which is #5)*

*Typing either a space or 'Enter' at line 3 will force a blank line in the text.*

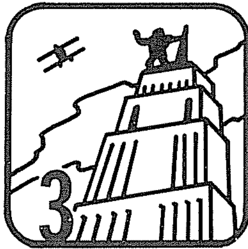
*Like this*

*We're finished so let's put the text out on a printer without line numbers*

```
     ***  SOUTH BROOKLYN COMPUTER CLUB  ***
        NOTICE OF FORTHCOMING MEETINGS

GOOD NEWS!  WE HAVE 4 GREAT MEETINGS LINED UP.
THREE OF THEM WILL FEATURE OUTSTANDING SPEAKERS

APRIL 3: JOHN ROM WILL TALK ABOUT LOSING YOUR MEMORY
APRIL 9: JOE RAM WILL DEMONSTRATE HIS ACME III MICRO
MAY  22: MS. FLO DISK WILL SHOW SLIDES OF TERMINALS
JUNE 16: BIG ELECTION MEETING. NO SPEAKER.

          ***  PLEASE POST  ***
```

*And here's what shows up on the printer*

## High-Level Design



The EDIT1000 program can be designed with five main program modules, followed by a number of command modules (in this case, six). We'll make provisions for expanding the program by reserving blocks of line numbers for additional command modules. The modules are as follows:

1. Set up line number variables; dimension arrays.
2. Give directions; open files (in advanced version)
3. Initialize the linked list.
4. Ask the user for a command.
5. Interpret the command, and branch to one of the command modules that follow.
6. I(nformation) command.
7. A(ppend) command.
8. C(hange) command.
9. D(elete command.
10. L(ist) command.
11. P(rint) command.
12. Future command modules.

*Note:* All command modules end with "GOTO step 4."

## Coding the Program

Before studying the code, you should know that there are two variables used to keep track of line numbers. These are CL, for *current line* number, and HL, for *highest line* number. CL keeps track of the line number you have most recently worked with, while HL holds the highest line number entered so far. HL starts out as 0, and every time a line is added it gets increased by 1. Every time a line is deleted, it gets decreased by 1.

There are also two variables, sometimes called *pointers,* used to keep track of which *slot* T$(I) corresponds to the current and highest line numbers. The current line pointer is called CP, and it "points" to the slot which holds the current line of text. In other words, the current text is in T$(CP). Think of CP as the number written (in ink) in the upper left hand corner of the file card holding the current text, while CL is the number you would get if you counted lines of text while following the links to get to this current text.

Similarly, a variable (or pointer) called HP is used to point to the slot where the highest line-numbered text is actually stored. In other words, the text with the highest line number is in T$(HP). Keep in mind that it's quite possible for text with a high line number (say HL = 10) to be stored in a low slot number (say HP = 3). This could happen if the text in slot 3 had been deleted earlier in the editing session.

The heart of the editor is the append module. Let's look at its design in further detail. Assume that the user has asked to append text after line L1. Then we must do the following (the program line numbers that will be used in the final program precede each step):

| | |
|---|---|
| 1310 | Check that L1 is not greater than HL. |
| 1311 | Input the line of text into a temporary buffer B$. |
| 1312 | Make B$ = " " if the user hit ENTER. |
| 1315 | If the user typed a period, exit from this module by way of line 1390. |
| 1320 | If not, make sure space is available in the linked list. |
| 1325 | Make the current line number (about to be appended) = L1 + 1. |
| 1330 | Start at the first slot (I = 1); start counting lines (K = 1). |
| 1335-1340 | Go down through the linked list, following the pointers (I = FP(I)), and counting (K = K + 1) until either we hit the end (FP(I) = 0), or we reach the desired line number (K = CL). |
| 1350 | Grab the first available slot (J = AV), and reset AV to the next free space (AV = FP(J)). |
| 1355 | Change the link in the new slot to the link that was in the previous slot (FP(J) = FP(I)). |
| 1360 | Change the link in the previous slot to the new slot number (FP(I) = J). |
| 1365 | Move the text from the temporary buffer into the new slot (T$(J) = B$). |
| 1370 | Reset the current pointer (CP = J) and increase the highest line number (HL = HL + 1). |
| 1375 | Reset the highest line pointer only if the new text has the highest line number (IF CL = HL THEN HP = CP). |
| 1380 | Go back to 1310 to possibly append another line of text. |
| 1390 | A period was typed, so go back for another command. |

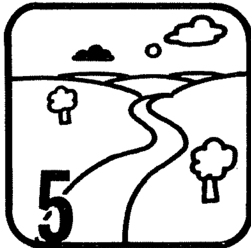
The above code is an English version that is a lot briefer when written in BASIC. You should study it by comparison with the actual lines of

BASIC from 1310 to 1390 in the program listing. The other modules work in a similar manner, and you should be able to figure them out with lots of doodling on paper to trace the changes in links. These ideas are usually taught in upper level college courses, so they may be a bit "slippery" at first. Here's a listing of the entire program EDIT1000:

```
10 ' ***************************************************
20 ' *     EDIT1000  (LINE EDITOR PROGRAM, PHASE 1)      *
30 ' ***************************************************
110 DEFINT A-Z                      'ALL VARIABLES TO BE INTEGER
120 CL=0 : HL=0                     'CL IS CURRENT LINE #
130 CLEAR 5000                      'HL IS HIGHEST LINE #
140 DIM T$(100), FP(100)            'T$( ) IS TEXT ARRAY BUFFER
190 CLS                             'CLS MEANS CLEAR SCREEN
191                                 'CLEAR 5000 MEANS RESERVE
192                                 '5000 BYTES FOR STRINGS
200 '--------------------------------------
201 '     GIVE DIRECTIONS; OPEN FILES
202 '--------------------------------------
210 GOSUB 610:PRINT "CURRENT LINE NUMBER NOW = 0.  TO APPEND TEXT"
215 PRINT "AT LINE # 1 RESPOND   CMD? A   AFTER? 0"
216 PRINT "..............................................."
300 '--------------------------------------
301 '     INITIALIZE LINKED LIST
302 '--------------------------------------
310 AV=2 : FP(1)=0                  'AV PTS TO TOP OF AVAIL LIST
320 FOR I=2 TO 99                   'FP(1) PTS TO TOP OF TEXT LIST
325   FP(I)=I+1                     '0 SIGNALS END OF LIST
330 NEXT I
340 FP(100)=0
400 '--------------------------------------
401 '     INPUT COMMAND
402 '--------------------------------------
450 PRINT"CMD? ";:LINE INPUT L$     'L$ HOLDS CMD CODE
455 L1=CL : L2=CL                   'DEFAULT VALUES
500 '--------------------------------------
501 '     INTERPRET COMMAND
502 '--------------------------------------
510 IF L$="I" THEN 1010
511 IF L$="H" GOSUB 610: GOTO 450
515 IF L$="V" PRINT USING"##:";CL;:PRINT T$(CP):GOTO 450
529 '
530 IF L$="A"  V$="AFTER" ELSE IF L$="G"  V$="WHERE" ELSE V$="FROM"
532 PRINT TAB(10);CHR$(27);V$;:INPUT L1: L2=L1
534 IF L$="A" THEN 1310
536 IF L$="G" THEN 1610
539 '
540 PRINT TAB(20);CHR$(27);"TO";:INPUT L2
542 IF L$="L" THEN 1610
544 IF L$="D" THEN 1510
546 IF L$="C" THEN 1410
548 IF L$="P" THEN 1710
590 PRINT "NO SUCH COMMAND" : GOTO 450
601 '
602 '........... HELP SUBROUTINE ...........
610 PRINT"AFTER 'CMD?' TYPE A(PPEND), C(HANGE), D(ELETE), G(O TO)"
620 PRINT"H(ELP), I(NFO), L(IST), P(RINT), T(RANSFER), OR V(ERIFY)."
640 PRINT"RESPOND TO OTHER ? PROMPTS WITH A LINE #.  RESPONDING WITH"
650 PRINT"'ENTER' GIVES CURRENT LINE #. USE I CMD TO FIND CURRENT #."
690 RETURN
```

```
1000 '-----------------------------------------
1001 '      I COMMAND (INFORMATION)
1002 '-----------------------------------------
1010 PRINT "CURRENT LINE IS #";CL;"    HIGHEST LINE IS #";HL
1090 GOTO 450   'RETURN FOR NEW CMD
1300 '-----------------------------------------
1301 '      A COMMAND (APPEND AFTER LINE L1)
1302 '-----------------------------------------
1310 IF L1>HL PRINT "LINE # TOO HIGH":GOTO 450
1311 PRINT USING "##>";L1+1;:LINE INPUT B$
1312 IF LEN(B$)=0 THEN B$=" "
1315 IF B$="." THEN 1390
1320 IF AV=0 PRINT "BUFFER FULL": GOTO 1390
1325 CL=L1+1
1330 I=1 : K=1
1335 IF FP(I)=0 OR K=CL THEN 1350        'I IS LINE POINTER
1340   I=FP(I) : K=K+1 : GOTO 1335       'K IS LINE COUNTER
1345 '
1350 J=AV : AV=FP(J)
1355 FP(J)=FP(I)
1360 FP(I)=J
1365 T$(J)=B$
1370 CP=J : HL=HL+1               'CP IS PTR TO CURRENT LINE
1375 IF CL=HL THEN HP=CP          'HP IS PTR TO HIGHEST LINE
1380 L1=CL : GOTO 1310
1390 GOTO 450   'RETURN FOR NEW CMD
1400 '-----------------------------------------
1401 '      C COMMAND (CHANGE LINES L1 TO L2)
1402 '-----------------------------------------
1410 CL=L1
1415 IF L2>HL PRINT "2ND LINE TOO HIGH":GOTO 450
1420 I=1 : K=1
1425 IF FP(I)=0 OR K=CL THEN 1440
1430   I=FP(I) : K=K+1 : GOTO 1425
1435 '
1440 IF FP(I)=0 PRINT "NO SUCH LINE" : GOTO 1490
1445   CP=FP(I)
1450   PRINT USING "##:";CL; : PRINT T$(CP)
1452   PRINT USING "##>";CL; : LINE INPUT T$(CP)
1455 IF CL<L2 THEN CL=CL+1 : GOTO 1420
1490 GOTO 450   'RETURN FOR NEW CMD
1500 '-----------------------------------------
1501 '      D COMMAND (DELETE LINES L1 TO L2)
1502 '-----------------------------------------
1510 CL=L1
1511 IF L2>HL THEN L2=HL
1512 FOR C=0 TO L2-L1
1515 I=1 : K=1
1520 IF FP(I)=0 OR K=CL THEN 1540
1525   I=FP(I) : K=K+1 : GOTO 1520
1530 '
1540 IF FP(I)=0 PRINT "NO TEXT TO DELETE": GOTO 1585
1545    J=FP(I)
1550    FP(I)=FP(J)
1555    FP(J)=AV
1560    AV=J
1565    HL=HL-1
1570    CP=I
1575   IF HL=CL THEN HP=CP
1580 NEXT C
1585 CL=L1-1: IF CL<0 THEN CL=0
1590 GOTO 450   'RETURN FOR NEW CMD
```

```
1600 '----------------------------------------
1601 '     L COMMAND (LIST LINES L1 TO L2)
1602 '----------------------------------------
1610 I=FP(1) : K=1
1615 IF L2>HL THEN L2=HL
1620 IF I=0 PRINT "BUFFER EMPTY" : GOTO 1690
1630   IF K<L1  THEN 1650
1640   CP=I: PRINT USING "##:";K;: PRINT T$(I)
1650   I=FP(I) : K=K+1
1655   IF K>L2 THEN 1680
1660   GOTO 1630
1680 CL=K-1
1690 GOTO 450   'RETURN FOR NEW CMD
1700 '----------------------------------------
1701 '     P COMMAND (PRINTS ON LINE PRINTER)
1702 '----------------------------------------
1710 I=FP(1) : K=1
1715 IF L2>HL THEN L2=HL
1720 IF I=0 PRINT "BUFFER EMPTY" : GOTO 1790
1725 PRINT "TURN ON PRINTER -- READY";:INPUT Z$
1730   IF K<L1 THEN 1750
1740   CP=I: LPRINT T$(I)
1750   I=FP(I) : K=K+1
1755   IF K>L2 THEN 1780
1760   GOTO 1730
1780 CL=K-1
1790 GOTO 450   'RETURN FOR NEW CMD
1799 '=====  BLOCKS RESERVED FOR FUTURE COMMANDS  =====
1801 '     M COMMAND (MOVE BLOCKS OF TEXT)
1901 '     E COMMAND (ECHO BLOCKS OF TEXT)
2001 '     R COMMAND (READ FILE INTO BUFFER AFTER CL)
2101 '     W COMMAND (WRITE BUFFER ONTO FILE)
2201 '     F COMMAND (FIND LINE WITH /TEXT/)
2301 '     C/ COMMAND (CHANGE /TEXT1/ TO /TEXT2/)
```




### Future Extensions and Revisions

Further extensions to this program will be discussed in section 4.6. However, one modification that should be considered here has to do with the use of the LINE INPUT command used in lines 450, 1311, and 1452. If you have a TRS-80 with Level II BASIC, but don't yet have disk-extended BASIC, LINE INPUT isn't available. This causes problems as explained in the note at the end of section 1.7. A way around this for Level II users is found in the next project.

### Project *EDIT2000*

Write a subroutine that simulates LINE INPUT in Level II BASIC by using the INKEY$ function (described on page 5/5 of the TRS-80 Level II BASIC reference book). Then modify EDIT1000 to use this subroutine so that text with commas can be input to the editor.

## 4.5 MACHINE LANGUAGE ANYONE? MLDEMO

The principal theme of this book has been a structured approach to creative programming. The principal tool recommended for pursuing this theme has been extended BASIC, used in conjunction with a five step design process. One might very well ask if languages other than BASIC could work in such a setting. The answer is a definite "Yes." In fact one of the reasons for stressing a structured design approach is to make the applications independent of the computer language used.

There are now several other interesting languages available for personal computers. On the one hand there are older languages (like FORTRAN) resurrected mostly because they are familiar to experienced programmers. There are also some new faces on the scene, with Pascal as the most interesting contender. We won't discuss either of these here except to warn you that the advocates of these and other languages tend to get carried away in extolling their virtues. Be assured that all computer languages have their faults, some of them very serious. Also be assured that you're not going to find a language with a better mix of conviviality and expressiveness than extended BASIC.

The main fault of BASIC is actually a consequence of its principal virtue: ease of use in an interactive mode. Technically this is because BASIC is usually implemented as an *interpreter* rather than a *compiler.* An interpreter translates one line at a time into *machine code,* (ie: instructions, all in binary representation, that a computer understands directly) and it does this every time the line is executed. So in a loop, lots of time is spent with the translation process. On the other hand, a compiler waits until you've entered the entire program, and then it translates it as a whole into machine code. So with a compiler, if you have made no mistakes, you can execute the machine language translation rather than the original BASIC program (called the *source* program). The problem is that doing all this, and keeping track of which stage you're at is more complicated than just saying RUN.* The advantage to using a compiler is that the execution time of the machine language program it produces will be very fast. But getting it to successfully produce a correct machine language program could take lots of *your* time.

One way to get some of the advantages of speed while still using an interactive BASIC interpreter is to translate parts of the program into machine code "by hand," and to replace the corresponding lines of BASIC with a *call* to this machine language program. However, this is not easy to do at present. Fortunately, most applications don't need all that speed (the main exception is animated computer graphics).

*Also with a compiler you cannot stop and print (or modify) variables, and then continue by typing CONT. To see one reason why you'd want to do this reread Program BUGPROG in section 2.2

## Program MLDEMO

To satisfy a natural curiosity about how one goes about mixing machine language with BASIC, we'll show you a demonstration program called MLDEMO in this section. However, we won't explain the machine code (this would really take another book). The program allows you to choose between using BASIC and machine language to do the following task: successively fill the screen with the letter A, then the letter B, and so on, through the alphabet up to Z. The speed with which this task is accomplished will be quite different when using machine language. In fact it is so fast, the program includes a feature that allows you to slow it down different amounts by inputting something called a *wait counter.* If you enter 255 for the wait counter, the program will wait about 0.8 of a second between putting up each screenful of letters. If you try a wait count of 10—zap! Try it and see what happens.

One possible use of this program is as a speed reading machine to teach children the alphabet. In this case, the child will probably prefer the BASIC version at first.

The machine code for displaying the alphabet at different speeds is found in lines 160, 165, and 170. This won't make any sense to you, of course. We got this code by first writing the program in something called assembly language, then translating it into hexadecimal machine language, finally changing this into the decimal numbers you see. This machine language program is loaded into memory by the loop containing the POKE statement in line 140. It is "called upon," and executed, by the USR1 function in line 550. Line 110 tells the USR1 function where we poked the machine language program (at hexadecimal address 5000 which is the same as 5 * 4096 = 20480 decimal).

Even if you didn't understand a word of the above, go ahead and use the program. Here it is.

```
10 REM*******************************************************
20 REM*   MLDEMO  (COMPARES BASIC WITH MACHINE LANGUAGE)    *
30 REM*******************************************************
50 CLS
100 '-----------------------------------------------
101 '      INITIALIZATION; LOAD Z-80 ML PROGRAM
102 '-----------------------------------------------
110 DEFUSR1=&H5000
115 K=0
120 PRINT "STARTING LOC FOR ML PROG IS 20480"
122 PRINT "CHANGE LINES 125 AND 110 IF A DIFFERENT LOC IS USED."
125 S=20480
130 READ X
135 IF X<0 THEN 210
140   POKE S+K,X
```

```
145   K=K+1
150   GOTO 130
160 DATA 22,26,30,65,33,0,60,1,0,4,115,35
165 DATA 11,120,177,194,10,80,28,33,255,255
170 DATA 43,124,181,194,22,80,21,194,4,80,201,-1
200 '--------------------------------------------------
201 '     PRINT INSTRUCTIONS; INPUT USER'S CHOICE
202 '--------------------------------------------------
210 PRINT "THIS PROGRAM DISPLAYS A SCREEN FULL OF A'S, OF B'S"
211 PRINT "OF C'S ETC. UP TO Z'S.  YOU HAVE A CHOICE OF SEEING"
212 PRINT "IT RUN WITH 2 VERSIONS OF bASIC OR MACHINE LANGUAGE."
218 PRINT "DO YOU WANT BASIC (B), POKE BASIC (P),"
219 PRINT "    MACHINE LANGUAGE (M), OR EXIT (E)";
220 INPUT A$
230 IF A$="B" THEN 310
240 IF A$="P" THEN 410
250 IF A$="M" THEN 510
260 IF A$="E" THEN 999
290 GOTO 218
300 '-------------------------------------------------
301 '     BASIC VERSION (USES PRINT STATEMENT)
302 '-------------------------------------------------
310 PRINT "BASIC USING PRINT--GET YOUR WATCH SET--READY";
315 INPUT D$
317 CLS
320 FOR A=65 TO 90
330   FOR L=1 TO 16
340     FOR C=1 TO 64
350       PRINT CHR$(A);
360     NEXT C
370   NEXT L
380   CLS
390 NEXT A
395 GOTO 610      'GO TO "DONE" SECTION
400 '-------------------------------------------------
401 '     COMPACT BASIC VERSION USING POKE
402 '-------------------------------------------------
410 T=15360: B=16383: A=65: Z=90
412 PRINT "BASIC USING POKE--GET YOUR WATCH SET--READY";
415 INPUT D$
417 CLS
440 FORL=ATOZ:FORP=TTOB:POKEP,L:NEXT:CLS:NEXT:CLS
490 GOTO 610      'GO TO "DONE" MESSAGE
500 '------------------------------------------------------
501 '     MACHINE LANGUAGE VERSION (8080 SUBSET OF Z-80)
502 '------------------------------------------------------
510 CLS : PRINT "Z-80 ML PROGRAM"
515 PRINT "ENTER WAIT COUNTER (0 TO 255)";
520 INPUT W
525 IF W<0 OR W>255 THEN 515
530 POKE S+21, W
535 IF W>0 THEN 550
540   PRINT "ENTER LOW ORDER WAIT BYTE (1 TO 255)";
545   INPUT LB
546   IF LB<1 OR LB>255 THEN 540
547   POKE S+20, LB
550 HL=USR1(X)
610 CLS
620 PRINT CHR$(23):PRINT @ 468, "DONE"
625 PRINT @704,"PRESS ENTER TO CONTINUE--READY";
626 INPUT D$ : CLS
630 GOTO 218
999 END
```

## A NOTE ON ASSEMBLY LANGAUGE

The program MLDEMO is written in BASIC, but it contains a machine language program stored in the form of decimal numbers. These are found in the DATA statements beginning at line 160 with the sequence 22, 26, 30, 65, and so on. When not using BASIC, these same machine language instructions are sometimes written as hexadecimal numbers (see Appendix A for an explanation of how decimal numbers can be written as hexadecimal numbers). If written in hex, this same machine language program would be written as the sequence 16, 1A, 1E, 41, and so on.

Programmers find it difficult to memorize all these strange looking machine codes, so they use something called assembly language instead. An assembly language program looks like this:

```
LD  D, 1AH     ; LOAD THE NUMBER 26 IN D
LD  E, 41H     ; LOAD THE NUMBER 65 IN E

        .......... etc. ..........
```

Here's a complete listing of the machine language program used in MLDEMO. The first column (5000, 5002, etc.) gives the hex memory addresses where the program is stored. The second column shows the machine language program written in hex (161A, 1E41), etc.). You should mentally insert a space between each pair of these symbols and read this as 16 1A 1E 41 etc. The third column is the set of editor line numbers used to write the assembly language program. These are numbered 00100 to 00310 in steps of 10 in our example. The last three columns are the assembly language form of the program. This is written first, using an editor program. Then it is run through a system program called an *assembler* which produces the first two columns.

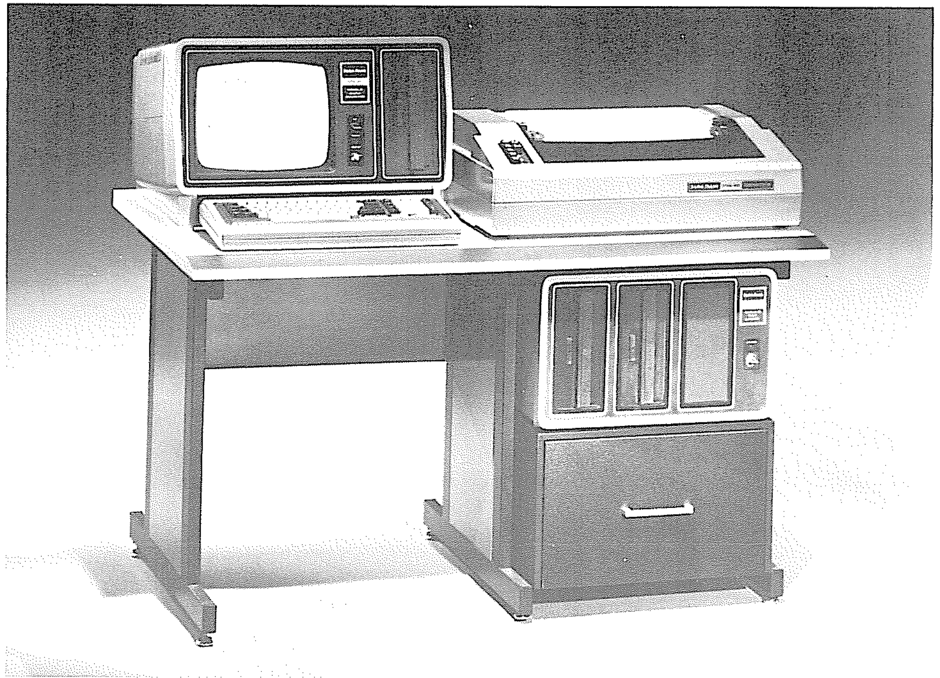
```
5000            00100         ORG    5000H
3C00            00110 VIDEO   EQU    3C00H
5000 161A       00120         LD     D, 1AH
5002 1E41       00130         LD     E, 41H
5004 21003C     00140 START   LD     HL, VIDEO
5007 010004     00150         LD     BC, 400H
500A 73         00160 LOOP    LD     (HL), E
500B 23         00170         INC    HL
500C 0B         00180         DEC    BC
500D 78         00190         LD     A, B
500E B1         00200         OR     C
500F C20A50     00210         JP     NZ, LOOP
5012 1C         00220         INC    E
5013 21FFFF     00230         LD     HL, 0FFFFH
5016 2B         00240 WAIT    DEC    HL
5017 7C         00250         LD     A, H
5018 B5         00260         OR     L
5019 C21650     00270         JP     NZ, WAIT
501C 15         00280         DEC    D
501D C20450     00290         JP     NZ, START
5020 C9         00300         RET
0000            00310         END
00000 TOTAL ERRORS
WAIT     5016
LOOP     500A
START    5004
VIDEO    3C00
```

## 4.6 SHOULD YOU INVEST IN A BUSINESS MICROCOMPUTER?

Now that we've seen some of the ways in which printers and disk drives can turn a personal computer into a professional machine, it's logical to ask what other improvements can be made. One possibility is to add more exotic peripheral devices, such as game paddles; voice synthesizers, which allow the computer to talk to you; speech recognizers, which allow you to talk to the computer; music synthesizers; device controllers, to turn appliances on or off; and device sensors, to check if windows, doors, etc are open or closed. All of these are fun, and available for most low-cost personal computers.

Suppose your interests lie more in the direction of business-related applications, including such things as word processing, payroll management, inventory control, accounts receivable, and so on? Then it makes sense to consider investing in a second machine. We used the word invest, rather than buy, to emphasize that more money will be involved, and that your decision should be based on the return you expect, possible tax credits, and other business-like considerations.

There have been some impressive price reductions in business computing systems in recent years, and you can now start with an investment that's less than the price of an automobile or small delivery van. The things to look for in purchasing such a machine are (1) good system software, especially a powerful, proven disk-extended BASIC, (2) a professional video monitor with 24 lines of 80 characters each, (3) adequate disk storage capability—at least a million bytes (which suggests 8-inch disks), (4) a modern and relatively fast central processing unit, (what most people would call the computer), (5) lots of main memory, preferably 64 K bytes total, and (6) provision for expansion, especially through RS-232-C interfaces. It is understood that a good printer would be added to this system, but this could very well be the printer you already own. For word processing applications a letter-quality printer should be used.



*This is the TRS-80 model II computer. It features a screen with 24 lines of 80 characters each, two built-in RS-232-C interfaces plus a parallel printer interface, and plug-in memory boards. One 8" floppy disk drive is built into the video display unit, and three more drives can be added externally. Its disk-extended BASIC has all the features described in this chapter.*

In addition to hardware and system software (like BASIC and DOS), a professional system needs good application software. This can be purchased as is, or it can be customized to your special needs. This book has tried to show you some of the techniques used in designing and coding such customized software. As you now know, it can be a challenging and time-consuming task. If you decide to hire someone else to do the job, keep in mind that large computer users have learned that this is no area in which to be penny-wise and pound-foolish. They have found that the cost of software in a big installation will often be more than that of the hardware. They have also found that maintaining software, ie: making changes, adding new features, and getting rid of bugs, is going to represent a major part of its cost. This is why we strongly advocate that you buy software that is very well-documented. If you expect to maintain it yourself, then only buy software for which the source code (eg: a BASIC listing) is furnished.

A good question to ask is whether you should attempt software maintenance, even assuming you have well-documented source listings. As our final project and superproject we'll suggest some additions and improvements to EDIT1000. If you can handle these (allowing for lots of trial and error), the answer is that you can probably handle some of your own software devleopment and maintenance, and more importantly, that you

would probably enjoy doing it. If you had to depend a lot on looking at our solution to the project, but you basically understood the solution after you saw it, then the answer is that you are capable of supervising a programmer who does the actual coding according to your specifications (which is a very important consideration.) Here's the project, followed by the superproject.

**Project** *EDIT5000*

Improve EDIT1000 by adding the following commands and command modules:

1. T(ransfer). This should use the SNAPSHOT program explained in section 4.2 to transfer whatever is on the screen onto the printer, including all the commands that were given.
2. C/OLD/NEW/. This is a "contextual" form of the C(hange) command. It should allow you to change part of the current line as follows:
   14: HI JOHN, HOWS IS YOU?
   CMD? C/WS IS/W ARE/
   14: HI JOHN, HOW ARE YOU?
3. F(ind). This locates the next line containing a given text as follows:
   CMD? F/O.K./
   17: YOUR PACKAGE WAS RECEIVED O.K.

The most difficult of these new modules is C(hange). Here are some ideas on how to write it. The line numbers in our explanation correspond to those used in the solution shown in chapter 5.

| | |
|---|---|
| 450 | Input the command into L$ as usual. |
| 520 | See if someone typed C/OLD/NEW/, by asking IF LEFT$(L$,2)="C/". If they did, go to the new contextual change module at line 2310 (if not, check for other commands). |
| 2310 | Initialize R$ as the empty string; this is where we'll put "OLD". |
| 2315-2325 | Extract the characters between the first two slashes one by one, using concatenation (+) to string them together in the variable R$. The trick to extracting each character is to use MID$(L$,P,1) for P = 3, 4, 5, ... until a slash is found. |
| 2330-2345 | Extract the characters between the second and third slash. Use the MID$ function again, but this time starting at position Q = P + 1, and going to a position one less than the number of characters in L$ (LEN(L$) − 1). |
| 2350-2390 | Now take the OLD text in R$ (which has a length of LR = P−3), and successively try to match it with the same number of characters in T$(CP). If you find a match (line 2365), then make up a new string for T$(CP) that consists of everything up to OLD, plus NEW, plus everything after OLD (line 2380). |

The above is tricky, but the string functions in BASIC make it relatively easy to write. It would be a good idea to restudy the explanations of these functions given at the end of section 2.2 before trying this module. (Incidently, it would be much harder to write this program in a professional language such as FORTRAN because it does not have such string functions.)

The command F/TEXT/ uses some of these same ideas. However, the solution we've shown has a problem: it only finds the *first* occurrence of TEXT. In the spirit of ending by encouraging a creative approach to programming, we will leave you with the superproject *(EDIT7000)* which suggests removing this "bug," and adding two more command modules as follows.

### Superproject *(EDIT7000)*

1. Modify the F(ind) commmand so that it searches forward from the current line number each time it's used, cycling back through the top of T$(  ) if necessary.
2. Add a W(rite) command that saves the contents of the buffer T$(I) onto a given disk file.
3. Add a R(ead) command that loads the contents of a given file into T$(I).
4. Add a Q(uit) command that ends the program, but first gives you a chance to save T$(  ) on disk if you haven't already done so, and which then closes all files.

For some help with 2, 3, and 4 restudy the program FILEBOX and the projects *FILEBOX2* and *FILEBOX3.*

Since all the projects and superprojects were based on extensions of programs explained in the book, you may be wondering if you could handle a tough programming assignment that didn't have this kind of help. There's only one way to find out. Here you go.

### Final Superproject *(TEXTFORM)*

Write a program that uses the text file produced by *(EDIT7000)* as input, and produces a custom formatted printed page as its output. Instructions explaining what kind of formatting you want should be inserted in the text in the form of two-letter commands preceded by a period. For example, to make part of a text indent 5 spaces and print with double spacing between lines, with another part indented 10 spaces, with single spacing between lines, you might insert commands as follows (using *(EDIT7000)):*

```
1 >  .IN5
2 >  .LS2
3 >  This is text that will
4 >  be double spaced and
```

```
 5 >   indented 5 spaces to the right.
 6 >   .IN10
 7 >   .LS1
 8 >   But this text will be
 9 >   single spaced and indented
10 >   ten spaces to the right.
```

When run through *(TEXTFORM)*, this text would print as follows:

```
     This is text that will

     be double spaced and

     indented 5 spaces to the right.

          But this text will be
          single spaced and indented
          ten spaces to the right.
```

Use your imagination to add other useful commands (or ask a typist what he or she would like to have). Bon Voyage!

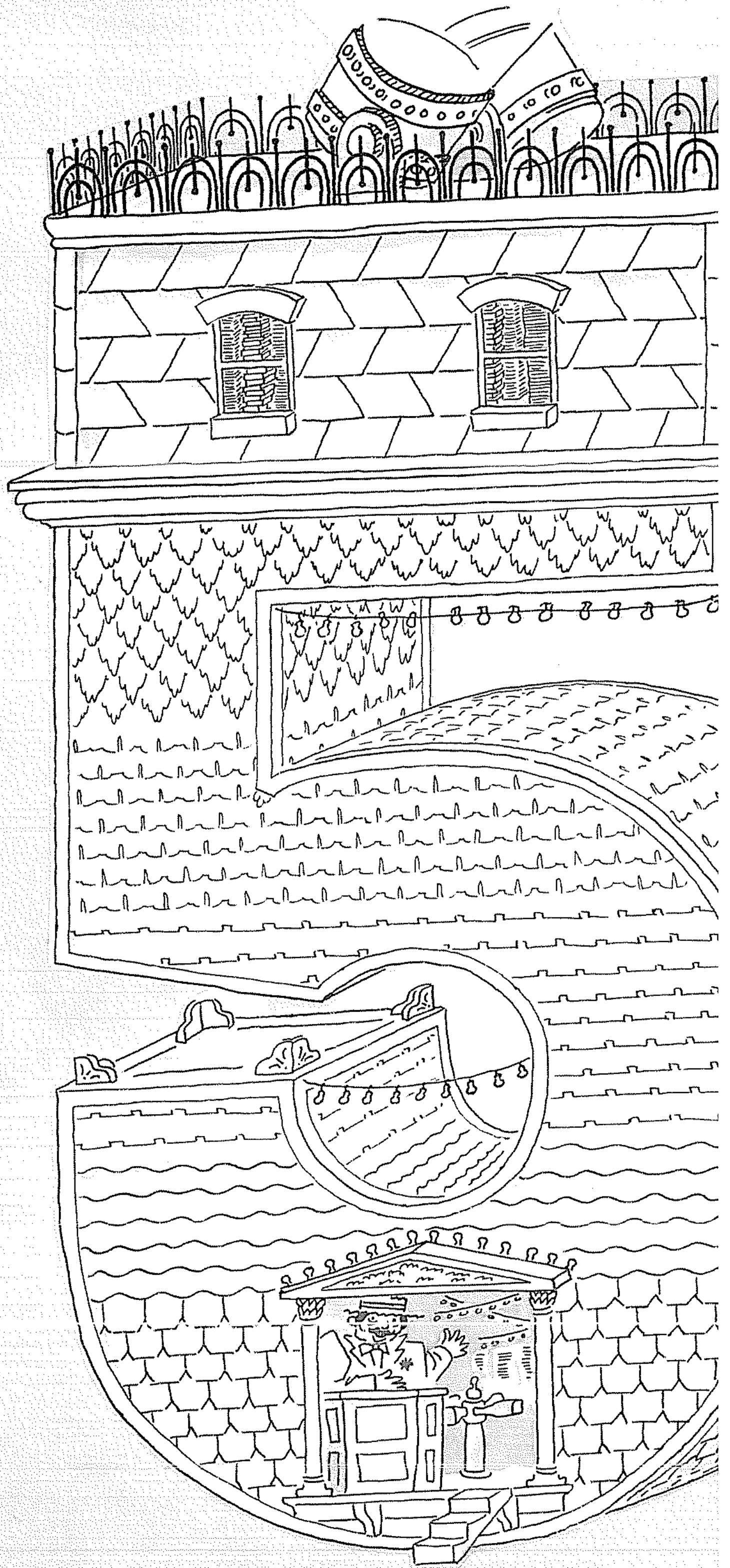
## SOURCES OF FURTHER INFORMATION FOR CHAPTER 4

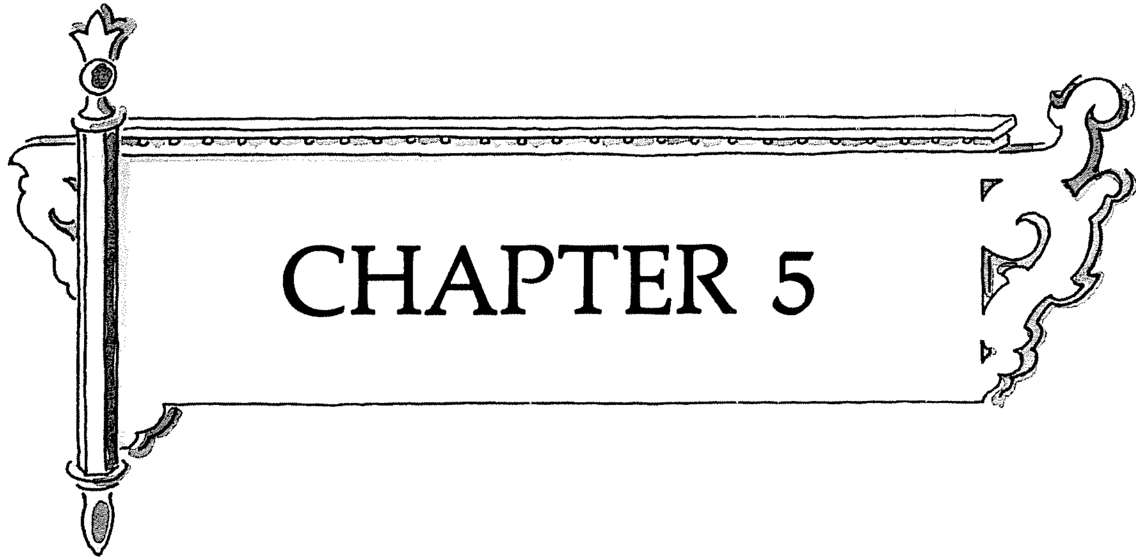
1. The best way to keep up with new hardware is to read the advertisements in the magazines referenced in earlier chapters. Professional journals like *Computer Design* are also helpful.

2 *BYTE*, *Creative Computing*, *Microcomputer*, *Interface Age*, and *Personal Computing*, all carry reviews of equipment. A new computing magazine for beginners that has longer, in-depth equipment reviews (in addition to many other features) is *onComputing* (70 Main St., Peterborough, NH 03458).

4. We haven't discussed computer music but it's a fascinating subject. The best technical journal in this area is *Computer Music* (People's Computer Co., Box E, 1263 El Camino Real, Menlo Park, CA 94025).

5. If you are interested in assembling a customized computer, subscribe to *S-100 Micro Systems*, Box 1192, Mountainside, NJ 07092.






# SOFTWARE CITY: SOLUTIONS TO THE PROJECTS

## 5.0 INTRODUCTION

The following sections contain solutions for all of the projects in this book. The shorter programs are written in minimal BASIC, while the longer programs use the features of TRS-80 Level II extended BASIC. The solutions for Chapter 4 were written on a TRS-80 disk system, using the disk operating system TRSDOS along with the disk-extended BASIC on this machine (which is essentially Microsoft disk-extended BASIC).

The solutions for Chapter 1 are given in section 5.1, solutions for Chapter 2 in section 5.2, and so on. The solutions given here represent only one of the possible approaches for each project, and you shouldn't hesitate to develop variations or improvements of your own. Although all the programs were successfully run in the form shown, it should be understood that there is no guarantee that you won't find bugs. In fact, since the solutions are meant to be used as learning aids rather than as production programs, safety features such as tests for illegal input have been occasionally omitted, especially where the extra code might have been confusing.

## 5.1 SOLUTIONS TO THE PROJECTS IN CHAPTER 1.

### Solutions to the Projects in Section 1.2

#### Project *GUTENTAG*

This program is very similar to the program BONJOUR in section 1.1. It is meant to be a first experience in typing and modifying a BASIC program.

```
5  REM --- GUTENTAG ---
10 FOR K=1 TO 5
20   PRINT TAB(2*K); "GUTENTAG"
30 NEXT K
40 END

>RUN
  GUTENTAG
    GUTENTAG
      GUTENTAG
        GUTENTAG
          GUTENTAG
```

#### Project *GOTCHA*

This program is similar to HYACINTH, section 1.2, but it produces output which looks best on a 32-column video screen. The solution given here is for a TRS-80 computer using double-width letters.

```
NEW
10 CLS
20 PRINT CHR$(23)
30 PRINT "*   GOTCHA!   *";
40 GOTO 30
RUN
```

To stop this program, press BREAK. If you then type LIST you'll see the program in double-sized letters. To see the program in regular-sized letters, press CLEAR, and type LIST again.

## Solution to the Project in Section 1.3

### Project *RICHQUIK*

*RICHQUIK* prints out both words and the results of calculations. It shows how large numbers are printed with scientific notation.

```
5 REM---RICHQUIK---
10 PRINT "DAY 1 = ";2[1
20 PRINT "DAY 2 = ";2[2
30 PRINT "DAY 3 = ";2[3
40 PRINT "DAY 4 = ";2[4
50 PRINT "DAY 5 = ";2[5
60 PRINT "DAY 10 = ";2[10
70 PRINT "DAY 20 = ";2[20
80 PRINT "DAY 30 = ";2[30
90 END

>RUN
DAY 1 =  2
DAY 2 =  4
DAY 3 =  8
DAY 4 =  16
DAY 5 =  32
DAY 10 =  1024
DAY 20 =  1.04858E+06
DAY 30 =  1.07374E+09
```

*The symbol ([) is the way some printers show the arrow (↑). When entering this program, type 2[1 as 2↑1.*

## Solutions to the Projects in Section 1.4

### Project *CARPET1*

This program takes input values from the user and produces calculations based on them.

```
5 REM---CARPET1---
10 PRINT "TYPE LENGTH, WIDTH (IN FT.) OF"
20 PRINT "BEDROOM:  ";
30 INPUT L1, W1
40 PRINT "LIVING ROOM:  ";
50 INPUT L2, W2
60 PRINT "DEN:  ";
70 INPUT L3, W3
80 LET T = L1*W1 + L2*W2 + L3*W3
90 PRINT "TOTAL CARPET NEEDED = ";T;" SQ. FT."
100 END

>RUN
TYPE LENGTH, WIDTH (IN FT.) OF
BEDROOM:  ? 20,15
LIVING ROOM:  ? 25,18
DEN:  ? 15,21
TOTAL CARPET NEEDED =  1065  SQ. FT.
```

### Project *CARPET2*

*CARPET2* requires adding another request for input, ie: cost per yard. This value is then used to get a total cost estimate.

```
5 REM---CARPET2  (CALCS. COST)---
10 PRINT "TYPE LENGTH, WIDTH (IN FT.) OF"
20 PRINT "BEDROOM:  ";
30 INPUT L1, W1
40 PRINT "LIVING ROOM:  ";
50 INPUT L2, W2
60 PRINT "DEN:  ";
70 INPUT L3, W3
80 LET T = L1*W1 + L2*W2 + L3*W3
90 PRINT "TOTAL CARPET NEEDED = ";T;" SQ. FT."
100 PRINT "WHAT IS COST PER SQ. YD. ";
110 INPUT C
120 PRINT "TOTAL CARPET NEEDED = ";T/9;" SQ. YDS."
130 PRINT "TOTAL COST =  $";(T/9)*C
140 END

>RUN
TYPE LENGTH, WIDTH (IN FT.) OF
BEDROOM:  ? 20,15
LIVING ROOM:  ? 25,18
DEN:  ? 15,21
TOTAL CARPET NEEDED =  1065  SQ. FT.
WHAT IS COST PER SQ. YD. ? 13.95
TOTAL CARPET NEEDED =  118.333  SQ. YDS.
TOTAL COST =  $ 1650.75
```

**Project** *GROCERY 2*

This is a modification of GROCERY1. It produces more shelf labels, and handles both pounds and ounces in the data.

```
2 REM---GROCERY2  (TAKES WEIGHT IN LBS., OZ.)---
3 LET S=0
4 PRINT"-----------------------------------------------------"
5 IF S<4 THEN 10
6 PRINT "TYPE 'ENTER' TO SEE MORE OUTPUT--READY";
7 INPUT D$
8 LET S=0
9 PRINT"-----------------------------------------------------"
10 READ N, L, Z, P
15 LET Q=16*L+Z
20 PRINT"PRODUCT LBS. OZ.  PRICE","UNIT PRICE"
30 PRINT N;TAB(7);L;TAB(12);Z;TAB(17);P,100*P/Q;"CENTS PER OZ."
35 LET S=S+1
40 GOTO 4
50 DATA 1, 1, 0, .89,     2, 0, 8, 1.49
60 DATA 3, 1, 8, .99,     4, 0, 2.5, 3.50
70 DATA 5, 10, 0, 2.99,     6, 0, 12, 1.25
80 DATA 7, 5, 0, 1.27,    8, 1, 0, 4.99
90 DATA 9, 0, 12, 2.77,     10, 1, 8, 12.89
100 END

>RUN
-----------------------------------------------------
PRODUCT LBS. OZ.  PRICE          UNIT PRICE
 1       1    0    .89            5.5625 CENTS PER OZ.
-----------------------------------------------------
PRODUCT LBS. OZ.  PRICE          UNIT PRICE
 2       0    8    1.49           18.625 CENTS PER OZ.
-----------------------------------------------------
PRODUCT LBS. OZ.  PRICE          UNIT PRICE
 3       1    8    .99            4.125 CENTS PER OZ.
-----------------------------------------------------
PRODUCT LBS. OZ.  PRICE          UNIT PRICE
 4       0    2.5  3.5            140 CENTS PER OZ.
-----------------------------------------------------
TYPE 'ENTER' TO SEE MORE OUTPUT--READY?
-----------------------------------------------------
PRODUCT LBS. OZ.  PRICE          UNIT PRICE
 5       10   0    2.99           1.86875 CENTS PER OZ.
-----------------------------------------------------
PRODUCT LBS. OZ.  PRICE          UNIT PRICE
 6       0    12   1.25           10.4167 CENTS PER OZ.
-----------------------------------------------------
PRODUCT LBS. OZ.  PRICE          UNIT PRICE
 7       5    0    1.27           1.5875 CENTS PER OZ.
-----------------------------------------------------
PRODUCT LBS. OZ.  PRICE          UNIT PRICE
 8       1    0    4.99           31.1875 CENTS PER OZ.
-----------------------------------------------------
TYPE 'ENTER' TO SEE MORE OUTPUT--READY?

-----------------------------------------------------
```

```
PRODUCT LBS. OZ.  PRICE          UNIT PRICE
 9       0    12  2.77            23.0833 CENTS PER OZ.
-----------------------------------------------------
PRODUCT LBS. OZ.  PRICE          UNIT PRICE
 10      1    8   12.89           53.7083 CENTS PER OZ.
-----------------------------------------------------
OUT OF DATA IN 10
READY
```

## Solution to the Project in Section 1.5

### Project *RETIRE*

This program is similar to the program VOTER. It adds another conditional statement.

```
10 REM---RETIRE  (ADDS MESSAGE)---
15 PRINT "WHAT IS YOUR AGE";
25 INPUT A
35 IF A < 18 THEN 65
45   PRINT "YOU ARE ELIGIBLE TO VOTE."
50   IF A < 65 THEN 55
51     PRINT"YOU ARE ELIGIBLE FOR '65+' BENEFITS."
55   STOP
65 PRINT"YOU WILL BE ABLE TO VOTE IN";18-A;"YEARS."
75 END

>RUN
WHAT IS YOUR AGE? 18
YOU ARE ELIGIBLE TO VOTE.
BREAK IN 55
READY
>RUN
WHAT IS YOUR AGE? 66
YOU ARE ELIGIBLE TO VOTE.
YOU ARE ELIGIBLE FOR '65+' BENEFITS.
BREAK IN 55
READY
```

## Solution to the Project in Section 1.6

### Project *SQCUBE*

This program gives further practice in making the computer print out tables of calculations using a FOR loop.

```
10 REM---SQCUBE---
20 PRINT"NUMBER","SQUARE","CUBE"
30 FOR X = 1 TO 12
40 PRINT X, X*X, X*X*X
50 NEXT X
60 END

>RUN
NUMBER          SQUARE          CUBE
 1               1               1
 2               4               8
 3               9               27
 4               16              64
 5               25              125
 6               36              216
 7               49              343
 8               64              512
 9               81              729
 10              100             1000
 11              121             1331
 12              144             1728
```

### Project *GOWRONG2*

*GOWRONG2* gives more practice using nested FOR loops.

```
5 REM---GOWRONG2---
10 PRINT "THIS IS A COMPUTER..."
20 FOR J = 1 TO 2
30 PRINT "NOTHING CAN
40 FOR K = 1 TO 3
50 PRINT "GO WRONG.","J =";J,"K =";K
60 NEXT K
70 NEXT J
80 END
```

```
>RUN
THIS IS A COMPUTER...
NOTHING CAN
GO WRONG.        J = 1          K = 1
GO WRONG.        J = 1          K = 2
GO WRONG.        J = 1          K = 3
NOTHING CAN
GO WRONG.        J = 2          K = 1
GO WRONG.        J = 2          K = 2
GO WRONG.        J = 2          K = 3
```

### Project *GOWRONG3*

More nested FOR loops (three this time) are used in this program.

```
10 REM---GOWRONG3---
20 FOR X = 1 TO 3
30 PRINT"THE GREEN KNIGHT"
40 FOR Y = 1 TO 2
50 PRINT"   RODE";
70 FOR Z = 1 TO 2
80 PRINT" OVER THE HILL,";
90 NEXT Z
95 PRINT
100 NEXT Y
110 NEXT X
120 PRINT"IN SEARCH OF THE DRAGON."
130 END

>RUN
THE GREEN KNIGHT
   RODE OVER THE HILL, OVER THE HILL,
   RODE OVER THE HILL, OVER THE HILL,
THE GREEN KNIGHT
   RODE OVER THE HILL, OVER THE HILL,
   RODE OVER THE HILL, OVER THE HILL,
THE GREEN KNIGHT
   RODE OVER THE HILL, OVER THE HILL,
   RODE OVER THE HILL, OVER THE HILL,
IN SEARCH OF THE DRAGON.
```

## Project *ICECREAM*

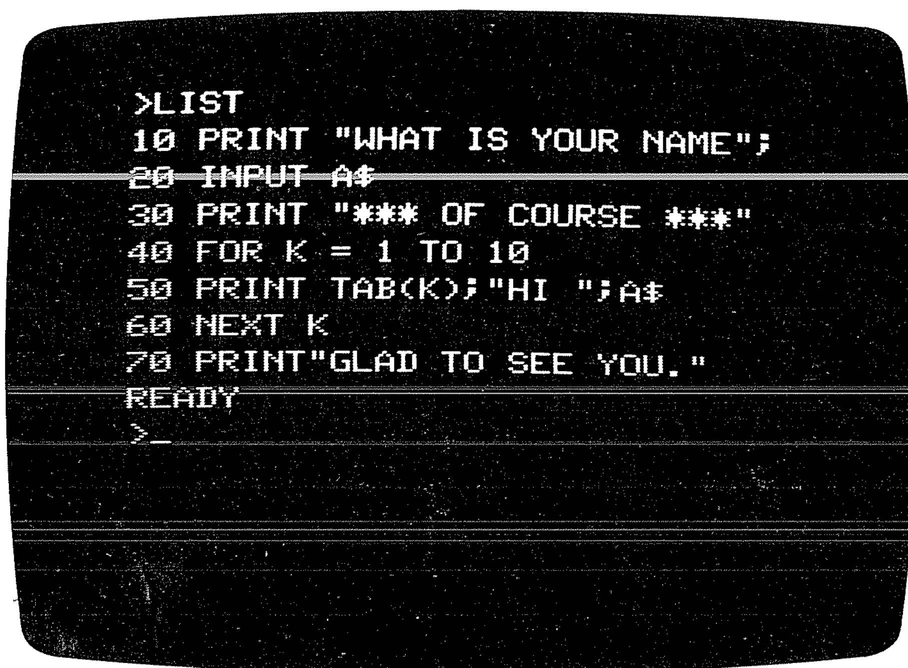
This is a modification of SALEDATA. It uses one FOR loop instead of two.

```
10 REM---ICECREAM (SALEDATA VERSION 2)---
15 DIM D(30)
20 PRINT"HOW MANY DAYS (MAX.= 30)";
25 INPUT N
30 IF N > 30 THEN 20
35 IF N < 1 THEN 998
40 LET L = 0
45 LET S = 999999
50 LET T = 0
60 FOR I = 1 TO N
70   PRINT"SALES FOR DAY #";I;
75   INPUT D(I)
120   IF D(I) > L THEN LET L = D(I)
130   IF D(I) < S THEN LET S = D(I)
140   T = T + D(I)
150 NEXT I
155 PRINT".................END OF DATA..............."
160 PRINT"SMALLEST SALE =";S;TAB(25);"TOTAL SALES  =";T
170 PRINT"LARGEST SALE  =";L;TAB(25);"AVERAGE SALE =";T/N
998 PRINT"..................FINISHED................."
999 END


>RUN
HOW MANY DAYS (MAX.= 30)? 9
SALES FOR DAY # 1 ? 47
SALES FOR DAY # 2 ? 24
SALES FOR DAY # 3 ? 78
SALES FOR DAY # 4 ? 0
SALES FOR DAY # 5 ? 12
SALES FOR DAY # 6 ? 33
SALES FOR DAY # 7 ? 79
SALES FOR DAY # 8 ? 80
SALES FOR DAY # 9 ? 32
.................END OF DATA...............
SMALLEST SALE = 0        TOTAL SALES  = 385
LARGEST SALE  = 80       AVERAGE SALE = 42.7778
..................FINISHED.................
```

## Solution to the Project in Section 1.7

### Project *SAM*

This program is a modification of FAMOUS that will print out nicely with a 40-column video screen.




*The photograph shows a solution to the project SAM that was listed using double-sized letters. To obtain such a listing, first press CLEAR to get a fresh screen. Then press the SHIFT and right arrow (→)* to go into double-sized mode, and type LIST. A photo of a run of SAM was shown in section 1.7.

### Project *POEM*

Here is a simple example of a "poem writing" program. Input words are combined with a standard text in print statements.

```
10 REM --- POEM ---
20 PRINT"WITH YOUR HELP, THIS PROGRAM WILL"
30 PRINT"WRITE AN 'ORIGINAL' POEM."
40 PRINT
50 PRINT"WHAT'S A GOOD WORD FOR A PLACE--"
60 PRINT"LIKE DESERT, MOON, FOREST, ALLEY, ETC.    ";
65 INPUT P$
70 PRINT"THANKS."
80 PRINT"GIVE ME A PLURAL NOUN FOR SOME"
85 PRINT"THINGS THAT BELONG ON LAND.               ";
86 INPUT Q$
90 PRINT"THANKS."
100 PRINT"NOW I NEED A PLURAL NOUN FOR SOME"
110 PRINT"THINGS THAT LIVE IN THE SEA.              ";
115 INPUT R$
120 PRINT"THANKS."
130 PRINT
150 PRINT"          A RIDDLE"
160 PRINT
170 PRINT"THE MAN IN THE ";P$;" ASKED OF ME,"
180 PRINT"HOW MANY ";Q$;" GROW IN THE SEA?"
190 PRINT"I ANSWERED HIM AS I THOUGHT GOOD,"
200 PRINT"AS MANY AS ";R$;" GROW IN THE WOOD."
205 PRINT
206 PRINT
210 PRINT"WANT TO PRINT IT AGAIN (Y = YES)";
220 A$ = " "
225 INPUT A$
226 IF A$ = "Y" THEN 130
230 PRINT"WANT TO WRITE ANOTHER (Y = YES)";
235 A$ = " "
240 INPUT A$
250 IF A$ = "Y" THEN 40
260 END


>RUN
WITH YOUR HELP, THIS PROGRAM WILL
WRITE AN 'ORIGINAL' POEM.

WHAT'S A GOOD WORD FOR A PLACE--
LIKE DESERT, MOON, FOREST, ALLEY, ETC.    ? ARCTIC
THANKS.
GIVE ME A PLURAL NOUN FOR SOME
THINGS THAT BELONG ON LAND.               ? HOT CAKES
THANKS.
NOW I NEED A PLURAL NOUN FOR SOME
THINGS THAT LIVE IN THE SEA.              ? OYSTERS
THANKS.


          A RIDDLE

THE MAN IN THE ARCTIC ASKED OF ME,
HOW MANY HOT CAKES GROW IN THE SEA?
I ANSWERED HIM AS I THOUGHT GOOD,
AS MANY AS OYSTERS GROW IN THE WOOD.


WANT TO PRINT IT AGAIN (Y = YES)? N
WANT TO WRITE ANOTHER (Y = YES)? N
```

### Project *APRIL1*

Again the basic idea is to fill in the blanks with the user's words and have the computer print out the results.

```
10 '--------------------------------------------------
11 '     APRIL1  (INSERTS USER'S WORDS IN A TEXT)
12 '--------------------------------------------------
15 CLS
20 PRINT"***  APRIL 1ST NEWSPAPER ARTICLE  ***
30 PRINT
40 CLEAR 200
100 REM---GET PERSON'S WORDS------------------------
110 PRINT"PLEASE TYPE IN THE FOLLOWING:"
120 PRINT"A NOUN (A MESSY SUBSTANCE)          ";
125 INPUT A$
130 PRINT"A NOUN (PERSONAL PROPERTY)          ";
135 INPUT B$
140 PRINT"  OH, HOW DO YOU SPELL THE PLURAL OF THAT";
145 INPUT C$
150 PRINT"THE NAME OF A CRAFT OR INDUSTRY     ";
155 INPUT D$
160 PRINT"A TOOL                              ";
165 INPUT E$
170 PRINT"A COMMON HOUSEHOLD ARTICLE          ";
175 INPUT F$
180 PRINT"  AND HOW DO YOU SPELL THE PLURAL OF THAT";
185 INPUT G$
190 PRINT"ANOTHER HOUSEHOLD ARTICLE           ";
195 INPUT H$
196 PRINT"A LIQUID                            ";
197 INPUT I$
198 PRINT"     THANK YOU."
200 REM---PRINT NEWS ARTICLE, INSERT WORDS---------
202 FOR I = 1 TO 200: NEXT I: CLS
205 PRINT
210 PRINT
215 PRINT"               AUNT HOMEBODY"
220 PRINT
225 PRINT"     WE'VE TOLD YOU HOW TO REMOVE ";A$;" SPOTS FROM"
230 PRINT"WALLS AND FURNITURE--NOW FOR ";C$;"!"
235 PRINT"     THE ";D$;" ASSOCIATION TELLS US THAT IF THE"
240 PRINT B$;" IS WASHABLE AND HAS ONLY A FEW SPOTS, DO THIS:"
245 PRINT"     REMOVE AS MUCH OF THE ";A$;" AS POSSIBLE WITH A"
250 PRINT"DULL ";E$;" (A LEFT-HANDED ";E$;" IS PERFECT).  NOW PLACE"
255 PRINT"THE STAINED AREA BETWEEN ";G$;" AND PRESS WITH A"
260 PRINT"WARM ";H$;".  AFTER YOU HAVE DONE THIS, PLACE THE"
265 PRINT B$;", STAIN SIDE DOWN, ON A ";F$;" AND RUB ANY"
270 PRINT"REMAINING STAIN ON THE BACK WITH ";I$"."
275 PRINT"     LET DRY AND LAUNDER AS USUAL."
500 END
```

## Solutions to the Projects for Section 1.8

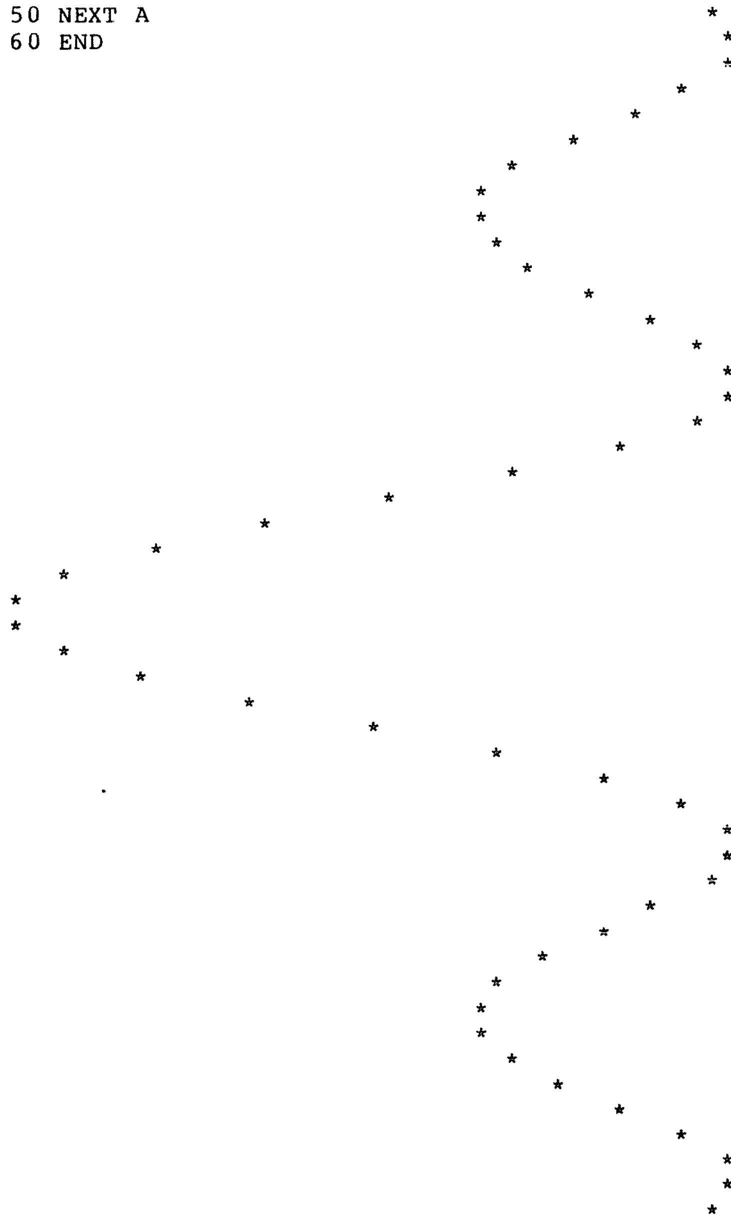
### Project *CRAPS*

This program uses the RND function in a game situation—as a subroutine in a Craps simulation program (subroutines are explained in section 1.9).

```
100 '--------------------------------------------------------------
101 '                      CRAPS GAME
102 '--------------------------------------------------------------
110 '------ASK FOR BANKROLL, BET------
120 PRINT"SIMULATED CRAPS.  WHAT IS YOUR BANKROLL";
130 INPUT M
135 IF M < 5 THEN PRINT"NOT ENOUGH FOR THIS GAME.": GOTO 120
140 PRINT"YOUR BET IS...";
145 B=0
150 INPUT B
152 IF B > M THEN PRINT"TOO HIGH.";: GOTO 140
154 IF B < 1 THEN PRINT"TOO LOW.";: GOTO 140
160 PRINT"***TO ROLL DICE PRESS ENTER (OR RETURN) AT THE ?***"
170 GOSUB 1000         'ROLLS DICE
200 '------TEST FOR WIN, LOSE OR POINT------
205 IF R = 7 THEN 400
210 IF R = 11 THEN 400
220 IF R = 2 THEN PRINT"SNAKE EYES": GOTO 300
230 IF R = 3 THEN PRINT"CRAPS": GOTO 300
240 IF R = 12 THEN PRINT"BOX CARS": GOTO 300
250 LET P = R: PRINT"YOUR POINT IS";P;
260 GOSUB 1000         'ROLLS DICE
280 IF R = 7 THEN PRINT"CRAPPED OUT":GOTO 300
290 IF R = P THEN 400
295 GOTO 260
300 '---------LOSE CALCULATION------
305 PRINT"  SORRY, YOU LOSE.  ";
310 LET M = M - B
320 PRINT"YOU NOW HAVE $";M
330 GOTO 450
400 '---------WIN CALCULATION------
405 PRINT"  YOU WIN!  ";
410 LET M = M + B
420 PRINT"YOU NOW HAVE $";M
445 '------DECIDE WHETHER TO CONTINUE------
450 IF M <= 0 THEN PRINT"LOOKS LIKE YOU'RE BROKE": GOTO 480
455 PRINT"WANT TO PLAY AGAIN (Y = YES)";
460 INPUT X$
470 IF X$ = "Y" THEN 140
480 PRINT"YOU ENDED WITH $";M;".  ";
490 PRINT"SEE YOU LATER": GOTO 9999
1000 '-------------------------------------------------------------
1001 '                     SUBROUTINE: ROLL DICE
1002 '-------------------------------------------------------------
1010 PRINT"      ";
1020 INPUT R$
1030 LET D1 = INT(RND(0)*6 + 1)
1040 LET D2 = INT(RND(0)*6 + 1)
1050 LET R = D1 + D2
1060 PRINT"YOUR ROLL IS";R;".  ";
1070 RETURN
9999 END
```

### Project *TABGRAPH*

This is a first exercise in mathematical graphing using the TAB function. You could also print out the values of Y and study them from a numerical as well as a pictorial point of view.



```
10 REM ----- TABGRAPH -----
20 FOR A=0 TO 9.5 STEP .2
30   LET Y=COS(2*A)+SIN(A)
40   PRINT TAB(15*Y+30); "*"
50 NEXT A
60 END
```

## 5.2 SOLUTIONS TO THE PROJECTS FOR CHAPTER 2

### Solutions to the Projects in Section 2.1

### Project *DATARITH*

The following four programs, DATARITH, DATARIT2, DATARIT3 (version 1), and DATARIT3 (version 2), are arithmetic quizzes like ARITH. They show some of the different ways data can be handled with DATA statements.

```
10  REM---DATARITH  (USES READ,DATA FOR NUMS.)---
100 REM---INTRODUCTION, INITIALIZE VARIABLES---
110 PRINT "ADDITION PRACTICE"
120 LET R = 0
130 PRINT"HOW MANY PROBLEMS (UP TO 10)";
140 INPUT N
150 IF N > 10 THEN 130
200 REM---GENERATE PROBLEM---------------------
205 FOR I = 1 TO N
220   READ A,B
230   DATA 2,3, 3,7, 7,9, 9,12, 12,5
240   DATA 12,7, 13,7, 13,8, 14,9, 15,20
300   REM---PRESENT PROBLEM---------------------
310   PRINT A;"+";B;"=  ";
400   REM---INPUT ANSWER---
410   INPUT X
500   REM---CHECK IF ANSWER IS RIGHT/WRONG-------
510   IF X = A + B THEN 550
520     PRINT "WRONG.  THE ANSWER IS";A+B
530     GOTO 610
550   PRINT "RIGHT!!"
560   LET R = R + 1
600 REM---GET NEXT PROBLEM---------------------
610 NEXT I
700 REM---REPORT SCORE-------------------------
710 PRINT "YOUR SCORE IS ";R/N*100;"% RIGHT."
800 REM---FUTURE EXTENSION OF PROGRAM----------
900 END


>RUN
ADDITION PRACTICE
HOW MANY PROBLEMS (UP TO 10)? 3
 2 + 3 =  ? 5
RIGHT!!
 3 + 7 =  ? 11
WRONG.  THE ANSWER IS 10
 7 + 9 =  ? 17
WRONG.  THE ANSWER IS 16
YOUR SCORE IS  33.3333 % RIGHT.
```

## Project *DATARIT2*

```
10  REM---DATARIT2  (USES RESTORE)-------------
100 REM---INTRODUCTION, INITIALIZE VARIABLES---
110 PRINT "ADDITION PRACTICE"
120 PRINT"HOW MANY PROBLEMS (UP TO 10)";
130 INPUT N
140 LET R = 0
150 IF N > 10 THEN 130
200 REM---GENERATE PROBLEM---------------------
205 FOR I = 1 TO N
220   READ A,B
230   DATA 2,3, 3,7, 7,9, 9,12, 12,5
240   DATA 12,7, 13,7, 13,8, 14,9, 15,20
300   REM---PRESENT PROBLEM--------------------
310   PRINT A;"+";B;"=  ";
400   REM---INPUT ANSWER---
410   INPUT X
500   REM---CHECK IF ANSWER IS RIGHT/WRONG-----
510   IF X = A + B THEN 550
520     PRINT "WRONG.  THE ANSWER IS";A+B
530     GOTO 610
550   PRINT "RIGHT!!"
560   LET R = R + 1
600 REM---GET NEXT PROBLEM---------------------
610 NEXT I
700 REM---REPORT SCORE-------------------------
710 PRINT "YOUR SCORE IS ";R/N*100;"% RIGHT."
800 REM---FUTURE EXTENSION OF PROGRAM----------
810 PRINT"WANT TO TRY THE SAME PROBLEMS AGAIN (Y = YES)";
820 INPUT A$
830 IF A$ <> "Y" THEN 900
840 RESTORE
850 GOTO 140
900 END


>RUN
ADDITION PRACTICE
HOW MANY PROBLEMS (UP TO 10)? 3
 2 + 3 =  ? 5
RIGHT!!
 3 + 7 =  ? 10
RIGHT!!
 7 + 9 =  ? 17
WRONG.  THE ANSWER IS 16
YOUR SCORE IS  66.6667 % RIGHT.
WANT TO TRY THE SAME PROBLEMS AGAIN (Y = YES)? Y
 2 + 3 =  ?
BREAK IN 410
```

## Project *DATARIT3*

```
10  REM---DATARIT3  (1 - USES INT(I/10) )---
100 REM---INTRODUCTION, INITIALIZE VARIABLES---
110 PRINT "ADDITION PRACTICE"
120 LET R = 0
130 PRINT"HOW MANY PROBLEMS";
140 INPUT N
200 REM---GENERATE PROBLEM--------------------
205 FOR I = 1 TO N
220   READ A,B
225 IF INT(I/10) = I/10 THEN RESTORE
230   DATA 2,3, 3,7, 7,9, 9,12, 12,5
240   DATA 12,7, 13,7, 13,8, 14,9, 15,20
300   REM---PRESENT PROBLEM---------------------
310   PRINT A;"+";B;"=  ";
400   REM---INPUT ANSWER---
410   INPUT X
500   REM---CHECK IF ANSWER IS RIGHT/WRONG-------
510   IF X = A + B THEN 550
520     PRINT "WRONG.  THE ANSWER IS";A+B
530     GOTO 610
550   PRINT "RIGHT!!"
560   LET R = R + 1
600 REM---GET NEXT PROBLEM--------------------
610 NEXT I
700 REM---REPORT SCORE------------------------
710 PRINT "YOUR SCORE IS ";R/N*100;"% RIGHT."
800 REM---FUTURE EXTENSION OF PROGRAM----------
900 END


10  REM---DATARIT3  (2 - USES A = 0 )---
100 REM---INTRODUCTION, INITIALIZE VARIABLES---
110 PRINT "ADDITION PRACTICE"
120 LET R = 0
130 PRINT"HOW MANY PROBLEMS";
140 INPUT N
200 REM---GENERATE PROBLEM--------------------
205 FOR I = 1 TO N
220   READ A,B
225 IF A = 0 THEN RESTORE: I=I-1:  GOTO 610
230   DATA 2,3, 3,7, 7,9, 9,12, 12,5
240   DATA 12,7, 13,7, 13,8, 14,9, 15,20, 0,0
300   REM---PRESENT PROBLEM---------------------
310   PRINT A;"+";B;"=  ";
400   REM---INPUT ANSWER---
410   INPUT X
500   REM---CHECK IF ANSWER IS RIGHT/WRONG-------
510   IF X = A + B THEN 550
520     PRINT "WRONG.  THE ANSWER IS";A+B
530     GOTO 610
550   PRINT "RIGHT!!"
560   LET R = R + 1
600 REM---GET NEXT PROBLEM--------------------
610 NEXT I
700 REM---REPORT SCORE------------------------
710 PRINT "YOUR SCORE IS ";R/N*100;"% RIGHT."
800 REM---FUTURE EXTENSION OF PROGRAM----------
900 END
```

## Project *DATARIT4*

This program is the fanciest data-using arithmetic quiz, but it allows the user to specify different levels of difficulty in the quiz.

```
10  REM---DATARIT4  (USES DUMMY READ)---
100 REM---INTRODUCTION, INITIALIZE VARIABLES---
110 PRINT "ADDITION PRACTICE"
120 LET R = 0
130 PRINT"HOW MANY PROBLEMS";
140 INPUT N
160 PRINT"HOW HARD?  0=EASY 1=MEDIUM 2=HARD";
170 INPUT X
180 IF X = 0 THEN 205
190 FOR I = 1 TO X*10
192 READ D,D
194 NEXT I
200 REM---GENERATE PROBLEM--------------------
205 FOR I = 1 TO N
220   READ A,B
225 IF A = 0 THEN RESTORE: I=I-1:  GOTO 610
230   DATA 2,2, 3,2, 1,4, 5,5, 6,2
235   DATA 3,7, 4,5, 8,6, 4,7, 5,8
240   DATA 9,6, 9,9, 10,7, 12,8, 13,6
245   DATA 14,8, 15,6, 15,9, 16,5, 16,9
250   DATA 17,5, 17,9, 18,2, 18,6, 19,5
255   DATA 19,8, 20,5, 21,8, 22,9, 23,8, 0,0
300   REM---PRESENT PROBLEM---------------------
310   PRINT A;"+";B;"=  ";
400   REM---INPUT ANSWER---
410   INPUT X
500   REM---CHECK IF ANSWER IS RIGHT/WRONG-------
510   IF X = A + B THEN 550
520     PRINT "WRONG.  THE ANSWER IS";A+B
530     GOTO 610
550   PRINT "RIGHT!!"
560   LET R = R + 1
600 REM---GET NEXT PROBLEM--------------------
610 NEXT I
700 REM---REPORT SCORE------------------------
710 PRINT "YOUR SCORE IS ";R/N*100;"% RIGHT."
800 REM---FUTURE EXTENSION OF PROGRAM----------
900 END
```

## Project *RNDARITH*

The following five arithmetic quizzes use the RND function in various ways to get the numbers for the problems. They do not use DATA statements.

```
10 REM---RNDARITH  (USES RND TO GET NUMS.)-----
100 REM---INTRODUCTION, INITIALIZE VARIABLES---
110 PRINT "ADDITION PRACTICE"
120 LET R = 0
130 PRINT "HOW MANY PROBLEMS DO YOU WANT";
140 INPUT N
200 REM---GENERATE PROBLEM--------------------
205 FOR I = 1 TO N
220   LET A = INT(RND(0)*15)+1
230   LET B = INT(RND(0)*15)+1
300   REM---PRESENT PROBLEM-------------------
310   PRINT A;"+";B;"=  ";
400   REM---INPUT ANSWER----------------------
410   INPUT X
500   REM---CHECK IF ANSWER IS RIGHT/WRONG-----
510   IF X = A + B THEN 550
520     PRINT "WRONG.  THE ANSWER IS";A+B
530     GOTO 610
550   PRINT "RIGHT!!"
560   LET R = R + 1
600 REM---GET NEXT PROBLEM--------------------
610 NEXT I
700 REM---REPORT SCORE------------------------
710 PRINT "YOUR SCORE IS ";R/N*100;"% RIGHT."
800 REM---FUTURE EXTENSION OF PROGRAM----------
900 END


>RUN
ADDITION PRACTICE
HOW MANY PROBLEMS DO YOU WANT? 3
 15 + 15 =  ? 30
RIGHT!!
 3 + 4 =  ? 7
RIGHT!!
 5 + 14 =  ? 20
WRONG.  THE ANSWER IS 19
YOUR SCORE IS  66.6667 % RIGHT.
```

### Project *MULT*

*MULT* adds interest to the quiz by giving different responses to a correct answer (it makes use of ON K GOTO).

```
10 REM---MULT  (USES RANDOM REWARDS)-----------
100 REM---INTRODUCTION, INITIALIZE VARIABLES---
110 PRINT "MULTIPLICATION PRACTICE"
120 LET R = 0
130 PRINT "HOW MANY PROBLEMS DO YOU WANT";
140 INPUT N
200 REM---GENERATE PROBLEM--------------------
```

```
205 FOR I = 1 TO N
220   LET A = INT(RND(0)*20)+1
230   LET B = INT(RND(0)*10)+1
300   REM---PRESENT PROBLEM--------------------
310   PRINT A;"*";B;"=  ";
400   REM---INPUT ANSWER----------------------
410   INPUT X
500   REM---CHECK IF ANSWER IS RIGHT/WRONG-----
510   IF X = A * B THEN 540
520     PRINT "WRONG.  THE ANSWER IS";A*B
530     GOTO 610
540   LET Z = INT(RND(0)*3+1)
545   ON Z GOTO 550, 555, 560
550     PRINT "RIGHT!!"
551     GOTO 570
555     PRINT "VERY GOOD!!"
556     GOTO 570
560     PRINT "CORRECT!!"
570   LET R = R + 1
600 REM---GET NEXT PROBLEM---------------------
610 NEXT I
700 REM---REPORT SCORE------------------------
710 PRINT "YOUR SCORE IS ";R/N*100;"% RIGHT."
800 REM---FUTURE EXTENSION OF PROGRAM----------
900 END


>RUN
MULTIPLICATION PRACTICE
HOW MANY PROBLEMS DO YOU WANT? 3
 20 * 4 =  ? 80
RIGHT!!
 3 * 8 =  ? 27
WRONG.  THE ANSWER IS 24
 6 * 9 =  ? 54
VERY GOOD!!
YOUR SCORE IS  66.6667 % RIGHT.

>RUN
MULTIPLICATION PRACTICE
HOW MANY PROBLEMS DO YOU WANT? 3
 20 * 3 =  ? 60
CORRECT!!
 16 * 5 =  ? 80
RIGHT!!
 16 * 9 =  ? 144
VERY GOOD!!
YOUR SCORE IS  100 % RIGHT.
```

## Project *VARIETY1*

The next two programs, *VARIETY1* and *VARIETY2*, use the ON K GOSUB statement, as well as some interesting variations on RND. They reward "pretty close" answers as well as exact ones.

```
10  REM---VARIETY1  (MATH QUIZ  + - * / )-----
100 REM---INTRO., INITIALIZE VARIABLES--------
105 R=0: X=0
110 PRINT"HOW MANY PROBLEMS";
120 INPUT P
200 REM---CHOOSE + - * / ---------------------
210 PRINT"WHAT KIND?"
220 PRINT"  ADDITION = 1  SUBTRACTION = 2"
230 PRINT"  MULTIPLICATION = 3  DIVISION = 4  --";
240 INPUT K
300 REM---GENERATE PROBLEM--------------------
310 FOR I = 1 TO P
320   N1 = INT(RND(0) * 90 + 10)
330   N2 = INT(RND(0) * 9 + 1)
400 REM---PRESENT PROBLEM/INPUT ANSWER--------
410   ON K GOSUB 1000, 1100, 1200, 1300
420   INPUT N
500 REM---CHECK IF ANSWER IS RIGHT/WRONG------
510   IF ABS(N-A) < .1 THEN 540
520     PRINT"SORRY, THE ANSWER WAS";A
530     GOTO 560
540   PRINT"RIGHT!!!"
550   R=R+1
560 NEXT I
600 REM---PRINT SCORE-------------------------
610 PRINT"DONE.  YOU HAD";R;"CORRECT.  ";
700 REM---AGAIN?------------------------------
710 PRINT"WANT TO DO SOME MORE (YES=1)";
720 INPUT X
730 IF X=1 THEN 105
740 GOTO 9999
800 REM---FUTURE EXTENSION--------------------
1000 REM---SUBROUTINE FOR ADDITION------------
1010 PRINT N1;"+";N2;"=  ";
1020 A = N1 + N2
1030 RETURN
1100 REM---SUBROUTINE FOR SUBTRACTION---------
1110 PRINT N1;"-";N2;"=  ";
1120 A = N1 - N2
1130 RETURN
1200 REM---SUBROUTINE FOR MULTIPLICATION------
1210 PRINT N1;"*";N2;"=  ";
1220 A = N1 * N2
1230 RETURN
1300 REM---SUBROUTINE FOR DIVISION------------
1310 PRINT N1;"/";N2;"=  ";
1320 A = N1 / N2
1330 RETURN
9999 END
```

## Project *VARIETY2*

```
10  REM---VARIETY2  (MATH QUIZ  + - * / )-----
11  REM---(VARIABLE DIGITS, TOLERANCES)-------
100 REM---INTRO., INITIALIZE VARIABLES--------
105 R=0: X=0
110 PRINT"HOW MANY PROBLEMS";
120 INPUT P
200 REM---CHOOSE + - * / --------------------
210 PRINT"WHAT KIND?"
220 PRINT"  ADDITION = 1  SUBTRACTION = 2"
230 PRINT"  MULTIPLICATION = 3  DIVISION = 4  --";
240 INPUT K
250 PRINT"HOW MANY DIGITS IN THE TWO NUMBERS?"
260 PRINT"TYPE 2,1 FOR PROBLEMS LIKE 27 + 9  --";
270 INPUT D1, D2
280 D1 = 10[(D1-1)
290 D2 = 10[(D2-1)
300 REM---GENERATE PROBLEM-------------------
310 FOR I = 1 TO P
320   N1 = INT((RND(0) * 9 + 1) * D1)
330   N2 = INT((RND(0) * 9 + 1) * D2)
400 REM---PRESENT PROBLEM/INPUT ANSWER--------
410   ON K GOSUB 1000, 1100, 1200, 1300
420   INPUT N
500 REM---CHECK IF ANSWER IS RIGHT/WRONG------
510   IF ABS(N-A) < T THEN 540
520     PRINT"SORRY, THE ANSWER WAS";A
530     GOTO 560
540   PRINT"RIGHT!!!"
550   R=R+1
560 NEXT I
600 REM---PRINT SCORE------------------------
610 PRINT"DONE.  YOU HAD";R;"CORRECT.  ";
700 REM---AGAIN?-----------------------------
710 PRINT"WANT TO DO SOME MORE (YES=1)";
720 INPUT X
730 IF X=1 THEN 105
740 GOTO 9999
800 REM---FUTURE EXTENSION-------------------
1000 REM---SUBROUTINE FOR ADDITION-----------
1010 PRINT N1;"+";N2;"=  ";
1020 A = N1 + N2
1025 T = .01
1030 RETURN
1100 REM---SUBROUTINE FOR SUBTRACTION--------
1110 PRINT N1;"-";N2;"=  ";
1120 A = N1 - N2
1125 T = .01
1130 RETURN
1200 REM---SUBROUTINE FOR MULTIPLICATION------
1210 PRINT N1;"*";N2;"=  ";
1220 A = N1 * N2
1225 T = .05
1230 RETURN
1300 REM---SUBROUTINE FOR DIVISION-----------
1310 PRINT N1;"/";N2;"=  ";
1320 A = N1 / N2
1325 T = .1
1330 RETURN
9999 END
```

### Project *SQROOT*

This is a quiz on estimating square roots. It is actually a less complicated program than *VARIETY1* and *VARIETY2*. It rewards "pretty close" answers using a percentage error criterion.

```
10  REM---SQROOT  (QUIZ, ESTIMATING SQ. ROOTS)---
100 REM---INTRO., INITIALIZE VARIABLES-----------
110 PRINT"PRACTICE ESTIMATING SQUARE ROOTS"
120 PRINT"  TRY TO GET THREE ANSWERS IN A ROW"
130 PRINT"  WITH LESS THAN 5% ERROR EACH."
140 K=0: A$=""
200 REM---GENERATE PROBLEM-----------------------
210 Q = INT(RND(0) * 100 + 1)
220 R = SQR(Q)
300 REM---PRESENT PROBLEM/GET ANSWER-------------
310 PRINT"WHAT IS THE SQUARE ROOT OF";Q;"  ";
320 INPUT A
400 REM---CHECK IF ANSWER IS CLOSE---------------
410 IF ABS(A/R-1) < .01 THEN PRINT"  WOW!": K=K+1: GOTO 440
420 IF ABS(A/R-1) < .05 PRINT "  PRETTY GOOD": K=K+1: GOTO 440
430 PRINT"  WAY OFF-- > 5% ERROR": K=0
440 PRINT"  SQUARE ROOT WAS";R
450 PRINT USING "  YOU MISSED BY ##.## %";100*ABS(A/R-1)
460 IF K < 3 THEN 210
500 REM---SCORE----------------------------------
510 PRINT"CONGRATULATIONS, THAT'S 3 IN A ROW WITH < 5%"
600 REM---AGAIN?---------------------------------
610 PRINT"WANT TO TRY AGAIN (YES = Y)";
620 INPUT A$
630 IF A$ = "Y" THEN 140
999 END
```

## Solution to the Project in Section 2.2

### Project *OKPROG*

This is a deceptively simple program. It shows how to eliminate the error in BUGPROG.

```
10 REM ------------------ OKPROG ------------------
15 DEFINT A-Z     'THIS MAKES ALL VARIABLES INTEGERS
20 PRINT "SUM OF POSITIVE INTEGERS FROM 1 TO";
30 INPUT N
40 S=1 : I=1
50 IF I>=N THEN 90
60   I=I+1
70   S=S+I
80   GOTO 50
90 PRINT "SUM OF POSITIVE INTEGERS REQUESTED IS"; S


>RUN
RUN
SUM OF POSITIVE INTEGERS FROM 1 TO? 5
SUM OF POSITIVE INTEGERS REQUESTED IS 15
READY
>RUN
SUM OF POSITIVE INTEGERS FROM 1 TO? -6.8
SUM OF POSITIVE INTEGERS REQUESTED IS 1
READY
>RUN
SUM OF POSITIVE INTEGERS FROM 1 TO? 3.1416
SUM OF POSITIVE INTEGERS REQUESTED IS 6
READY
```

## Solutions to Projects in Section 2.3

### Project *GUESS2*

This is an improvement on GUESS 1, using the values input by the player to set the range of numbers to be guessed.

```
10  '**************************************************
11  '*          GUESS2  (VARIABLE RANGE)              *
12  '**************************************************
100 '--------------------------------------------------
101 '     COMPUTER CHOOSES NUMBER, PERSON GUESSES
102 '--------------------------------------------------
105 CLS: PRINT"GUESS MY NUMBER GAME"
120 PRINT"WHAT'S THE HIGHEST THE NUMBER CAN BE";: INPUT H
130 PRINT"WHAT'S THE LOWEST THE NUMBER CAN BE";: INPUT L
135 IF L >= H THEN PRINT"?????  ";: GOTO 120
140 IF INT(H)<>H OR INT(L)<>L PRINT"??? INTEGERS ONLY,PLEASE!":GOTO 120
150 T = 1: J = 1: K = 1
155 '-----FIND # OF GUESSES NEEDED----------------------
160 K = K * 2
170 IF K >= (H - L + 1) THEN T1 = J: GOTO 210
180 J = J + 1: GOTO 160
```

```
200 '-----GENERATE RANDOM NUMBER------------------------
210 X = INT(RND(0) * (H-L+1) + L)
220 PRINT"I'VE GOT A NUMBER BETWEEN";L;"AND";H
300 '-----ASK PLAYER TO GUESS--------------------------
310 IF T > T1 + 2 THEN 510
320   PRINT"YOUR GUESS IS--";: INPUT G
400 '-----TEST IF GUESS IS ILLEGAL, >, <, OR = ---------
410   IF G > H  PRINT "MUST BE <=";H : GOTO 460
420     IF G < L  PRINT "MUST BE >=";L: GOTO 460
430       IF G > X PRINT "NOPE, TOO HIGH.": GOTO 460
440         IF G < X PRINT "NOPE, TOO LOW.": GOTO 460
450           PRINT"YOU GOT IT IN";T;"TURNS!!!  ";: GOTO 610
460   T = T + 1: GOTO 310
500 '-----TOO MANY TURNS, GIVE ANSWER-------------------
510 PRINT"YOU HAVE HAD";T-1;"TURNS, THAT'S THE LIMIT."
520 PRINT"MY NUMBER WAS";X;".  ";
600 '-----AGAIN?----------------------------------------
610 PRINT"WANT TO PLAY AGAIN (Y = YES)";: A$=" ": INPUT A$
620 IF A$ = "Y" THEN 105
1000 '-----------------------------------------------------
1001 '     PERSON CHOOSES NUMBER, COMPUTER GUESSES
1002 '-----------------------------------------------------
1005 CLS: PRINT"O.K."
1010 PRINT"NOW I WANT TO GUESS YOUR NUMBER."
1020 PRINT"YOU PICK A NUMBER FROM";L;"TO";H
1030 PRINT"DON'T TELL ME WHAT IT IS!"
1040 PRINT"JUST TYPE H IF MY GUESS IS TOO HIGH,"
1050 PRINT"L IF MY GUESS IS TOO LOW, AND R IF I GET IT RIGHT."
1060 PRINT"I THINK I CAN GUESS IT IN";T1;"TURNS."
1100 '-----ASK PLAYER TO PICK NUMBER--------------------
1110 PRINT"GOT A NUMBER (Y = YES)";: A$=" ": INPUT A$
1120 IF A$ <> "Y" THEN 1530
1200 '-----GENERATE COMPUTER'S GUESS-------------------
1210 L1 = L: H1 = H: T = 0
1230 IF L1 > H1 THEN 1410
1240   G = INT((L1+H1)/2)
1250   PRINT"I GUESS...";G;", AM I -- H, L, OR R";
1260   T = T+1
1300 '-----GET CLUE, TEST IF H, L, R, OR ILLEGAL--------
1310   INPUT C$
1320   IF C$ = "H" THEN H1 = G-1: GOTO 1230
1330     IF C$ = "L" THEN L1 = G+1: GOTO 1230
1340       IF C$ = "R" PRINT"I GOT IT IN";T;"TURNS.":GOTO 1510
1350         PRINT"YOU DIDN'T TYPE H, L, OR R";:GOTO 1310
1400 '-----NOT FOUND (CLUE IS INCONSISTENT)-------------
1410 PRINT"DID YOU GIVE ME THE RIGHT CLUES?"
1420 PRINT"LET ME TRY AGAIN. ": GOTO 1210
1500 '-----AGAIN?--------------------------------------
1510 PRINT"WANT TO PLAY AGAIN (Y = YES)";: A$=" ": INPUT A$
1520 IF A$ = "Y" THEN 1110
1530 PRINT"O.K.  BYE NOW."
9999 END
```

### Project *SRCH1*

The following three programs, *SRCH1, SRCH2,* and *SRCH3* extend the idea of searches (both binary and sequential) into an experiment. The setting for the experiment is the application of computers to the analysis of text by searching for common words and counting their occurrence.

```
10  '************************************************
11  '*        SRCH1  (BINARY SRCH, INPUT STS.)      *
12  '************************************************
110 CLS
120 PRINT"***  WORD OCCURRENCE COUNTER  ***"
125 PRINT
130 PRINT"THIS PROGRAM COUNTS HOW MANY TIMES THE"
135 PRINT"COMMONEST WORDS OCCUR IN THE TEXT YOU TYPE IN.
140 PRINT
145 PRINT"PLEASE TYPE IN YOUR WORD(S), ONE AT A TIME."
155 PRINT"WHEN YOU ARE FINISHED, TYPE $$$."
160 PRINT"(OMIT ALL PUNCTUATION MARKS.)"
165 PRINT: GOSUB 1000
200 '------------------------------------------------
201 '           READ STANDARD LIST (WORD, RANK)
202 '------------------------------------------------
205 DIM A$(100), R(100), A(100)
210 FOR I = 1 TO 100
215   READ A$(I), R(I)
220   A(I) = 0
225 NEXT I
240 DATA A,4,          ABOUT,48,      AFTER,94,      ALL,33
242 DATA AN,39,        AND,3,         ARE,15,        AS,16
244 DATA AT,20,        BE,21,         BEEN,75,       BUT,31
246 DATA BY,27,        CALLED,96,     CAN,38,        COULD,70
248 DATA DID,83,       DO,45,         DOWN,84,       EACH,47
250 DATA FIND,87,      FIRST,74,      FOR,12,        FROM,23
252 DATA HAD,29,       HAS,62,        HAVE,25,       HE,11
254 DATA HER,64,       HIM,67,        HIS,18,        HOW,49
256 DATA I,24,         IF,44,         IN,6,          INTO,61
258 DATA IS,7,         IT,10,         ITS,76,        JUST,97
260 DATA KNOW,100,     LIKE,66,       LITTLE,92,     LONG,91
262 DATA MADE,81,      MAKE,72,       MANY,55,       MAY,89
264 DATA MORE,63,      MOST,99,       MY,80,         NO,71
266 DATA NOT,30,       NOW,78,        OF,2,          ON,14
268 DATA ONE,28,       ONLY,85,       OR,26,         OTHER,60
270 DATA OUT,51,       OVER,82,       PEOPLE,79,     SAID,43
272 DATA SEE,68,       SHE,54,        SO,57,         SOME,56
274 DATA THAN,73,      THAT,9,        THE,1,         THEIR,42
276 DATA THEM,52,      THEN,53,       THERE,37,      THESE,58
278 DATA THEY,19,      THIS,22,       TIME,69,       TO,5
280 DATA TWO,65,       UP,50,         USE,88,        VERY,93
282 DATA WAS,13,       WATER,90,      WAY,86,        WE,36
284 DATA WERE,34,      WHAT,32,       WHEN,35,       WHERE,98
286 DATA WHICH,41,     WHO,77,        WILL,46,       WITH,17
288 DATA WORDS,95,     WOULD,59,      YOU,8,         YOUR,40
300 '------------------------------------------------
301 '           INPUT/BINARY SEARCH LOOP
302 '------------------------------------------------
310 L = 1: H = 100
```

```
330 PRINT"WORD  ";:X$="": INPUT X$
340   IF X$ = "$$$" THEN 404
345   PRINT "'";X$;"' IS";
350   IF L > H PRINT " NOT FOUND ON STANDARD LIST.": GOTO 385
355     M = INT((L + H)/2)
360     IF A$(M) > X$ THEN H = M - 1: GOTO 350
365       IF A$(M) < X$ THEN L = M + 1: GOTO 350
370         A(M) = A(M) + 1: P$ = " WORD #" + STR$(M) + " "
380 PRINT P$;"ON STD.LIST";TAB(35);"USED";A(M);"TIME(S) IN YOUR TEXT."
385 GOTO 310
400 '---------------------------------------------------
401 '           PRINT SUMMARY:  WORD & TIMES USED
402 '---------------------------------------------------
404 C = 0
405 PRINT"***  SUMMARY  ***"
406 PRINT"DATA IS IN THE FOLLOWING ORDER:  THE WORD,"
407 PRINT"THE NUMBER OF TIMES YOU USED IT, ITS RANK"
408 PRINT"(FREQUENCY) IN THE STANDARD LIST."
409 GOSUB 1000
410 S$="%    %###  ###  %    %###  ###  %    %###  ###  %    %###  ###"
414 PRINT"WORD USED RANK WORD USED RANK WORD USED RANK WORD USED RANK"
415 FOR I = 1 TO 25
420 C=C+1: IF C/14 = INT(C/14) THEN GOSUB 1000
430 PRINT USING S$; A$(I),A(I),R(I), A$(I+25),A(I+25),R(I+25),
    A$(I+50),A(I+50),R(I+50), A$(I+75),A(I+75),R(I+75)
440 NEXT I
450 GOTO 9999
1000 '-------------------------------------------------
1001 '        SUBROUTINE, INTERRUPTS PRINTING
1002 '-------------------------------------------------
1010 PRINT"---CONTINUE";: INPUT D$: CLS: RETURN
9999 END
```

**Project** *SRCH2*

This program is similar to *SRCH1* except that it uses a sequential search instead of binary search.

```
10  '**************************************************
11  '*         SRCH2  (SEQUENTIAL SRCH, INPUT STS.)    *
12  '**************************************************
110 CLS
120 PRINT"***  WORD OCCURRENCE COUNTER  ***"
125 PRINT
130 PRINT"THIS PROGRAM COUNTS HOW MANY TIMES THE"
135 PRINT"COMMONEST WORDS OCCUR IN THE TEXT YOU TYPE IN.
140 PRINT
145 PRINT"PLEASE TYPE IN YOUR WORD(S), ONE AT A TIME."
155 PRINT"WHEN YOU ARE FINISHED, TYPE $$$."
160 PRINT"(OMIT ALL PUNCTUATION MARKS.)"
165 PRINT: GOSUB 1000
```

```
200 '-----------------------------------------------
201 '          READ STANDARD LIST (WORD, RANK)
202 '-----------------------------------------------
205 DIM A$(100), R(100), A(100)
210 FOR I = 1 TO 100
215   READ A$(I), R(I)
220   A(I) = 0
225 NEXT I
240 DATA A,4,        ABOUT,48,     AFTER,94,     ALL,33
242 DATA AN,39,      AND,3,        ARE,15,       AS,16
244 DATA AT,20,      BE,21,        BEEN,75,      BUT,31
246 DATA BY,27,      CALLED,96,    CAN,38,       COULD,70
248 DATA DID,83,     DO,45,        DOWN,84,      EACH,47
250 DATA FIND,87,    FIRST,74,     FOR,12,       FROM,23
252 DATA HAD,29,     HAS,62,       HAVE,25,      HE,11
254 DATA HER,64,     HIM,67,       HIS,18,       HOW,49
256 DATA I,24,       IF,44,        IN,6,         INTO,61
258 DATA IS,7,       IT,10,        ITS,76,       JUST,97
260 DATA KNOW,100,   LIKE,66,      LITTLE,92,    LONG,91
262 DATA MADE,81,    MAKE,72,      MANY,55,      MAY,89
264 DATA MORE,63,    MOST,99,      MY,80,        NO,71
266 DATA NOT,30,     NOW,78,       OF,2,         ON,14
268 DATA ONE,28,     ONLY,85,      OR,26,        OTHER,60
270 DATA OUT,51,     OVER,82,      PEOPLE,79,    SAID,43
272 DATA SEE,68,     SHE,54,       SO,57,        SOME,56
274 DATA THAN,73,    THAT,9,       THE,1,        THEIR,42
276 DATA THEM,52,    THEN,53,      THERE,37,     THESE,58
278 DATA THEY,19,    THIS,22,      TIME,69,      TO,5
280 DATA TWO,65,     UP,50,        USE,88,       VERY,93
282 DATA WAS,13,     WATER,90,     WAY,86,       WE,36
284 DATA WERE,34,    WHAT,32,      WHEN,35,      WHERE,98
286 DATA WHICH,41,   WHO,77,       WILL,46,      WITH,17
288 DATA WORDS,95,   WOULD,59,     YOU,8,        YOUR,40
300 '-----------------------------------------------
301 '          INPUT/SEQUENTIAL SEARCH LOOP
302 '-----------------------------------------------
330 PRINT"WORD  ";:X$="": INPUT X$
340   IF X$ = "$$$" THEN 404
345   PRINT "'";X$;"' IS";
350   FOR M = 1 TO 100
355     IF X$ = A$(M) THEN 370
360   NEXT M
365   PRINT" NOT FOUND ON STANDARD LIST.": GOTO 385
370   A(M) = A(M) + 1: P$ = " WORD #" + STR$(M) + " "
380   PRINT P$;"ON STD.LIST";TAB(35);"USED";A(M);"TIME(S)IN YOUR TEXT."
385 GOTO 330
400 '-----------------------------------------------
401 '          PRINT SUMMARY:  WORD & TIMES USED
402 '-----------------------------------------------
404 C = 0
405 PRINT"***  SUMMARY  ***"
406 PRINT"DATA IS IN THE FOLLOWING ORDER:  THE WORD,"
407 PRINT"THE NUMBER OF TIMES YOU USED IT, ITS RANK"
408 PRINT"(FREQUENCY) IN THE STANDARD LIST."
409 GOSUB 1000
410 S$="%    %###  ###  %    %###  ###  %    %###  ###  %    %###  ###"
414 PRINT"WORD USED RANK WORD USED RANK WORD USED RANK WORD USED RANK"
415 FOR I = 1 TO 25
420 C=C+1: IF C/14 = INT(C/14) THEN GOSUB 1000
430 PRINT USING S$; A$(I),A(I),R(I), A$(I+25),A(I+25),R(I+25),
    A$(I+50),A(I+50),R(I+50), A$(I+75),A(I+75),R(I+75)
440 NEXT I
```

```
450 GOTO 9999
1000 '-----------------------------------------------------
1001 '        SUBROUTINE, INTERRUPTS PRINTING
1002 '-----------------------------------------------------
1010 PRINT"---CONTINUE";: INPUT D$: CLS: RETURN
9999 END
```

**Project** *SRCH3*

This program is similar to *SRCH1* except that it uses DATA statements for storing the user's text.

```
10   '**************************************************
11   '*        SRCH3  (BINARY SRCH, DATA STS.)         *
12   '**************************************************
110 CLS
120 PRINT"***  WORD OCCURRENCE COUNTER  ***"
125 PRINT
130 PRINT"THIS PROGRAM COUNTS HOW MANY TIMES THE COMMONEST"
135 PRINT"WORDS OCCUR IN THE TEXT IN THE DATA STATEMENTS."
140 GOSUB 1000
150 CLS
200 '--------------------------------------------------
201 '          READ STANDARD LIST (WORD, RANK)
202 '--------------------------------------------------
205 DIM A$(100), R(100), A(100)
210 FOR I = 1 TO 100
215   READ A$(I), R(I)
220   A(I) = 0
225 NEXT I
240 DATA A,4,          ABOUT,48,       AFTER,94,       ALL,33
242 DATA AN,39,        AND,3,          ARE,15,         AS,16
244 DATA AT,20,        BE,21,          BEEN,75,        BUT,31
246 DATA BY,27,        CALLED,96,      CAN,38,         COULD,70
248 DATA DID,83,       DO,45,          DOWN,84,        EACH,47
250 DATA FIND,87,      FIRST,74,       FOR,12,         FROM,23
252 DATA HAD,29,       HAS,62,         HAVE,25,        HE,11
254 DATA HER,64,       HIM,67,         HIS,18,         HOW,49
256 DATA I,24,         IF,44,          IN,6,           INTO,61
258 DATA IS,7,         IT,10,          ITS,76,         JUST,97
260 DATA KNOW,100,     LIKE,66,        LITTLE,92,      LONG,91
262 DATA MADE,81,      MAKE,72,        MANY,55,        MAY,89
264 DATA MORE,63,      MOST,99,        MY,80,          NO,71
266 DATA NOT,30,       NOW,78,         OF,2,           ON,14
268 DATA ONE,28,       ONLY,85,        OR,26,          OTHER,60
270 DATA OUT,51,       OVER,82,        PEOPLE,79,      SAID,43
272 DATA SEE,68,       SHE,54,         SO,57,          SOME,56
274 DATA THAN,73,      THAT,9,         THE,1,          THEIR,42
276 DATA THEM,52,      THEN,53,        THERE,37,       THESE,58
278 DATA THEY.19,      THIS,22,        TIME,69,        TO,5
```

```
280 DATA TWO,65,        UP,50,          USE,88,         VERY,93
282 DATA WAS,13,        WATER,90,       WAY,86,         WE,36
284 DATA WERE,34,       WHAT,32,        WHEN,35,        WHERE,98
286 DATA WHICH,41,      WHO,77,         WILL,46,        WITH,17
288 DATA WORDS,95,      WOULD,59,       YOU,8,          YOUR,40
300 '-----------------------------------------------------
301 '            INPUT/BINARY SEARCH LOOP
302 '-----------------------------------------------------
310 L = 1: H = 100
320 C=C+1: IF C/14 = INT(C/14) THEN GOSUB 1000
325   READ X$
340   IF X$ = "$$$" THEN 404
345   PRINT "'";X$;"'";TAB(11)" IS";
350   IF L > H PRINT " NOT FOUND ON STANDARD LIST.": GOTO 385
355     M = INT((L + H)/2)
360     IF A$(M) > X$ THEN H = M - 1: GOTO 350
365       IF A$(M) < X$ THEN L = M + 1: GOTO 350
370         A(M) = A(M) + 1: P$ = " WORD #" + STR$(M) + " "
380   PRINT P$;"ON STD.LIST";TAB(45);"USED";A(M);"TIME(S)."
385 GOTO 310
400 '-----------------------------------------------------
401 '            PRINT SUMMARY:  WORD & TIMES USED
402 '-----------------------------------------------------
404 C = 0
405 PRINT"***  SUMMARY  ***"
406 PRINT"DATA IS IN THE FOLLOWING ORDER:  THE WORD,"
407 PRINT"THE NUMBER OF TIMES YOU USED IT, ITS RANK"
408 PRINT"(FREQUENCY) IN THE STANDARD LIST."
409 GOSUB 1000
410 S$="%    %###  ###  %    %###  ###  %    %###  ###  %    %###  ###"
414 PRINT"WORD USED RANK WORD USED RANK WORD USED RANK WORD USED RANK"
415 FOR I = 1 TO 25
420 C=C+1: IF C/14 = INT(C/14) THEN GOSUB 1000
430 PRINT USING S$; A$(I),A(I),R(I), A$(I+25),A(I+25),R(I+25),
 A$(I+50),A(I+50),R(I+50), A$(I+75),A(I+75),R(I+75)
440 NEXT I
450 GOTO 9999
500 '-----------------------------------------------------
501 '             TEXT TO BE ANALYZED
502 '-----------------------------------------------------
510 DATA WHEN,IN,THE,COURSE,OF,HUMAN,EVENTS,IT,BECOMES
511 DATA NECESSARY,FOR,ONE,PEOPLE,TO,DISSOLVE,THE
512 DATA POLITICAL,BANDS,WHICH,HAVE,CONNECTED,THEM,WITH
513 DATA ANOTHER,AND,TO,ASSUME,AMONG,THE,POWERS,OF,THE
514 DATA EARTH,THE,SEPARATE,AND,EQUAL,STATION,TO,WHICH
515 DATA THE,LAWS,OF,NATURE,AND,OF,NATURE'S,GOD,ENTITLE
516 DATA THEM,A,DECENT,RESPECT,TO,THE,OPINIONS,OF
517 DATA MANKIND,REQUIRES,THAT,THEY,SHOULD,DECLARE,THE
518 DATA CAUSES,WHICH,IMPEL,THEM,TO,THE,SEPARATION
519 DATA WE,HOLD,THESE,TRUTHS,TO,BE,SELF,EVIDENT,THAT
520 DATA ALL,MEN,ARE,CREATED,EQUAL,THAT,THEY,ARE
521 DATA ENDOWED,BY,THEIR,CREATOR,WITH,CERTAIN
522 DATA UNALIENABLE,RIGHTS,THAT,AMONG,THESE,ARE,LIFE
523 DATA LIBERTY,AND,THE,PURSUIT,OF,HAPPINESS,$$$
590 GOTO 9999
1000 '----------------------------------------------------
1001 '        SUBROUTINE, INTERRUPTS PRINTING
1002 '----------------------------------------------------
1010 PRINT"---CONTINUE";: INPUT D$: CLS: RETURN
9999 END
```

## Solutions to Projects in Section 2.4

### Project *SCRAM2*

This program is an improvement on *SCRAM1* with fresh data and several added features.

```
10  '**************************************************
11  '*     SCRAM2  (WORD QUIZ)                        *
12  '**************************************************
100 '--------------------------------------------------
101 '     INITIALIZE, GIVE INSTRUCTIONS
102 '--------------------------------------------------
105 CLS: DIM P(15)
110 PRINT"    *** SCRAMBLED WORD GAME TWO ***"
120 PRINT"TYPE THE WORD WHOSE LETTERS ARE SCRAMBLED"
122 PRINT"ON THE SCREEN.  NOTICE THE CLUE."
125 PRINT
128 PRINT"PRESS ENTER TO CONTINUE";: INPUT D$: CLS
130 PRINT"HOW MANY WORDS DO YOU WANT";: N=0: INPUT N
135   IF N < 1 THEN 130
140 CLS
150 W=0: R=0: F=0
200 '--------------------------------------------------
201 '     GENERATE QUIZ WORD
202 '--------------------------------------------------
205 FOR I = 1 TO N
207   RESTORE
210   FOR X = 1 TO INT(20*RND(0)+1)     'WILL START AT
215     READ D$,D$,D$                   'RANDOM WORD
220   NEXT X                            'EACH WORD
225   READ W$, C$, S$
230   PRINT"CLUE IS:  ";C$
235   FOR Z = 1 TO 300: NEXT Z                  'WAIT
237   IF F=1 THEN CLS: PRINT CHR$(23)   'BIG LETTERS
240 '-----PRINT EACH LETTER IN A RANDOM POSITION-------
245   FOR J = 1 TO LEN(S$)
250     P(J) = INT((RND(0) * (64 * 10) + (64 * 6))/2) * 2
252 '-----CHECK IF POS. SAME AS ANY PREVIOUS POS.------
255     IF J = 1 THEN 275
260       FOR K = 1 TO J - 1
265         IF P(J) = P(K) THEN 250
270       NEXT K
275     PRINT @ P(J), MID$(S$,J,1);
280     FOR Z = 1 TO 200: NEXT Z                'WAIT
285   NEXT J
290   FOR Z = 1 TO 300: NEXT Z: CLS             'WAIT
300 '--------------------------------------------------
301 '     GET ANSWER, CHECK IF RIGHT/WRONG
302 '--------------------------------------------------
310   PRINT"WHAT IS YOUR ANSWER";: A$=" ": INPUT A$
320   IF A$=W$ PRINT "RIGHT!": F=0: R=R+1: GOTO 360
325     PRINT"WRONG."
330     IF F=1 THEN F=0: W=W+1: GOTO 355
340       F=1: PRINT"TRY AGAIN.": GOTO 235
355     PRINT"THE WORD WAS:  ";W$
```

```
360  PRINT"PRESS ENTER TO CONTINUE";: INPUT D$: CLS
370 NEXT I
400 '-------------------------------------------------
401 '     END OF GAME, DISPLAY SCORE
402 '-------------------------------------------------
420 PRINT"(YOU GOT";R;"RIGHT ANSWERS, AND";W;"WRONG.)"
430 PRINT"WANT TO PLAY AGAIN (Y = YES)";:A$=" ":INPUT A$
440 IF A$="Y" THEN 130
450 PRINT"SO LONG.": GOTO 999
500 '-------------------------------------------------
501 '    DATA:  WORDS, CLUES, SCRAMBLES
502 '-------------------------------------------------
510 DATA X,X,X
515 DATA GRANDILOQUENT, BOMBASTIC, NUOINRGADLQET
520 DATA KOALA, TEDDY BEAR, AAKLO
525 DATA PITTANCE, DOLE, CATPENTI
530 DATA ATTAIN, ACHIEVE, ITATAN
535 DATA SERMON, HOMILY, NMEORS
540 DATA SHACKLE, TO HOG-TIE, EKASLCH
545 DATA BACKWARD, RETROGRADE, DAKARWCB
550 DATA SENILE, DODDERING, EIELNS
555 DATA TRITE, STEREOTYPED, RETIT
560 DATA AMATEUR, DABBLER, RETAAMU
565 DATA COLLECT, AMASS, TELOCLCT
570 DATA ALWAYS, FOREVER, YAWALS
575 DATA FANTASTIC, CHIMERICAL, CTANFATSI
580 DATA CORPULENT, OBESE, TEURCNLPO
585 DATA PICKLE, DILL ------, EKILCP
590 DATA INVENT, CREATE, TENNVI
595 DATA PLEASE, ------ & THANK YOU, EELPAS
600 DATA WISE, SAGE, SIEW
605 DATA SWEAR, AFFIRM, RESAW
610 DATA TURBID, MUDDY, DIBRUT
999 END
```

**Project** *SCRAM3*

*SCRAM3* extends the ideas in SCRAM2 using sentences instead of words as data.

```
10  '**************************************************
11  '*     SCRAM3  (PROVERB QUIZ)                     *
12  '**************************************************
100 '-------------------------------------------------
101 '     INITIALIZE, GIVE INSTRUCTIONS
102 '-------------------------------------------------
103 CLEAR 100
105 CLS: DIM P(15), W$(15), S(15), A$(15)
110 PRINT"    *** SCRAMBLED PROVERB GAME ***"
120 PRINT"TYPE THE PROVERB WHOSE WORDS ARE SCRAMBLED"
122 PRINT"ON THE SCREEN.  IT IS AN OLD FAMILIAR SAYING."
```

```
128 PRINT: PRINT"PRESS ENTER TO CONTINUE";: INPUT D$: CLS
130 PRINT"HOW MANY PROVERBS DO YOU WANT";: N=0: INPUT N
135   IF N < 1 THEN 130
140 CLS: C=0: F=0
200 '-----------------------------------------------------
201 '     GENERATE QUIZ PROVERB
202 '-----------------------------------------------------
205 FOR X = 1 TO INT(150 * RND(0)+1)
206   READ D$                           'WILL START AT
207   IF D$ = "$$$" THEN RESTORE        'RANDOM PROVERB
208 NEXT X                              'EACH GAME
210 READ D$
211   IF D$ = "$$$" THEN RESTORE
212   IF D$ <> "$" THEN 210
214 '-----READ N PROVERBS----------------------------
215 FOR I = 1 TO N
220   J = 1
222   READ W$(J)
224     IF W$(J) = "$$$" THEN RESTORE: GOTO 222
226     IF W$(J) <> "$" THEN J=J+1: GOTO 222
227   J = J - 1
229 '-----CREATE A RANDOM ORDERING FOR WORDS-----------
230   FOR K = 1 TO J: S(K) = 0: NEXT K
235   FOR K = 1 TO J
237     R = INT(RND(0) * J + 1)
240     IF S(R) = 0 THEN S(R) = K: GOTO 245
242       R = R + 1: IF R > J THEN R = 1
244       GOTO 240
245   NEXT K
249 '-----DISPLAY THE WORDS SCATTERED RANDOMLY---------
250   FOR K = 1 TO J
255     P(K) = INT((RND(0) * (64 * 10) + (64 * 5))/16) * 16
260     IF K = 1 THEN 280
265       FOR L = 1 TO K - 1            'CHECK IF
270         IF P(K) = P(L) THEN 255     'SAME POSITION
275       NEXT L                        'USED BEFORE
280     PRINT @ P(K), W$(S(K));
285   NEXT K
300 '-----------------------------------------------------
301 '     GET ANSWER, CHECK IF RIGHT/WRONG
302 '-----------------------------------------------------
305   PRINT @ 0, "WHAT IS YOUR ANSWER?"
306   PRINT"(TYPE IT IN ONE WORD AT A TIME, FOLLOWING EACH ?)"
310   FOR K = 1 TO J
315     INPUT A$(K)
320     IF A$(K) <> W$(K) THEN 330
325   NEXT K
327   PRINT"RIGHT!!": F=0: C=C+1: GOTO 385
330   PRINT"WRONG."
335   IF F = 0 THEN F = 1: PRINT"TRY AGAIN.": GOTO 305
340     F = 0: PRINT"THE ANSWER IS:"
345     FOR K = 1 TO J: PRINT W$(K);" ";: NEXT K
350     PRINT
385   PRINT"PRESS ENTER TO CONTINUE";: INPUT D$: CLS
395 NEXT I
400 '-----------------------------------------------------
401 '     END OF GAME, DISPLAY SCORE
402 '-----------------------------------------------------
410 PRINT"YOU GOT";C;"RIGHT OUT OF";N;"PROVERB(S)."
430 PRINT"WANT TO PLAY AGAIN (Y = YES)";:A$=" ":INPUT A$
440 IF A$="Y" THEN 130
450 PRINT"SO LONG.": GOTO 999
```

```
500 '------------------------------------------------
501 '     DATA:  PROVERBS
502 '------------------------------------------------
510 DATA A, BAD, PENNY, ALWAYS, COMES, BACK, $
515 DATA A, BARKING, DOG, NEVER, BITES, $
520 DATA ABSENCE, MAKES, THE, HEART, GROW, FONDER, $
525 DATA ALL, WORK, AND, NO, PLAY, MAKES, JACK, A, DULL, BOY, $
530 DATA A, STITCH, IN, TIME, SAVES, NINE, $
535 DATA BETTER, THE, LEADER, IN, A, VILLAGE, THAN, SECOND
540 DATA IN, ROME, $
545 DATA BREVITY, IS, THE, SOUL, OF, WIT, $
550 DATA CONSIDER, THE, ANT, THOU, SLUGGARD, AND, BE, WISE, $
555 DATA CLEANLINESS, IS, NEXT, TO, GODLINESS, $
560 DATA DO, UNTO, OTHERS, AS, YOU, WOULD, HAVE, THEM
565 DATA DO, UNTO, YOU, $
570 DATA DON'T, TEACH, YOUR, GRANDMOTHER, TO, SUCK, EGGS, $
575 DATA EVERY, CLOUD, HAS, A, SILVER, LINING, $
580 DATA EVIL, TO, HIM, WHO, EVIL, THINKS, $
585 DATA FINE, WORDS, BUTTER, NO, PARSNIPS, $
590 DATA FINGERS, WERE, MADE, BEFORE, FORKS, $
595 DATA FOOLS, RUSH, IN, WHERE, ANGELS, FEAR, TO, TREAD, $
600 DATA GOD, HELPS, THOSE, WHO, HELP, THEMSELVES, $
605 DATA HE, WHO, FIGHTS, AND, RUNS, AWAY, LIVES, TO, FIGHT
610 DATA ANOTHER, DAY, $
615 DATA LIE, DOWN, WITH, DOGS, GET, UP, WITH, FLEAS, $
620 DATA NEVER, MARRY, A, WIDOW, UNLESS, HER, FIRST
625 DATA HUSBAND, WAS, HANGED, $
630 DATA $$$
999 END
```

### Project *CIPHER*

This secret code game uses a cipher, or mathematical transformation, for disguising the message.

```
10   '******************************************************
11   '*              CIPHER  (SECRET CODE GAME)            *
12   '*      NOTE:  THE WORD 'CODE' IS USED IN THIS GAME   *
13   '*      TO MEAN THE SAME AS 'CIPHER'.  FOR AN EX-     *
14   '*      PLANATION OF THE DIFFERENCE, SEE SECTION 2.4. *
15   '******************************************************
100 '------------------------------------------------------
101 '     INITIALIZE, GIVE INSTRUCTIONS
102 '------------------------------------------------------
104 CLS: CLEAR 900
108 PRINT"    *** DECODE THE SECRET MESSAGE ***"
110 PRINT"DO YOU NEED INSTRUCTIONS (YES/NO)";:A$=" ":INPUT A$
111 IF LEFT$(A$,1) <> "Y" THEN 204
112 PRINT: PRINT"   HERE IS A CODED MESSAGE:  Q HUT QFFBU"
114 PRINT"        HERE IT IS DECODED:  A RED APPLE": PRINT
118 PRINT"PRESS ENTER TO SEE WHAT THE CODE WAS";: INPUT D$
```

```
120 PRINT"*      * *           *        *   *"
122 PRINT"A B C D E F G H I J K L M N O P Q R S T U V W X Y Z"
124 FOR I = 1 TO 26: PRINT CHR$(92);" ";: NEXT I: PRINT
126 PRINT"Q R S T U V W X Y Z A B C D E F G H I J K L M N O P"
128 PRINT: FOR Z = 1 TO 300: NEXT Z
130 PRINT"  IN THIS GAME, NEW CODES ARE MADE BY 'SHIFTING'"
132 PRINT"  THE ENCODING ALPHABET TO A NEW POSITION"
134 PRINT: PRINT"PRESS ENTER TO SEE A NEW CODE";: INPUT D$
138 PRINT: FOR I = 65 TO 90: PRINT CHR$(I);" ";: NEXT I
140 PRINT: FOR I = 1 TO 26: PRINT CHR$(92);" ";: NEXT I: PRINT
142 SH = 4
144 FOR I = 65 TO 90
145   X = I + SH
146   IF X > 90 THEN X = X - 26
148   PRINT CHR$(X);" ";
150 NEXT I: PRINT
152 PRINT: PRINT"PRESS ENTER TO CONTINUE";: INPUT D$: CLS
154 PRINT"   TO PLAY THIS GAME YOU MUST--
156 PRINT"      1.  LOOK AT THE CODED MESSAGE ON THE SCREEN."
158 PRINT"      2.  PICK ONE LETTER AND FIGURE OUT WHAT"
160 PRINT"             LETTER WAS ENCODED TO GET IT."
162 PRINT: PRINT"PRESS ENTER TO CONTINUE";: INPUT D$: PRINT
166 PRINT"   HERE ARE SOME CLUES:"
168 PRINT"      --ALL THE MESSAGES ARE OLD FAMILIAR SAYINGS."
170 PRINT"      --SOME ONE-LETTER WORDS IN THEM ARE: I, A."
172 PRINT"      --SOME TWO-LETTER WORDS IN THEM ARE:"
174 PRINT"            TO, AT, IS, IN, IT, OF."
176 PRINT"      --SOME THREE-LETTER WORDS IN THEM ARE:"
178 PRINT"            AND, ALL, BUT, NOT, THE."
180 PRINT: PRINT"PRESS ENTER TO CONTINUE";: INPUT D$: CLS
185 PRINT"   YOUR SCORE WILL DEPEND ON HOW MANY MESSAGES"
187 PRINT"   YOU NEED TO SEE BEFORE DISCOVERING THE CODE"
189 PRINT"   (1 = 100%  2 = 50%  3 = 33%  4+ = 0)."
190 PRINT: PRINT"PRESS ENTER TO START THE GAME";: INPUT D$
200 '-----------------------------------------------------
201 '     READ & CODE MESSAGE
202 '-----------------------------------------------------
204 CLS
205 FOR X = 1 TO INT(30 * RND(0) + 1)
206   READ D$                            'PICK A
207   IF D$ = "$$$" THEN RESTORE         'RANDOM
208 NEXT X                               'MESSAGE
210 READ D$:IF D$="$$$" RESTORE
211   IF D$ <> "$" THEN 210
212 '-----NEW RANDOM SHIFT----------------------------
213 C = 0: R = INT(RND(0) * 25 + 1)
214 '-----READ  MESSAGE-------------------------------
215 C = C + 1
220 M$(C) = "": CM$(C) = ""
225 READ DT$
230   IF DT$="$$$" THEN RESTORE: GOTO 225
235   IF DT$ <> "$" THEN M$(C) = M$(C) + DT$: GOTO 225
238 '-----CREATE CODED MESSAGE------------------------
240 FOR I = 1 TO LEN(M$(C))
245   CR$ = MID$(M$(C),I,1)
249   PRINT @ 55, I; CR$
250   IF ASC(CR$)>90 OR ASC(CR$)<65 THEN CM$(C)=CM$(C)+CR$: GOTO 265
255     NC = ASC(CR$) + R
257     IF NC > 90 THEN NC = NC - 26
260     CM$(C) = CM$(C) + CHR$(NC)
265 NEXT I
```

```
300 '--------------------------------------------------------
301 '     DISPLAY CODED MESSAGE, GET INPUT, DECODE MESSAGE
302 '--------------------------------------------------------
304 CLS: PRINT"HERE IS THE CODED MESSAGE:"
305 PRINT: PRINT CM$(C): PRINT
306 PRINT"   WHAT LETTER DO YOU THINK YOU CAN DECODE";
307 X$ = " ": INPUT X$
308 PRINT"   WHAT LETTER DO YOU THINK IT STANDS FOR";
310 Y$ = " ": INPUT Y$
311 IF ASC(X$)>=65 AND ASC(X$)<=90 AND ASC(Y$)>=65 AND ASC(Y$)<=90 THEN 314
312   PRINT"????": GOTO 306
313 '-----CREATE & PRINT DECODED MESSAGE---------------
314 Q = ASC(X$) - ASC(Y$): DM$ = ""
315 FOR I = 1 TO LEN(CM$(C))
316   CR$ = MID$(CM$(C),I,1)
317   PRINT @ 55, I; CR$
318   IF ASC(CR$)>90 OR ASC(CR$)<65 THEN DM$=DM$+CR$:GOTO 326
320     DC = ASC(CR$) - Q
322     IF DC < 65 THEN DC = DC + 26
323     IF DC > 90 THEN DC = DC - 26
324     DM$ = DM$ + CHR$(DC)
326 NEXT I
328 CLS: PRINT"THE DECODED MESSAGE IS:"
330 PRINT: PRINT DM$: PRINT
331 '-----COMPARE MESSAGES----------------------------
332 IF M$(C) = DM$ THEN PRINT"CONGRATULATIONS.": GOTO 404
334   PRINT"SORRY, THAT'S NOT IT."
336   PRINT"WOULD YOU LIKE TO ..."
337   PRINT"  1. TRY DECODING THE MESSAGE AGAIN."
338   PRINT"  2. TRY DECODING A DIFFERENT MESSAGE (SAME CODE)."
339   PRINT"  3. GIVE UP ON THIS CODE.
340   PRINT: PRINT"TYPE 1, 2, OR 3";: A = 0: INPUT A
345   ON A GOTO 304, 215, 416
350   IF A < 1 OR A > 3 THEN 336
400 '--------------------------------------------------------
401 '     END OF GAME, DISPLAY SCORE, MESSAGES
402 '--------------------------------------------------------
404 IF C > 3 THEN S = 0 ELSE S = INT(1/C * 100)
406 PRINT"   YOU HAVE BROKEN THE CODE AFTER RECEIVING"
408 PRINT"   ";C;"MESSAGE(S)."
416 IF A = 3 THEN S = 0: A = 0
418 PRINT"   YOUR SCORE IS";S;"%."
420 IF C < 2 THEN 455
422   PRINT"WANT TO SEE THE MESSAGES AND CODED MESSAGES"
423   PRINT"YOU RECEIVED IN THIS CODE (Y/N)";:A$=" ":INPUT A$
424   IF LEFT$(A$,1) <> "Y" THEN 455
425   FOR I = 1 TO C
430     PRINT M$(I): PRINT CM$(I): PRINT
435   NEXT I: PRINT
455 PRINT"WANT TO SEE ANOTHER MESSAGE--WITH A NEW CODE";
460 A$ = " ": INPUT A$
465 IF LEFT$(A$,1) = "Y" THEN 204
470   GOTO 999
500 '--------------------------------------------------------
501 '     DATA:  PROVERBS
502 '--------------------------------------------------------
505 DATA "IT'S BETTER TO HAVE LOVED AND LOST THAN NEVER TO"
510 DATA " HAVE LOVED AT ALL", $
520 DATA "KEEP YOUR EYES OPEN BEFORE MARRIAGE, HALF SHUT"
525 DATA " AFTERWARDS", $
530 DATA "MAN WORKS FROM SUN TO SUN, WOMAN'S WORK IS"
535 DATA " NEVER DONE", $
```

```
540 DATA NOTHING IS CERTAIN IN THIS WORLD BUT DEATH AND TAXES,$
545 DATA "OH, WHAT A TANGLED WEB WE WEAVE WHEN FIRST WE"
550 DATA " PRACTICE TO DECEIVE", $
555 DATA "PEOPLE WHO LIVE IN GLASS HOUSES SHOULD NOT THROW"
560 DATA " STONES", $
565 DATA "TAKE CARE OF THE PENNIES, THE DOLLARS WILL TAKE"
570 DATA " CARE OF THEMSELVES", $
575 DATA "WHEN A MAN IS WRAPPED UP IN HIMSELF,"
580 DATA " THE PACKAGE IS SMALL", $
585 DATA "YOU CAN CATCH MORE FLIES WITH HONEY THAN WITH"
590 DATA " VINEGAR", $
600 DATA "YOU CAN LEAD A HORSE TO WATER,"
605 DATA " BUT YOU CAN'T MAKE IT DRINK", $
610 DATA $$$
999 PRINT"    *** END OF SECRET CODE GAME ***": END
```

## Solution to Project in Section 2.5

### Project *BABYZAP*

This is an improved version of *BABYQ*. It has more hazards than *BABYQ* in the form of L-mines.

```
10 '**********************************************************
11 '*       BABYZAP  (BABYQ WITH L-MINE RAY WEAPONS)        *
12 '**********************************************************
20 CLS
100 '---------------------------------------------------------
101 '           INITIALIZE VARIABLES
102 '---------------------------------------------------------
110 PRINT"*** BABY ZAP RUNNING -- STAND BY ***"
120 DIM A$(9,9)
130 FOR I = 1 TO 9
140   FOR J = 1 TO 9
150     LET A$(I,J) = "-"
160   NEXT J
170 NEXT I
180 XE = 1: YE = 1: TF = 0: EE = 70
190 A$(9,9) = "G": A$(9,8) = "*": A$(8,9) = "*": A$(8,8) = "*"
200 '=========================================================
201 '           MAIN LOOP STARTS HERE
202 '
210 TF = TF + 1: CLS
215 IF RND(0) < 0.3 THEN A$(9,9) = " " ELSE A$(9,9) = "G"
220 A$(XE,YE) = "E"
300 '
301 '----------DISPLAY UNIVERSE----------------------------
310 FOR Y = 9 TO 1 STEP -1: PRINT Y;"  ";
320   FOR X = 1 TO 9
```

```
330     PRINT A$(X,Y);"    ";
340   NEXT X
350   PRINT
360 NEXT Y
370 PRINT"      1    2    3    4    5    6    7    8    9 ";
380 PRINT TAB(43);"EE =";INT(EE);"  TF =";TF
400 '
401 '----------INPUT & PROCESS E MOVE----------------------
410 PRINT"YOUR MOVE (MAX. DIST. = EE/100):   X,Y =";:INPUT X,Y
420 IF X<1 OR X>9 OR Y<1 OR Y>9 PRINT"ILLEGAL COORD.":GOTO 410
430 DE = SQR((X-XE)[2 + (Y-YE)[2)
440 IF DE <= EE/100 THEN 510
450 PRINT"INSUFFICIENT ENERGY, DISTANCE REQUESTED =";DE
460 GOTO 410
500 '
501 '----------MOVE E SHIP; CHECK FOR GATE COLLISION-------
510 A$(XE,YE) = "-"
520 XE = INT(X): YE = INT(Y)
530 IF A$(XE,YE) = "*" THEN 1000
540 IF XE = 9 AND YE = 9 THEN 2000
550 IF DE = 0 PRINT"RESTING: ENERGY INCREASED 10%":EE = 1.1*EE
600 '
601 '----------PLACE A K-MINE------------------------------
610 XK = INT(9 * RND(0) + 1)
620 YK = INT(9 * RND(0) + 1)
630 IF XK * YK >= 64 THEN 610
640 IF A$(XK,YK)="K" OR A$(XK,YK)="L" A$(XK,YK)="L":GOTO 650
645   A$(XK,YK)="K"
650 PRINT"K-MINE RELEASED AT";XK;",";YK
700 '
701 '----------CHECK FOR COLLISION WITH K OR L MINE--------
710 IF A$(XE,YE) = "-" THEN 790
720 PRINT"######:::::  !!!  :::::######"
730 PRINT"YOU HAVE HIT A MINE.  YOU'RE BLOWN BACK TO START."
740 PRINT"ENERGY LOSS IS 30%": EE = .7 * EE
750 A$(XE,YE) = "-"
760 XE = 1: YE = 1
790 GOSUB 5020      'CHECK FOR L-MINE RAY DAMAGE
800 '----------CHECK FOR END OF GAME----------
810 IF EE < 20 OR TF > 99 THEN 3000
820 PRINT"PRESS ENTER FOR NEXT TIME FRAME  --READY";: INPUT A$
900 GOTO 210
990 '
998 '                  END OF THE MAIN LOOP
999 '==================================================
1000 '-------------------------------------------------
1001 '                STAR GATE BARRIER COLLISION
1002 '-------------------------------------------------
1016 PRINT: PRINT"YOU HIT THE STAR GATE BARRIER"
1020 PRINT "POW!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
1025 PRINT "IT'S BACK TO START FOR YOU"
1030 XE=1: YE=1
1090 GOTO 820
2000 '-------------------------------------------------
2001 '                 STAR DIMENSION CHECK
2002 '-------------------------------------------------
2005 IF A$(9,9)="G" PRINT "YOU TRIED TO ENTER A CLOSED GATE
      ..... YOU ARE SENTENCED TO START OVER":XE=1:YE=1:GOTO 820
2007 '
2008 ' ----------------VICTORY !!! ----------------
2009 PRINT "****************************************"
2010 PRINT "****************************************"
```

```
2020 PRINT "*** YOU HAVE ACHIEVED STAR DIMENSION ***"
2025 PRINT "***     YOUR RATING IS";10*EE/(TF+1);TAB(37);"***"
2026 PRINT "*****************************************"
2028 PRINT "*****************************************"
2030 FOR I=1 TO 1000:NEXT I
2050 FOR I= 1 TO 500
2060 PRINT @ RND(1010)-1, "* .  +      .";
2070 NEXT I
2090 GOTO 10000
3000 '-----------------------------------------------------
3001 '               E OUT OF ENERGY
3002 '-----------------------------------------------------
3010 PRINT "YOU ARE FINISHED --- THIS IS THE END"
3020 PRINT "ENERGY ="; EE; "TIME FRAME ="; TF
3090 STOP
5000 '---------------------------------------------------
5001 '     SUBROUTINE TO CHECK FOR L-MINE RAY HITS
5002 '---------------------------------------------------
5020 FOR XSCAN=1 TO 9
5025   IF A$(XSCAN,YE)<>"L" THEN 5040
5030      IF ABS(XSCAN-XE)=2 THEN EE=EE*.9:PRINT"X2-RAY HIT*** EE=";EE
5035      IF ABS(XSCAN-XE)=1 THEN EE=EE*.8:PRINT"X1-RAY HIT*** EE=";EE
5040 NEXT XSCAN
5050 FOR YSCAN=1 TO 9
5055    IF A$(XE,YSCAN)<>"L" THEN 5070
5060      IF ABS(YSCAN-YE)=2 THEN EE=.9*EE:PRINT"Y2-RAY HIT*** EE=";EE
5065      IF ABS(YSCAN-YE)=1 THEN EE=.8*EE:PRINT"Y1-RAY HIT*** EE=";EE
5070 NEXT YSCAN
5090 RETURN
10000 END
```

## Solutions to Projects in Section 2.6

### Project *ARROW2*

This program graphically displays an arrow shot by the player at a randomly placed target.

```
5  '*******************************************************
6  '*          ARROW2  (TRAJECTORY GRAPHICS GAME)          *
7  '*        THIS PROGRAM WAS WRITTEN FOR THE TRS-80       *
8  '*          USING THE SET(X,Y) GRAPHICS COMMAND.        *
9  '*******************************************************
10 CLS: RANDOM
15 G=32.2
20 PRINT"HOW MANY ARROWS DO YOU WANT";
30 INPUT N
32 D=INT(100 * RND(0) + 50)
33 S1=98/D        'X SCALE FACTOR
```

```
34 S2=S1*.5       'Y SCALE FACTOR
35 '=================================================
36 '              START OF LOOP
37 '
40 FOR L=1 TO N
50 PRINT"YOUR TARGET IS";D;"YARDS AWAY."
60 PRINT"WHAT ANGLE,VELOCITY WILL YOU USE (E.G. 35,50): ";
70 INPUT A,V
80 IF A>89 OR A<15  PRINT"ANGLE MUST BE 15 TO 89":GOTO 60
90 IF V>100 OR V<50 PRINT"VELOCITY MUST BE 50 TO 100":GOTO 60
95 CLS
100 A = A * 3.14159265/180
110 R = V * V * SIN(2*A)/G
120 H = (V * SIN(A))[2/(2*G)
200 '
201 '----------DISPLAY AXES----------
210 FOR K=1 TO 15: PRINT "!": NEXT K
220 PRINT "%";
230 FOR K=3 TO 50: PRINT "-";: NEXT K
240 PRINT @1009, "T--(" ;RIGHT$(STR$(D),LEN(STR$(D))-1);" YDS)--";
299 '
300 '-----CALCULATE AND PLOT TRAJECTORY-----
310 T1=2*V*SIN(A)/G
320 K1=S1*V*COS(A)
330 K2=S2*V*SIN(A)
340 K3=G*S2/2
350 FOR T=0 TO T1 STEP T1/50
360   X=INT(K1*T+2.5)
370   Y=INT((K2-K3*T)*T+2)
380   IF Y>47 OR Y<0 OR X>127 OR X<0 THEN 395
390   SET(X,47-Y)
395 NEXT T
399 '
400 '------PRINT RESULTS-----
410 PRINT @ 0, "KERPLOP !!!"
420 IF ABS(R-D) < 3 PRINT "HIT !!!!!!!": GOTO 515
430 PRINT "YOU MISSED BY";R-D;"YDS."
440 PRINT
450 PRINT"***  YOU HAVE";N-L;"ARROWS LEFT ***"
460 NEXT L
470 '
480 '             END OF MAIN LOOP
490 '=================================================
497 PRINT"YOU'RE OUT OF ARROWS--SCORE = 0"
498 PRINT"IF YOU WANT TO TRY AGAIN TYPE RUN."
499 STOP
500 '------CALCULATE SCORES FOR HITS------
515 PRINT"YOU USED";L;"OUT OF YOUR";N;"ARROWS."
517 PRINT"YOUR FINAL SCORE IS";
520 PRINT 100 * (12 - L)/(L * N)
530 PRINT"POSSIBLE SCORES FOR SOME CHOICES OF ARROWS ARE:"
535 PRINT"  1-ARROW 2-ARROWS 3-ARROWS 4-ARROWS 5-ARROWS 6-ARROWS"
540 FOR K = 1 TO 6
550   FOR N = K TO 6
560     PRINT TAB(9*(N-1)+3);100*(12-K)/(K*N);
570   NEXT N
580 PRINT
590 NEXT K
900 END
```

## 5.3 SOLUTIONS TO THE PROJECTS IN CHAPTER 3

### Solutions to the Projects in Section 3.2

### Project *FINANCE*

This program is a menu program which includes LOAN1, SAVE, AMORT, and LOAN2.

```
10 '*******************************************************
11 '*      FINANCE  (4 BORROWING/SAVING INTEREST REPORTS)     *
12 '*******************************************************
100 '-----------------------------------------------------
101 '                          PRESENT MENU
102 '-----------------------------------------------------
105 CLS: N=0
106 PRINT"             ***  BORROWING & SAVING CALCULATIONS  ***"
108 PRINT
110 PRINT"CHOOSE ONE OF THE FOLLOWING:"
120 PRINT"  1. INSTALLMENT PAYMENTS, WITH ADD-ON INTEREST."
130 PRINT"  2. INSTALLMENT PAYMENTS, WITH INT. ON UNPAID BAL."
140 PRINT"  3. SAVINGS ACCT. INTEREST, COMPOUNDED DAILY."
150 PRINT"  4. INSTALLMENT PAYMENTS, WITH AMORTIZATION."
160 PRINT"  5. END THE PROGRAM.
170 PRINT"TYPE 1, 2, 3, 4, OR 5  --";:INPUT N
175 IF N<1 OR N>5 THEN 170
180 ON N GOTO 205, 305, 405, 505, 9999
200 '************************************************
201 '*                        LOAN1                  *
202 '************************************************
205 CLS: A$=" "
207 PRINT"* INSTALLMENT PAYMENTS WITH ADD-ON INTEREST *"
210 '----------GET INPUT--------------------------------
212 PRINT"AMOUNT BORROWED (PRINCIPAL)";: INPUT P
215 PRINT"NUMBER OF MONTHS TO PAY";: INPUT M
216 IF M <= 0 OR INT(M) <> M THEN 215
220 PRINT"INTEREST RATE PER YEAR (6.5% = 6.5)";: INPUT R
222 '----------CALCULATE--------------------------------
225 R = R * .01
226 TI = P * R * M / 12                 'TOTAL INTEREST
227 '----------PRINT REPORT-----------------------------
229 PRINT
230 PRINT"MONTHS", "PRINCIPAL", "INTEREST", "MONTHLY"
240 PRINT"TO PAY",,            "PER MO.",  "PAYMENT"
250 FOR K = 1 TO 60: PRINT"-";: NEXT K: PRINT
255 F1$ ="  ##         $#,###,###.##     $#,###.##       $##,###.##"
260 PRINT USING F1$; M, P, TI/M, (P+TI)/M
265 PRINT
270 PRINT,,"TOTAL","TOTAL AMOUNT"
271 PRINT,,"INTEREST","PAID BACK"
275 FOR K = 1 TO 60: PRINT"-";: NEXT K: PRINT
279 F2$ ="$###,###.##     $#,###,###.##"
280 PRINT TAB(31);: PRINT USING F2$; TI, P+TI
285 '----------AGAIN?----------------------------------
290 PRINT"AGAIN (Y = YES)";: INPUT A$: IF A$ ="Y" THEN 205
```

```
295 GOTO 105
300 '*****************************************************
301 '*                        LOAN2                      *
302 '*****************************************************
305 CLS: A$=" "
307 PRINT"* INSTALLMENT PAYMENTS WITH INT. ON UNPAID BAL. *"
310 '----------GET INPUT--------------------------------
312 PRINT"AMOUNT BORROWED (PRINCIPAL)";: INPUT P
315 PRINT"NUMBER OF MONTHS TO PAY";: INPUT M
316 IF M <= 0 OR INT(M) <> M THEN 315
317 PRINT"INTEREST RATE PER YEAR (6.5% = 6.5)";: INPUT R
319 PRINT"--CONTINUE";: INPUT D$
320 '----------CALCULATE FIXED QUANTITIES----------------
322 R = R * .01
324 P1 = P/M
325 '----------CALCULATE & PRINT TABLE-------------------
326 TI = 0: TP = 0: SP = 0
327 PRINT"MONTH   PRINCIPAL    INTEREST   +   PRINCIPAL  =  MONTHLY"
329 PRINT"NUMBER  OWED         PAYMENT        PAYMENT       PAYMENT"
330 FOR K = 1 TO 60: PRINT"-";: NEXT K: PRINT
332 F3$ =" ##    $###,###.##    $#,###.##      $#,###.##    $#,###.##"
340 FOR J = 1 TO M
342   IF J/12 = INT(J/12) PRINT"--CONTINUE";: INPUT D$
345   I1 = P * R / 12
350   PRINT USING F3$; J, P, I1, P1, P1+I1
355   TI = TI + I1
360   TP = TP + P1 + I1
363   SP = SP + P1
365   P = P - P1
370 NEXT J
372 PRINT"--CONTINUE";: INPUT D$
374 '----------SUMMARY----------------------------------
375 PRINT TAB(21);"TOTAL           TOTAL            TOTAL"
377 PRINT TAB(21);"INTEREST        PRINCIPAL        PAYMENTS"
380 FOR K = 1 TO 60: PRINT"-";: NEXT K: PRINT
382 F4$ ="$##,###.##     $###,###.##     $###,###.##"
383 PRINT TAB(19);:PRINT USING F4$; TI, SP, TP
384 '----------AGAIN?-----------------------------------
385 PRINT"AGAIN (Y = YES)";: INPUT A$: IF A$="Y" THEN 305
390 GOTO 105
400 '*****************************************************
401 '*                        SAVE                       *
402 '*****************************************************
405 CLS: A$=" "
410 PRINT"* SAVINGS ACCT. INTEREST, COMPOUNDED DAILY *"
412 '----------GET INPUT--------------------------------
415 PRINT"STARTING BALANCE";: INPUT P
417 PRINT"NUMBER OF DAYS COMPOUNDED";: INPUT D
418 IF D <= 0 OR INT(D) <> D THEN 417
420 PRINT"INTEREST RATE PER YEAR (6.5% = 6.5)";: INPUT R
422 PRINT"--CONTINUE";: INPUT D$
425 '----------CALCULATE & PRINT TABLE-------------------
430 R = R * .01 / 360          'BANKER'S YEAR = 360 DAYS
435 PRINT"DAY", "BALANCE", "INTEREST
437 FOR K = 1 TO 60: PRINT"-";: NEXT K: PRINT
438 PRINT 0, P, 0
439 TI = 0
440 FOR J = 1 TO D
443   IF J/12 = INT(J/12) PRINT "--CONTINUE";:INPUT D$
444   I1 = P * R
445   TI = TI + I1
450   P = P + I1
```

```
455   PRINT J, P, I1
460 NEXT J
465 PRINT"--CONTINUE";: INPUT D$
470 '----------SUMMARY-----------------------------------
472 PRINT,"BALANCE","TOTAL INTEREST"
474 FOR K = 1 TO 60: PRINT"-";: NEXT K: PRINT
477 X$="          $##,###.##       ###.######"
478 PRINT USING X$; P, TI
480 '----------AGAIN?------------------------------------
485 PRINT"AGAIN (Y = YES)";: INPUT A$: IF A$="Y" THEN 405
495 GOTO 105
500 '****************************************************
501 '*                      AMORT                       *
502 '****************************************************
505 CLS: A$=" "
507 PRINT"* AMORTIZATION TABLE *"
510 '----------GET INPUT---------------------------------
515 PRINT"AMOUNT BORROWED (PRINCIPAL)";: INPUT P
517 PRINT"NUMBER OF MONTHS TO PAY";: INPUT M
518 IF M <= 0 OR INT(M) <> M THEN 517
519 PRINT"INTEREST RATE PER YEAR (6.5% = 6.5)";: INPUT R
520 PRINT"--CONTINUE";: INPUT D$
522 '----------CALCULATE FIXED QUANTITIES----------------
523 R = R * .01 / 12
524 E = (P * R * (1 + R)[M) / ((1 + R)[M-1)
525 '----------CALCULATE & PRINT TABLE-------------------
526 TI = 0: TP = 0: SP = 0
527 PRINT"MONTH   PRINCIPAL     INTEREST   +   PRINCIPAL =   MONTHLY"
529 PRINT"NUMBER  OWED          PAYMENT        PAYMENT       PAYMENT"
530 FOR K = 1 TO 60: PRINT"-";: NEXT K: PRINT
532 F3$=" ##    $###,###.##    $#,###.##      $#,###.##   $#,###.##"
545 FOR J = 1 TO M
550   IF J/12 = INT(J/12) PRINT"--CONTINUE";: INPUT D$
555   I1 = P * R
560   P1 = E - I1
565   IF J = M THEN P1 = P: I1 = E - P1
570   PRINT USING F3$; J, P, I1, P1, E
575   TI = TI + I1
580   TP = TP + P1 + I1
584   SP = SP + P1
587   P = P - P1
590 NEXT J
600 PRINT"--CONTINUE";: INPUT D$
602 '----------SUMMARY-----------------------------------
605 PRINT TAB(21);"TOTAL           TOTAL            TOTAL"
610 PRINT TAB(21);"INTEREST        PRINCIPAL        PAYMENTS"
615 FOR K = 1 TO 60: PRINT"-";: NEXT K: PRINT
620 F4$="$##,###.##      $###,###.##      $###,###.##"
630 PRINT TAB(19);: PRINT USING!F4$; TI, SP, TP
635 '----------AGAIN?--------,---------------------------
640 PRINT"AGAIN (Y = YES)";: INPUT A$: IF A$="Y" THEN 505
650 GOTO!105
9999 END
```

### Project *FINAN2*

This is an improvement on *FINANCE* that uses subroutines for input and printing.

```
10 '*********************************************************
11 '*     FINAN2  (USES SUBROUTINES)                        *
12 '*********************************************************
100 '-------------------------------------------------------
101 '                       PRESENT MENU
102 '-------------------------------------------------------
105 CLS: N=0
106 PRINT"          ***  BORROWING & SAVING CALCULATIONS  ***"
108 PRINT
110 PRINT"CHOOSE ONE OF THE FOLLOWING:"
120 PRINT"  1. INSTALLMENT PAYMENTS, WITH ADD-ON INTEREST."
130 PRINT"  2. INSTALLMENT PAYMENTS, WITH INT. ON UNPAID BAL."
140 PRINT"  3. SAVINGS ACCT. INTEREST, COMPOUNDED DAILY."
150 PRINT"  4. INSTALLMENT PAYMENTS, WITH AMORTIZATION."
160 PRINT"  5. END THE PROGRAM.
170 PRINT"TYPE 1, 2, 3, 4, OR 5  --";:INPUT N
175 IF N<1 OR N>5 THEN 170
180 ON N GOTO 205, 305, 405, 505, 9999
200 '**************************************************
201 '*                      LOAN1                     *
202 '**************************************************
205 CLS: A$=" "
207 PRINT"* INSTALLMENT PAYMENTS WITH ADD-ON INTEREST *"
210 '----------GET INPUT--------------------------------
220 GOSUB 1010
222 '----------CALCULATE--------------------------------
225 R = R * .01
226 TI = P * R * M / 12               'TOTAL INTEREST
227 '----------PRINT REPORT-----------------------------
229 PRINT
230 PRINT"MONTHS", "PRINCIPAL", "INTEREST", "MONTHLY"
240 PRINT"TC PAY",,                  "PER MO.",  "PAYMENT"
250 GOSUB 1130
255 F1$ ="  ##         $#,###,###.##     $#,###.##       $##,###.##"
260 PRINT USING F1$; M, P, TI/M, (P+TI)/M
265 PRINT
270 PRINT,,"TOTAL","TOTAL AMOUNT"
271 PRINT,,"INTEREST","PAID BACK"
275 GOSUB 1130
279 F2$ ="$###,###.##     $#,###,###.##"
280 PRINT TAB(31);: PRINT USING F2$; TI, P+TI
285 '----------AGAIN?-----------------------------------
290 PRINT"AGAIN (Y = YES)";: INPUT A$: IF A$ ="Y" THEN 205
295 GOTO 105
300 '**************************************************
301 '*                      LOAN2                     *
302 '**************************************************
305 CLS: A$=" "
307 PRINT"* INSTALLMENT PAYMENTS WITH INT. ON UNPAID BAL. *"
310 '----------GET INPUT--------------------------------
315 GOSUB 1010
319 GOSUB 2010
320 '----------CALCULATE FIXED QUANTITIES---------------
```

```
322 R = R * .01
324 P1 = P/M
325 '----------CALCULATE & PRINT TABLE------------------
326 TI = 0: TP = 0: SP = 0
330 GOSUB 1070
340 FOR J = 1 TO M
342   IF J/12 = INT(J/12) THEN GOSUB 2010
345   I1 = P * R / 12
350   PRINT USING F3$; J, P, I1, P1, P1+I1
355   TI = TI + I1
360   TP = TP + P1 + I1
363   SP = SP + P1
365   P = P - P1
370 NEXT J
372 GOSUB 2010
374 '----------SUMMARY----------------------------------
380 GOSUB 1150
383 PRINT TAB(19);:PRINT USING F4$; TI, SP, TP
384 '----------AGAIN?-----------------------------------
385 PRINT"AGAIN (Y = YES)";: INPUT A$: IF A$="Y" THEN 305
390 GOTO 105
400 '**************************************************
401 '*                    SAVE                         *
402 '**************************************************
405 CLS: A$=" "
410 PRINT"* SAVINGS ACCT. INTEREST, COMPOUNDED DAILY *"
412 '----------GET INPUT--------------------------------
415 PRINT"STARTING BALANCE";: INPUT P
417 PRINT"NUMBER OF DAYS COMPOUNDED";: INPUT D
418 IF D <= 0 OR INT(D) <> D THEN 417
420 PRINT"INTEREST RATE PER YEAR (6.5% = 6.5)";: INPUT R
422 GOSUB 2010
425 '----------CALCULATE & PRINT TABLE------------------
430 R = R * .01 / 360          'BANKER'S YEAR = 360 DAYS
435 PRINT"DAY", "BALANCE", "INTEREST
437 GOSUB 1130
438 PRINT 0, P, 0
439 TI = 0
440 FOR J = 1 TO D
443   IF J/12 = INT(J/12) THEN GOSUB 2010
444   I1 = P * R
445   TI = TI + I1
450   P = P + I1
455   PRINT J, P, I1
460 NEXT J
465 GOSUB 2010
470 '----------SUMMARY----------------------------------
472 PRINT,"BALANCE","TOTAL INTEREST"
474 GOSUB 1130
477 X$="              $##,###.##      ###.######"
478 PRINT USING X$; P, TI
480 '----------AGAIN?-----------------------------------
485 PRINT"AGAIN (Y = YES)";: INPUT A$: IF A$="Y" THEN 405
486 GOTO 105
500 '**************************************************
501 '*                 AMORT                           *
502 '**************************************************
505 CLS: A$=" "
507 PRINT"* AMORTIZATION TABLE *"
510 '----------GET INPUT--------------------------------
515 GOSUB 1010
520 GOSUB 2010
```

```
522 '----------CALCULATE FIXED QUANTITIES----------------
523 R = R * .01 / 12
524 E = (P * R * (1 + R)[M) / ((1 + R)[M-1)
525 '----------CALCULATE & PRINT TABLE-------------------
526 TI = 0: TP = 0: SP = 0
530 GOSUB 1070
545 FOR J = 1 TO M
550   IF J/12 = INT(J/12) THEN GOSUB 2010
555   I1 = P * R
560   P1 = E - I1
565   IF J = M THEN P1 = P: I1 = E - P1
570   PRINT USING F3$; J, P, I1, P1, E
575   TI = TI + I1
580   TP = TP + P1 + I1
584   SP = SP + P1
587   P = P - P1
590 NEXT J
600 GOSUB 2010
602 '----------SUMMARY-----------------------------------
610 GOSUB 1150
630 PRINT TAB(19);: PRINT USING F4$; TI, SP, TP
635 '----------AGAIN?------------------------------------
640 PRINT"AGAIN (Y = YES)";: INPUT A$: IF A$="Y" THEN 505
690 GOTO 105
999 GOTO 9999
1000 '-------------------------------------------------
1001 '    SUBROUTINE, GETS INPUT
1002 '-------------------------------------------------
1010 PRINT"AMOUNT BORROWED (PRINCIPAL)";: INPUT P
1020 PRINT"NUMBER OF MONTHS TO PAY";: INPUT M
1030 IF M <= 0 OR INT(M) <> M THEN 1020
1040 PRINT"INTEREST RATE PER YEAR (6.5% = 6.5)";: INPUT R
1050 RETURN
1060 '-------------------------------------------------
1061 '    SUBROUTINE, HEADING 1, LOAN2, AMORT
1062 '-------------------------------------------------
1070 PRINT"MONTH   PRINCIPAL    INTEREST   +   PRINCIPAL =   MONTHLY"
1080 PRINT"NUMBER  OWED         PAYMENT        PAYMENT       PAYMENT"
1090 GOSUB 1130            'DRAWS LINE
1100 F3$=" ##   $###,###.##    $###.##       $#,###.##    $#,###.##"
1110 RETURN
1120 '-------------------------------------------------
1130 FOR K=1 TO 60: PRINT"-";: NEXT K: PRINT: RETURN
1135 '-------------------------------------------------
1140 '-------------------------------------------------
1141 '    SUBROUTINE, HEADING 2, LOAN2, AMORT
1142 '-------------------------------------------------
1150 PRINT TAB(21);"TOTAL          TOTAL           TOTAL"
1160 PRINT TAB(21);"INTEREST       PRINCIPAL       PAYMENTS"
1170 GOSUB 1130            'DRAWS LINE
1180 F4$="$#,###.##     $###,###.##     $###,###.##"
1190 RETURN
2000 '-------------------------------------------------
2001 '    SUBROUTINE, SCREEN PAUSE
2002 '-------------------------------------------------
2010 PRINT"--CONTINUE";: INPUT D$: RETURN
9999 END
```

## Solution to the Project in Section 3.5

### Project *HARDSALE*

This program is an extension of SALESLIP which prints the final saleslip on a computer-controlled printer.

```
10 '****************************************************
20 '*   HARDSALE  (COLLECTS TRANSACTIONS & LPRINTS BILL)   *
30 '****************************************************
40 CLS: CLEAR 500
100 '--------------------------------------------------
101 '                    INITIALIZATION
102 '--------------------------------------------------
110 DATA"JAN.","FEB.","MAR.","APRIL","MAY","JUNE","JULY","AUG."
120 DATA "SEPT.","OCT.","NOV.","DEC."
125 PRINT "DATE (mm,DD,YY)";: INPUT M,D,Y
130 FOR K=1 TO M: READ M$: NEXT K
140 PRINT "TYPE LAST SALES SLIP # USED";: INPUT S: S=S+1
160 DIM Q(25),C$(25),D$(25),P(25)
200 '--------------------------------------------------
201 '                    SALES DATA ENTRY
202 '--------------------------------------------------
210 CLS: PRINT "SALES SLIP #";S;TAB(27);"DATE:";M$;" ";D;",";1900 + Y
220 PRINT "..............................................."
225 L=0
230 PRINT "CUSTOMER NAME";:INPUT N$
240 PRINT "STREET ADDRESS";:INPUT S$
250 PRINT "CITY & ZIP";:INPUT Z$
255 PRINT "TYPE     QTY   CAT#     DESCRIPTION   UNIT PRICE (0 = EXIT)
260 L=L+1
280 N=L : GOSUB 2000
290 GOTO 260
300 '--------------------------------------------------
301 '                  CORRECTION ROUTINE
302 '--------------------------------------------------
310 PRINT "TYPE F IF FINISHED, C TO CHANGE, R TO RESUME, V TO VOID";
315 INPUT A$
320 IF A$="F" THEN 400
330 IF A$="C" THEN 360
340 IF A$="V" THEN 210
345 IF A$="R" THEN 280
350 GOTO 310
360 PRINT "WHICH ITEM # TO BE CHANGED";:INPUT C
365 IF C > L THEN 360
370 N=C: GOSUB 2000
375 IF C=L THEN L=L+1
380 GOTO 310
400 '--------------------------------------------------
401 '                  VIDEO SALES INVOICE
402 '--------------------------------------------------
410 CLS: PRINT TAB(15); "<*> MADE BY HAND SHOPS <*>"
415 PRINT: PRINT "INVOICE #";S, TAB(39); "DATE:  ";M$;" ";D;",";1900+Y
420 PRINT "SOLD TO: "; N$
425 PRINT TAB(9);S$
430 PRINT TAB(9);Z$
```

```
440 PRINT
445 PRINT "          QTY    CAT#      DESCRIPTION  PRICE     TOTAL"
450 PRINT "......................................................."
455 T=0
460 FOR K=1 TO L-1
465 E=P(K)*Q(K)
468 F$="ITEM ##:  ###    %       %  %             %  ###.##    #####.##"
470 PRINT USING F$;K,Q(K),C$(K),D$(K),P(K),E
475 T=T+E
480 NEXT K
485 PRINT
486 X=.06*T     'CHANGE TAX FORMULA FOR YOUR STATE
487 PRINT TAB(47);:PRINT USING "  SUM = ####.##";T
488 PRINT TAB(47);:PRINT USING "  TAX = ####.##";X
489 PRINT TAB(47);:PRINT USING "TOTAL = ####.##";T+X
490 PRINT CHR$(27);"TYPE P TO PRINT, V TO VOID";:A$="":INPUT A$
494 IF A$="V" THEN 210
495 IF A$="P" THEN 497
496 PRINT "??? PLEASE RETYPE";:INPUT A$: GOTO 494
497 PRINT "IF CASH, TYPE PAYMENT; IF CREDIT TYPE 0";:INPUT A
498 IF A>0 PRINT USING "CHANGE IS ##.##";A-T-X: GOTO 4010
500 '----------------------------------------------------
501 '          CHARGE ACCOUNT SECTION
502 '----------------------------------------------------
510 PRINT "CHARGE ACCT. SECTION NOT WRITTEN YET *** RECORD BY HAND"
590 PRINT "PRESS 'ENTER' TO CONTINUE -- READY";:INPUT A$
595 GOTO 4010
2000 '---------------------------------------------------
2001 '        CURSOR-CONTROLLED INPUT SUBROUTINE
2002 '---------------------------------------------------
2010 PRINT "ITEM";N;":";:INPUT Q(N):IF Q(N)=0 THEN 300
2020 PRINT TAB(14);CHR$(27);:INPUT C$(N):IF C$(N)="0" THEN 300
2030 PRINT TAB(23);CHR$(27);:INPUT D$(N):IF D$(N)="0" THEN 300
2040 PRINT TAB(37);CHR$(27);:INPUT P(N):IF P(N)=0 THEN 300
2050 RETURN
4000 '---------------------------------------------------
4001 '        LINE PRINTER SECTION
4002 '----------------------------------------------------
4010 PRINT "TURN PRINTER ON -- READY";:INPUT A$
4020 LPRINT TAB(18);"<*><*><*><*><*><*><*><*><*><*>"
4021 LPRINT TAB(18);"<*>                        <*>"
4022 LPRINT TAB(18);"<*>   MADE BY HAND SHOPS   <*>"
4024 LPRINT TAB(18);"<*>  PITTSBURGH, PA 15213  <*>"
4025 LPRINT TAB(18);"<*>                        <*>"
4026 LPRINT TAB(18);"<*><*><*><*><*><*><*><*><*><*>"
4028 LPRINT" "
4030 LPRINT" ":LPRINT"INVOICE #";S,TAB(39);"DATE:";M$;" ";D;",";1900+Y
4032 LPRINT"SOLD TO: ";N$
4034 LPRINT TAB(9);S$
4036 LPRINT TAB(9);Z$
4038 LPRINT " "
4040 LPRINT"          QTY    CAT#      DESCRIPTION  PRICE     TOTAL"
4042 LPRINT"......................................................."
4046 FOR K=1 TO L-1
4048 E=P(K)*Q(K)
4050 F$=F$
4052 LPRINT USING F$;K,Q(K),C$(K),D$(K),P(K),E
4056 NEXT K
4058 LPRINT" "
4062 LPRINT TAB(47);:LPRINT USING"  SUM = ####.##";T
4064 LPRINT TAB(47);:LPRINT USING"  TAX = ####.##";X
4066 LPRINT TAB(47);:LPRINT USING"TOTAL = ####.##";T+X
```

```
4068 LPRINT" ":IF A>0 THEN 4080
4070 LPRINT"***CREDIT PURCHASE--ABOVE TOTAL IS DUE IN 25 DAYS***":GOTO 4085
4080 LPRINT USING "AMOUNT REMITTED = ####.##";A
4082 LPRINT USING "    YOUR CHANGE = ####.##";A-T-X
4085 FOR I=1 TO 10:LPRINT" ":NEXT I
4090 PRINT "RESUME(R), PRINT AGAIN(P), OR QUIT(Q)";:A$="":INPUT A$
4092 IF A$="R" THEN S=S+1: GOTO210
4094 IF A$="P" THEN 4010
4096 IF A$="Q" THEN 9999
4099 GOTO 4090
9999 END
```

## Solution to the Project in Section 3.6

### Project *MONEY2*

This program is an extension of MONEY. It adds a V(oid) command in lines 1210 and 1225, and has the void subroutine in lines 1900 to 1990.

```
10  ' **************************************************
15  ' *                    MONEY2                       *
20  ' *   COMPUTERIZED FINANCIAL/TAX REPORTING SYSTEM   *
40  ' **************************************************
100 ' --------------------------------------------------
101 '       INITIALIZE VARIABLES; DECLARE DATA TYPES
102 ' --------------------------------------------------
105 CLS
110 PRINT "          <*>   FINANCIAL TAX/REPORTING SYSTEM    <*>
112 DEFINT C-P
114 CLEAR 2000
115 PRINT STRING$(60,"-")
120 INPUT "FOR CALENDAR YEAR 19";IY
130 IF IY<70 OR IY>99  PRINT "INVALID YEAR" : GOTO 120
140 INPUT "FOR PERIOD #"; P
150 IF P<1 OR P>366 PRINT "INVALID PERIOD #" : GOTO 140
160 INPUT "STARTING BALANCE THIS PERIOD (E.G. 589.65)"; BS
165 BC=BS                'INITIALIZE CURRENT CK BALANCE
170 INPUT "CK. ACCT. OR BANK NAME"; CA$
175 INPUT "STARTING CK. # THIS PERIOD"; CS
176 CK=CS                'CK IS CURRENT CK ABOUT TO BE PROCESSED
177 CH=CS-1              'CH IS HIGHEST CK # ALREADY PROCESSED
179 PRINT:PRINT TAB(25);"STAND BY"
180 DS=1 : PS=1 : HS=1 'STARTING DEP, CASH PAY, & CASH RCVD #S
181 DK=DS : DH=DS-1      'CURRENT & HIGHEST DEP #
183 PK=PS : PH=PS-1      'CURRENT AND HIGHEST CASH PAY #
184 HK=HS : HH=HS-1      'CURRENT AND HIGHEST CASH RCVD #
185 F$="### ##/## $####.##  %            % %            % %   % ### ###
186 D$="$#####.##"
187 F1$="####: ##/##/##  $#####.##   %              %"
```

```
188 F2$=" %              % %   % ###  ###"
189 LP$=F1$+F2$
190 DIM GD(200),GA(200),GV$(200),GD$(200),GF$(200),GP(200),GC(200)
198 FOR J=1 TO 200: GD(J)=-1: NEXT J
200 ' -----------------------------------------------
201 '                  MAIN MENU SELECTION
202 ' -----------------------------------------------
210 CLS
220 PRINT "                 <<<    MAIN MENU SELECTION    >>>
225 PRINT STRING$(60,"-")
230 INPUT "TRANSACTION (1), REPORT (2), OR EXIT (3)";  MS
240 IF MS<1 OR MS>3 THEN 230
250 ON MS GOTO 1010, 5010, 9910
999 '
1000 ' ================================================
1001 '                TRANSACTION PROGRAMS
1002 '
1003 ' ------------------------------------------------
1004 '         TRANSACTION  SUB-MENU SELECTION
1005 ' ------------------------------------------------
1010 CLS
1020 PRINT "          <   TRANSACTION MENU SELECTION   >"
1030 PRINT STRING$(60,"-")
1040 PRINT"     1 = CK PAYMENTS BY YOU"
1042 PRINT"     2 = CASH PAYMENTS BY YOU"
1044 PRINT"     3 = CK RECEIPTS DEPOSITED BY YOU"
1046 PRINT"     4 = CASH RECEIPTS"
1048 PRINT"     5 = EXIT FROM TRANSACTION SECTION"
1050 PRINT
1055 INPUT"1, 2, 3, 4, OR 5";  MS
1060 IF MS<1 OR MS>5 THEN 1055
1065 IF MS=5 THEN 210
1070 ON MS GOSUB 1410, 1510, 1610, 1710
1099 '
1100 ' ------------------------------------------------
1101 '              MAIN TRANSACTION PROGRAM
1102 ' ------------------------------------------------
1110 CLS
1115 PRINT"*** ";LEFT$(TR$,3);"TRANSACTIONS FOR PERIOD";P;"OF";1900+IY
1120 PRINT"***STARTING CK BALANCE THIS PERIOD=";TAB(36); USING D$;BS
1125 PRINT"***CURRENT CK BALANCE THIS PERIOD =";TAB(36); USING D$;BC
1130 PRINT "TO MAKE A CHANGE  OR EXIT TYPE 0,0 FOR MO,DAY"
1132 PRINT "TO RESTART A LINE TYPE -1 IN ANY COLUMN"
1134 PRINT STRING$(60,"-")
1135 PRINT TR$;"  MO,DAY: AMT:     VENDOR:      ";
1140 PRINT "FOR:        TAX CODE: %: ACT#"
1150 GK=GH+1        'GK IS TR. # ABOUT TO BE PROCESSED
1155 I=(GK-GS)+C1      'INDEX TO TR. RECORDS IN G ARRAY
1157 IF I>MAX PRINT "OUT OF SPACE" : GOTO 1210
1160 GOSUB 3010     'COLLECT TR. DATA
1165 IF M=0 AND D=0 THEN 1210
1170   GH=GH+1
1175   GOTO 1150
1200 '
1201 '.............TRANSACTION EXIT ROUTINE............
1210 PRINT "EXIT(E), CHANGE(C), RESUME(R), LIST(L), OR VOID(V)";
1215 INPUT A$
1222 IF A$="R" THEN 1135
1225 IF A$="V" THEN : GOSUB 1910: : GOTO 1210
1230 IF A$="E" THEN 4010
1235 IF A$="L" THEN 1258
1237 IF A$="C" THEN 1245
```

```
1240 GOTO 1210
1241 '
1242 '.............CHANGE ROUTINE.............
1245 PRINT "CHANGE DATA FOR WHICH #";:INPUT GK
1247 IF GK>GH PRINT TR$" # TOO HIGH": GOTO 1210
1248  IF GK<GS PRINT "TOO LOW" : GOTO 1210
1250   I=(GK-GS)+C1 : BC=BC-M1*GA(I)*.01
1253   PRINT TR$;"  MO,DAY: AMT:      VENDOR:      ";
1254   PRINT "FOR:        TAX CODE: %: ACT#"
1256   GOSUB 3010 : GOTO 1210
1257 '
1258 '.............LIST ROUTINE................
1259 PRINT "LIST OF ";LEFT$(TR$,3);" TRANSACTIONS NOW IN G ARRAY"
1260 FOR J=GS TO GH
1261   K=(J-GS)+C1
1263   M=INT(GD(K)/100) : D=GD(K)-M*100
1265   PRINT USING F$;J,M,D,GA(K)*.01,GV$(K),GD$(K),GF$(K),GP(K),GC(K)
1270 NEXT J
1290 GOTO 1210
1400 '-----------------------------------------------
1401 '    SUBROUTINES TO PASS TRANSACTION PARAMETERS
1402 '-----------------------------------------------
1410 TR$="CK #" : GS=CS : GH=CH : GK=CK : MAX=99 : C1=1 : M1=-1
1490 RETURN
1500 '
1510 TR$="PAY#" : GS=PS : GH=PH : GK=PK : MAX=149 : C1=101 : M1=0
1590 RETURN
1600 '
1610 TR$="DEP#" : GS=DS : GH=DH : GK=DK : MAX=189 : C1=151 : M1=1
1690 RETURN
1700 '
1710 TR$="CSH#" : GS=HS : GH=HH : GK=HK : MAX=199 : C1=191 : M1=0
1790 RETURN
1900 ' -----------------------------------------------
1901 '       SUBROUTINE TO VOID TRANSACTIONS
1902 ' -----------------------------------------------
1910 PRINT "  * HIGHEST TRANSACTION # PROCESSED SO FAR IS";GH
1920 PRINT "  * VOID REMOVES DATA FROM # XX ON UP. WHAT VALUE"
1925 PRINT "  * OF XX DO YOU WISH TO VOID FROM (-1 MEANS NONE)";
1930 INPUT XX
1950 IF XX<0 THEN 1990
1955 IF XX<GS PRINT "TR # TOO LOW.  XX =";:GOTO 1930
1957 IF XX>GH PRINT "TR # TOO HIGH.   XX =";:GOTO 1930
1960 FOR J=XX TO GH
1961   K=(J-GS)+C1
1962   BC=BC-M1*GA(K)*.01
1964 NEXT J
1968 GH=XX-1
1970   PRINT "ALL TRANSACTIONS FROM #";XX;"ON UP ARE  VOID"
1980   PRINT "HIGHEST TRANSACTION # PROCESSED IS NOW";GH
1990 RETURN
3000 ' -----------------------------------------------
3001 '      SUBROUTINE TO COLLECT TRANSACTION DATA
3002 ' -----------------------------------------------
3010 PRINT GK;TAB(5);: INPUT M,D: IF M=0 AND D=0 THEN 3090
3012 IF M<0 OR D<0 THEN 3010
3015 GD(I)=100*M + D
3016 AM=0: GV$(I)="NO ENTRY": GD$(I)="NO ENTRY"
3017 GF$(I)="NONE": GP$=" ": GC(I)=0
3018 PRINT TAB(12);CHR$(27);:INPUT AM:IF AM<0 THEN 3010
3025 PRINT TAB(21);CHR$(27);:INPUT GV$(I):IF GV$(I)="-1"THEN 3010
3030 PRINT TAB(33);CHR$(27);:INPUT GD$(I):IF GD$(I)="-1"THEN 3010
```

```
3045 PRINT TAB(45);CHR$(27);:INPUT GF$(I):IF GF$(I)="-1"THEN 3011
3055 PRINT TAB(53);CHR$(27);:INPUT GP$:IF GP$="-1"THEN 3010
3057 BC=BC+AM*M1
3058 GA(I)=INT(100*AM)
3060 IF GP$="" OR GP$=" " THEN GP(I)=100 ELSE GP(I)=VAL(GP$)
3065 PRINT TAB(58);CHR$(27);:INPUT GC(I): IF GC(I)<0 THEN 3010
3090 RETURN
4000 '-----------------------------------------------,--------
4001 '    TRANSACTION "CLEAN-UP" PROGRAM; DISK SAVE
4002 '--------------------------------------------------------
4010 PRINT: PRINT"  < EXIT FROM TRANSACTION SECTION >"
4012 PRINT ">>> CURRENT BALANCE =";BC
4015 IF BC<0 PRINT "*** WARNING: CURRENT CHECKING BAL. IS NEGATIVE ***"
4020 PRINT"SAVE(S), RESUME(R), OR EXIT(E)";:INPUT A$
4030 IF A$="S" THEN 4110
4040 IF A$="R" THEN 1134
4050 IF A$="E" THEN 4510
4060 GOTO 4020
4100 '
4101 '.............DISK SAVE ROUTINE.............
4110 PRINT "NO DISK SAVE YET": GOTO 4020
4510 '
4530 CK=CH+1
4550 IF TR$="CK #":CS=GS:CH=GH:CK=GK: GOTO 1010
4555 IF TR$="DEP#":DS=GS:DH=GH:DK=GK: GOTO 1010
4560 IF TR$="PAY#":PS=GS:PH=GH:PK=GK: GOTO 1010
4565 IF TR$="CSH#":HS=GS:HH=GH:HK=GK: GOTO 1010
4570 PRINT "ERROR IN TR$ -- IT = ";TR$: STOP
5000 ' ==================================================
5001 '              REPORT GENERATING PROGRAMS
5002 '
5010 PRINT "REPORT SECTION NOT WRITTEN YET"
5055 FOR J=1 TO 500: NEXT J     'DELAY LOOP
5090 GOTO 210
9000 ' ================================================
9900 ' -----------------------------------------------
9901 '              FINAL EXIT MESSAGES
9902 ' -----------------------------------------------
9910 CLS: PRINT "*** END OF PERIOD";P;"TRANSACTIONS FOR";1900+IY;"***"
9915 PRINT STRING$(50,"."): PRINT
9920 PRINT USING "STARTING CHECKING ACCOUNT BALANCE WAS   $#####.##";BS
9930 PRINT USING "CLOSING CHECKING ACCOUNT BALANCE IS     $#####.##";BC
9935 PRINT
9940 PRINT "STARTING CK # THIS PERIOD WAS  ";CS
9950 PRINT "HIGHEST CK # THIS PERIOD WAS   ";CH: PRINT
9960 PRINT ">>> THE NEXT TIME THIS PROGRAM IS RUN RESPOND AS FOLLOWS:"
9965 PRINT "FOR PERIOD # ?";P+1
9970 PRINT "STARTING BALANCE THIS PERIOD ?";BC
9975 PRINT "STARTING CK # THIS PERIOD ?";CH+1 : PRINT
9980 PRINT "PROGRAM IS ABOUT TO TERMINATE.  VERIFY O.K. (Y/N)";
9985 INPUT Z$:  IF LEFT$(Z$,1)<>"Y" THEN 210
9999 END
```

## 5.4 SOLUTIONS TO THE PROJECTS IN CHAPTER 4

### Solution to the Project in Section 4.2

### Project *TAXRPT*

This program extends MONYLIST by adding another report which summarizes transactions sorted by the tax code they correspond to. Only the second half of the program is shown. The first half is the same as MONYLIST and MONEY2.

```
5000 ' ====================================================
5001 '              REPORT GENERATING PROGRAMS
5002 '
5010 CLS: F1=0
5020 PRINT "             <   REPORT MENU SELECTION   >"
5030 PRINT STRING$(60,"-")
5035 PRINT "      0 = EXIT FROM REPORT SECTION"
5040 PRINT "      1 = CHECKING ACCOUNT BALANCE"
5042 PRINT "      2 = PRINTED LIST OF ALL TRANSACTIONS"
5044 PRINT "      3 = TAX FORM REPORTS"
5050 PRINT
5055 INPUT "0, 1, 2, OR 3"; MS
5060 IF MS<0 OR MS>3 THEN 5055
5070 IF MS=0 THEN 210
5080 ON MS GOTO 6010, 7010, 8010
5090 GOTO 210
6010 GOTO 5010
7000 '-----------------------------------------------
7001 '   PRINTED LIST OF ALL TRANSACTIONS ENTERED
7002 '-----------------------------------------------
7010 PRINT "TURN PRINTER ON -- READY";:INPUT Z$
7012 LPRINT" ":LPRINT" ":LPRINT" ":LPRINT" "
7015 LPRINT "LIST OF ALL TRANSACTIONS FOR PERIOD";P;"OF";IY+1900"
7016 LPRINT "CHECKING ACCOUNT IS AT ";CA$;" BANK"
7017 LPRINT USING "STARTING BALANCE WAS $#####.##";BS
7025 LPRINT" "
7030 PF$="###  ##/##  $####.##  %              %"
7031 PF$=PF$+" %              % %   % ### ####"
7110 LPRINT" ":LPRINT"*** CHECKING ACCOUNT PAYMENTS ***":LPRINT" "
7120 LPRINT"CK#  DATE:  AMOUNT:   TO:                FOR";
7121 LPRINT":                FORM: TX%: ACCT:"
7130 LO=1: HI=99: S=CS: C1=1: GOSUB7510
7210 LPRINT" ":LPRINT"*** CASH PAYMENTS ***":LPRINT" "
7220 LPRINT"PAY# DATE:  AMOUNT:   TO:                FOR";
7221 LPRINT":                FORM: TX% ACCT:"
7230 LO=101: HI=149: S=PS: C1=101: GOSUB 7510
7310 LPRINT" ":LPRINT"*** CHECKING ACCOUNT DEPOSITS ***":LPRINT" "
7320 LPRINT"DEP# DATE:  AMOUNT:   FROM:              FOR";
7321 LPRINT":                FORM: TX% ACCT:"
7330 LO=151: HI=189: S=DS: C1=151: GOSUB 7510
7410 LPRINT" ":LPRINT"*** CASH RECEIPTS NOT DEPOSITED ***":LPRINT" "
7420 LPRINT"CSH# DATE:  AMOUNT:   FROM:              FOR";
7421 LPRINT":                FORM: TX% ACCT:"
7430 LO=191: HI=200: S=HS: C1=191: GOSUB 7510
```

```
7450 IF F1=1 THEN 8010
7480 LPRINT" ":LPRINT "CURRENT CHECKING ACCT. BALANCE IS ";
7481 LPRINT USING "$#####.##";BC
7490 GOTO 5010
7499 '
7500 '      PRINT SUBROUTINE
7501 '
7510 FOR K=LO TO HI
7520   IF GD(K)<0 THEN 7580
7524 IF F1=0 THEN 7530
7525 IF GF$(K)<>TF$ THEN 7560
7530   J=K+S-C1
7540   M=INT(GD(K)/100): D=GD(K)-M*100: A=GA(K)/100
7550   LPRINT USING PF$;J,M,D,A,GV$(K),GD$(K),GF$(K),GP(K),GC(K)
7560 NEXT K
7580 LPRINT STRING$(70,"-")
7590 RETURN
8000 '-----------------------------------------------------
8001 '     LIST OF TRANSACTIONS GROUPED BY TAX CODE
8002 '-----------------------------------------------------
8010 PRINT "REPORT FOR WHICH TAX FORM CODE ($$=EXIT)";:INPUT TF$
8020 IF LEN(TF$)<>2 PRINT "TYPE 2 CHARACTERS":GOTO 8010
8030 IF TF$="$$" THEN 8080
8040 PRINT "TURN PRINTER ON -- READY";:INPUT Z$
8042 LPRINT" ":LPRINT" ":LPRINT" ":LPRINT STRING$(70,"=")
8045 LPRINT "*** TRANSACTIONS FOR TAX-FORM CODE ";TF$;" ***"
8047 LPRINT STRING$(70,"=")
8050 F1=1          'FLAG TO ENABLE TESTS AT LINES 7450 & 7525
8055 GOTO 7025     'USE PRINT ROUTINE ALREADY WRITTEN
8070 REM LINES 7450, 7524, AND 7525 WERE ADDED TO ALLOW
8075 REM THIS REPORT TO SHARE USE OF THE PRINT ROUTINE AT 7000
8080 F1=0         'TURN OFF TAX RPT FLAG
8090 GOTO 5010
9000 ' ================================================
9900 ' ------------------------------------------------
9901 '              FINAL EXIT MESSAGES
9902 ' ------------------------------------------------
9910 CLS: PRINT "*** END OF PERIOD";P;"TRANSACTIONS FOR";1900+IY;"***"
9915 PRINT STRING$(50,"."): PRINT
9920 PRINT USING "STARTING CHECKING ACCOUNT BALANCE WAS  $#####.##";BS
9930 PRINT USING "CLOSING CHECKING ACCOUNT BALANCE IS    $#####.##";BC
9935 PRINT
9940 PRINT "STARTING CK # THIS PERIOD WAS   ";CS
9950 PRINT "HIGHEST CK # THIS PERIOD WAS    ";CH: PRINT
9960 PRINT ">>> THE NEXT TIME THIS PROGRAM IS RUN RESPOND AS FOLLOWS:"
9965 PRINT "FOR PERIOD # ?";P+1
9970 PRINT "STARTING BALANCE THIS PERIOD ?";BC
9975 PRINT "STARTING CK # THIS PERIOD ?";CH+1 : PRINT
9980 PRINT "PROGRAM IS ABOUT TO TERMINATE.  VERIFY O.K. (Y/N)";
9985 INPUT Z$:  IF LEFT$(Z$,1)<>"Y" THEN 210
9999 END
```

## Solutions to the Projects in Section 4.3

### Project *FILEBOX2*

This program is an improvement of FILEBOX. It is very similar, but it fields the I/0 buffer to put four logical (user) records in each physical record.

```
10 '*********************************************************************
20 '* FILEBOX2  (EXTENSION OF FILEBOX; PUTS 4 SUBRECORDS ON EACH RECORD) *
30 '*********************************************************************
110 CLEAR 500 : CLS
115 ON ERROR GOTO 989
120 PRINT "THIS PROGRAM ALLOWS YOU TO SAVE AND/OR RETRIEVE PAIRS"
130 PRINT "OF DATA OF THE FORM (KEY, INFO).      EXAMPLES:"
150 PRINT "KEY? SAUERBRATEN  INFO? OLD HEIDELBERG, 555-1234"
160 PRINT "KEY? ACE BONDS    INFO? SAFE DEP BOX 24, MARGINAL TRUST CO."
170 PRINT:PRINT"NAME OF DATA FILE";:INPUT F$
172 IF F$="" THEN 999
175 IF LEN(F$)>8 OR ASC(F$)<65 OR ASC(F$)>90 PRINT "BAD NAME":GOTO 170
177 PRINT ">>> WARNING: DO NOT EXIT THIS PROGRAM BY PRESSING BREAK <<<"
180 OPEN "R", 1, F$
190 IF LOF(1)>0 PRINT "NOTE: THIS IS AN OLD FILE": GOTO 310
195 PRINT "THIS IS A NEW FILE.  STAND BY FOR INITIALIZATION"
200 '---------------------------------------------------
201 '       INITIALIZE 25 PHYSICAL DISK RECORDS
202 '---------------------------------------------------
210 C%=0
215 FOR SR%=0 TO 3
216   FIELD 1, (SR%*63) AS D$, 61 AS T$, 2 AS C$
217   LSET T$=" ": LSET C$=MKI$(C%)
218 NEXT SR%
220 FOR PR%=1 TO 25
230   PUT 1, PR%
240 NEXT PR%
300 '---------------------------------------------------
301 '   DETERMINE RECORD #'S;  FIELD THE I/0 BUFFER
302 '---------------------------------------------------
310 PRINT "----------------------------------------------------"
320 PRINT "RECORD # (0 = EXIT)";: INPUT KN%
330 IF KN%=0 THEN 910
350 GOSUB 810      'CALCULATE PHYSICAL RECORD # & SUBRECORD #
360 IF PR%>25 PRINT "FILE SIZE EXCEEDED": GOTO 310
390 FIELD 1, (SR%*63) AS D$, 11 AS A$, 50 AS B$, 2 AS C$
400 '---------------------------------------------------
401 '     RETRIEVE DATA FROM DISK AND DISPLAY IT
402 '---------------------------------------------------
410 GET 1, PR%
415 C%=CVI(C$): IF C%<>0 THEN 420
417   PRINT"* RECORD #";KN%;"IS BLANK *": GOTO 455
420 PRINT "DATA FOR RECORD #"; KN%
430 PRINT "KEY  : ";A$
440 PRINT "INFO : ";B$
450 PRINT "THIS RECORD HAS BEEN ACCESSED";C%;"TIME(S)":C%=C%+1
451 LSET C$=MKI$(C%)
452 PUT 1, PR%          'PUT NEW # OF ACCESES ON RECORD
```

```
455 PRINT "......... DO YOU WISH TO ........."
460 PRINT"CHANGE(C), DELETE(D), OR RESUME(R)";:R$="":INPUT R$
461 IF R$="R" OR R$="" THEN 310
462 IF R$="C" THEN 510
463 IF R$="D" LSET A$=" ":LSET B$=" ":C%=0:GOTO 540
464 GOTO 460
500 '-------------------------------------------------
501 '        INPUT NEW DATA AND SAVE IT ON DISK
502 '-------------------------------------------------
510 C%=1: PRINT "TYPE NEW DATA FOR RECORD #";KN%
520 PRINT "KEY : ";:LINEINPUT K$ : LSET A$=K$
530 PRINT "INFO: ";:LINEINPUT I$ : LSET B$=I$
540 LSET C$=MKI$(C%)
550 PUT 1, PR%
590 GOTO 310
800 '-------------------------------------------------
801 '   CALCULATE PHYSICAL RECORD # AND SUBRECORD #
802 '-------------------------------------------------
810 PR%=INT((KN%-1)/4)+1
820 SR%=KN%-4*(PR%-1)-1
890 RETURN
900 '-------------------------------------------------
901 '  EXIT ROUTINE; LIST ALL KEYS ON FILE IF DESIRED
902 '-------------------------------------------------
910 PRINT "QUIT(Q), PRINT KEYS(P), OR RESUME(R)";:INPUT R$
915 IF R$="Q" THEN 990
920 IF R$="P" THEN 940
925 IF R$="R" THEN 310
930 GOTO 910
940 PRINT"PRINT ROUTINE MISSING AT PRESENT": GOTO 910
989 PRINT "ERROR DETECTED IN PROGRAM"
990 CLOSE
999 PRINT "* END OF PROGRAM -- ALL FILES ARE CLOSED *":END
```

### Project *FILEBOX3*

This is another extension of FILEBOX and *FILEBOX2.* It produces a list of key numbers and the associated key data items.

```
10 '**********************************************************
20 '*     FILEBOX3  (ADDS PRINTING OF KEYS TO FILEBOX2)      *
30 '**********************************************************
110 CLEAR 500 : CLS
115 ON ERROR GOTO 989
120 PRINT "THIS PROGRAM ALLOWS YOU TO SAVE AND/OR RETRIEVE PAIRS"
130 PRINT "OF DATA OF THE FORM (KEY, INFO).     EXAMPLES:"
150 PRINT "KEY? SAUERBRATEN  INFO? OLD HEIDELBERG, 555-1234"
160 PRINT "KEY? ACE BONDS    INFO? SAFE DEP BOX 24, MARGINAL TRUST CO."
170 PRINT:PRINT"NAME OF DATA FILE";:INPUT F$
172 IF F$="" THEN 999
175 IF LEN(F$)>8 OR ASC(F$)<65 OR ASC(F$)>90 PRINT "BAD NAME":GOTO 170
```

```
177 PRINT ">>> WARNING: DO NOT EXIT THIS PROGRAM BY PRESSING BREAK <<<"
180 OPEN "R", 1, F$
190 IF LOF(1)>0 PRINT "NOTE: THIS IS AN OLD FILE": GOTO 310
195 PRINT "THIS IS A NEW FILE.  STAND BY FOR INITIALIZATION"
200 '-----------------------------------------------------
201 '        INITIALIZE 25 PHYSICAL DISK RECORDS
202 '-----------------------------------------------------
210 C%=0
215 FOR SR%=0 TO 3
216   FIELD 1, (SR%*63) AS D$, 61 AS T$, 2 AS C$
217   LSET T$=" ": LSET C$=MKI$(C%)
218 NEXT SR%
220 FOR PR%=1 TO 25
230   PUT 1, PR%
240 NEXT PR%
300 '-----------------------------------------------------
301 '   DETERMINE RECORD #'S;  FIELD THE I/O BUFFER
302 '-----------------------------------------------------
310 PRINT "-------------------------------------------------------"
320 PRINT "RECORD # (0 = EXIT)";: INPUT KN%
330 IF KN%=0 THEN 910
350 GOSUB 810      'CALCULATE PHYSICAL RECORD # & SUBRECORD #
360 IF PR%>25 PRINT "FILE SIZE EXCEEDED": GOTO 310
390 FIELD 1, (SR%*63) AS D$, 11 AS A$, 50 AS B$, 2 AS C$
400 '-----------------------------------------------------
401 '     RETRIEVE DATA FROM DISK AND DISPLAY IT
402 '-----------------------------------------------------
410 GET 1, PR%
415 C%=CVI(C$): IF C%<>0 THEN 420
417   PRINT"* RECORD #";KN%;"IS BLANK *": GOTO 455
420 PRINT "DATA FOR RECORD #"; KN%
430 PRINT "KEY  : ";A$
440 PRINT "INFO : ";B$
450 PRINT "THIS RECORD HAS BEEN ACCESSED";C%;"TIME(S)":C%=C%+1
451 LSET C$=MKI$(C%)
452 PUT 1, PR%           'PUT NEW # OF ACCESES ON RECORD
455 PRINT "......... DO YOU WISH TO ........."
460 PRINT"CHANGE(C), DELETE(D), OR RESUME(R)";:R$="":INPUT R$
461 IF R$="R" OR R$="" THEN 310
462 IF R$="C" THEN 510
463 IF R$="D" LSET A$=" ":LSET B$=" ":C%=0:GOTO 540
464 GOTO 460
500 '-----------------------------------------------------
501 '         INPUT NEW DATA AND SAVE IT ON DISK
502 '-----------------------------------------------------
510 C%=1: PRINT "TYPE NEW DATA FOR RECORD #";KN%
520 PRINT "KEY : ";:LINEINPUT K$ : LSET A$=K$
530 PRINT "INFO: ";:LINEINPUT I$ : LSET B$=I$
540 LSET C$=MKI$(C%)
550 PUT 1, PR%
590 GOTO 310
800 '-----------------------------------------------------
801 '   CALCULATE PHYSICAL RECORD # AND SUBRECORD #
802 '-----------------------------------------------------
810 PR%=INT((KN%-1)/4)+1
820 SR%=KN%-4*(PR%-1)-1
890 RETURN
900 '-----------------------------------------------------
901 '  EXIT ROUTINE; LIST ALL KEYS ON FILE IF DESIRED
902 '-----------------------------------------------------
910 PRINT "QUIT(Q), PRINT KEYS(P), OR RESUME(R)";:INPUT R$
915 IF R$="Q" THEN 990
```

```
920 IF R$="P" THEN 940
925 IF R$="R" THEN 310
930 GOTO 910
940 FOR J%=0 TO 3
945   FIELD 1, (J%*63) AS D$, 11 AS KY$(J%), 50 AS E$, 2 AS A$(J%)
950 NEXT J%
952 N%=1: PRINT "*** TURN PRINTER ON -- READY";: INPUT Z$
953 LPRINT "  #: KEY          #: KEY           #: KEY           #: KEY"
954 LPRINT"------------------------------------------------------------------"
955 FOR PR%=1 TO 25
960   GET 1, PR%
965   FOR J%=0 TO 3:LPRINT USING "###: ";N%+J%;:LPRINT KY$(J%);:NEXT J%
968   LPRINT" ": N%=N%+4
970 NEXT PR%
975 GOTO 910
989 PRINT "ERROR DETECTED IN PROGRAM"
990 CLOSE
999 PRINT "* END OF PROGRAM -- ALL FILES ARE CLOSED *":END
```

## Solution to the Project in section 4.4

### Project *EDIT2000*

To write EDIT2000 start with EDIT1000, and then replace lines 450, 1311, and 1452 as shown. Next add the subroutine to simulate LINE INPUT, using line numbers 10000 to 10050 as given below.

```
10 ' *****************************************************
20 ' *     EDIT2000  (LINE EDITOR PROGRAM, PHASE 2)      *
30 ' *****************************************************
110 DEFINT A-Z                      'ALL VARIABLES TO BE INTEGER
120 CL=0 : HL=0                     'CL IS CURRENT LINE #
130 CLEAR 5000                      'HL IS HIGHEST LINE #
140 DIM T$(100), FP(100)            'T$( ) IS TEXT ARRAY BUFFER
190 CLS                             'CLS MEANS CLEAR SCREEN
191                                 'CLEAR 5000 MEANS RESERVE
192                                 '5000 BYTES FOR STRINGS
200 '-------------------------------------
201 '     GIVE DIRECTIONS; OPEN FILES
202 '-------------------------------------
210 GOSUB 610:PRINT "CURRENT LINE NUMBER NOW = 0.  TO APPEND TEXT"
215 PRINT "AT LINE # 1 RESPOND   CMD? A   AFTER? 0"
216 PRINT "..........................................."
300 '-------------------------------------
301 '     INITIALIZE LINKED LIST
302 '-------------------------------------
310 AV=2 : FP(1)=0                  'AV PTS TO TOP OF AVAIL LIST
```

```
320 FOR I=2 TO 99                    'FP(1) PTS TO TOP OF TEXT LIST
325   FP(I)=I+1                      '0 SIGNALS END OF LIST
330 NEXT I
340 FP(100)=0
400 '---------------------------------------
401 '     INPUT COMMAND
402 '---------------------------------------
450 PRINT"CMD ";: INPUT L$   'L$ HOLDS CMD CODE
455 L1=CL : L2=CL                    'DEFAULT VALUES
500 '---------------------------------------
501 '     INTERPRET COMMAND
502 '---------------------------------------
510 IF L$="I" THEN 1010
511 IF L$="H" GOSUB 610: GOTO 450
515 IF L$="V" PRINT USING"##:";CL;:PRINT T$(CP):GOTO 450
529 '
530 IF L$="A"  V$="AFTER" ELSE IF L$="G"  V$="WHERE" ELSE V$="FROM"
532 PRINT TAB(10);CHR$(27);V$;:INPUT L1: L2=L1
534 IF L$="A" THEN 1310
536 IF L$="G" THEN 1610
539 '
540 PRINT TAB(20);CHR$(27);"TO";:INPUT L2
542 IF L$="L" THEN 1610
544 IF L$="D" THEN 1510
546 IF L$="C" THEN 1410
548 IF L$="P" THEN 1710
590 PRINT "NO SUCH COMMAND" : GOTO 450
601 '
602 '........... HELP SUBROUTINE ...........
610 PRINT"AFTER 'CMD?' TYPE A(PPEND), C(HANGE), D(ELETE), G(O TO)"
620 PRINT"H(ELP), I(NFO), L(IST), P(RINT), T(RANSFER), OR V(ERIFY)."
640 PRINT"RESPOND TO OTHER ? PROMPTS WITH A LINE #.  RESPONDING WITH"
650 PRINT"'ENTER' GIVES CURRENT LINE #. USE I CMD TO FIND CURRENT #."
690 RETURN
1000 '---------------------------------------
1001 '     I COMMAND (INFORMATION)
1002 '---------------------------------------
1010 PRINT "CURRENT LINE IS #";CL;"    HIGHEST LINE IS #";HL
1090 GOTO 450   'RETURN FOR NEW CMD
1300 '---------------------------------------
1301 '     A COMMAND (APPEND AFTER LINE L1)
1302 '---------------------------------------
1310 IF L1>HL PRINT "LINE # TOO HIGH":GOTO 450
1311 PRINT USING "##>";L1+1;:GOSUB 10010   'SIMULATED LINE INPUT
1312 IF LEN(B$)=0 THEN B$=" "
1315 IF B$="." THEN 1390
1320 IF AV=0 PRINT "BUFFER FULL": GOTO 1390
1325 CL=L1+1
1330 I=1 : K=1
1335 IF FP(I)=0 OR K=CL THEN 1350        'I IS LINE POINTER
1340   I=FP(I) : K=K+1 : GOTO 1335       'K IS LINE COUNTER
1345 '
1350 J=AV : AV=FP(J)
1355 FP(J)=FP(I)
1360 FP(I)=J
1365 T$(J)=B$
1370 CP=J : HL=HL+1                'CP IS PTR TO CURRENT LINE
1375 IF CL=HL THEN HP=CP           'HP IS PTR TO HIGHEST LINE
1380 L1=CL : GOTO 1310
1390 GOTO 450   'RETURN FOR NEW CMD
1400 '---------------------------------------
1401 '     C COMMAND (CHANGE LINES L1 TO L2)
1402 '---------------------------------------
1410 CL=L1
1415 IF L2>HL PRINT "2ND LINE TOO HIGH":GOTO 450
```

```
1420 I=1 : K=1
1425 IF FP(I)=0 OR K=CL THEN 1440
1430   I=FP(I) : K=K+1 : GOTO 1425
1435 '
1440 IF FP(I)=0 PRINT "NO SUCH LINE" : GOTO 1490
1445   CP=FP(I)
1450   PRINT USING "##:";CL; : PRINT T$(CP)
1452   PRINT USING "##>";CL;:GOSUB 10010: T$(CP)=B$
1455 IF CL<L2 THEN CL=CL+1 : GOTO 1420
1490 GOTO 450   'RETURN FOR NEW CMD
1500 '----------------------------------------
1501 '     D COMMAND (DELETE LINES L1 TO L2)
1502 '----------------------------------------
1510 CL=L1
1511 IF L2>HL THEN L2=HL
1512 FOR C=0 TO L2-L1
1515 I=1 : K=1
1520 IF FP(I)=0 OR K=CL THEN 1540
1525   I=FP(I) : K=K+1 : GOTO 1520
1530 '
1540 IF FP(I)=0 PRINT "NO TEXT TO DELETE": GOTO 1585
1545    J=FP(I)
1550    FP(I)=FP(J)
1555    FP(J)=AV
1560    AV=J
1565    HL=HL-1
1570    CP=I
1575   IF HL=CL THEN HP=CP
1580 NEXT C
1585 CL=L1-1: IF CL<0 THEN CL=0
1590 GOTO 450   'RETURN FOR NEW CMD
1600 '----------------------------------------
1601 '     L COMMAND (LIST LINES L1 TO L2)
1602 '----------------------------------------
1610 I=FP(1) : K=1
1615 IF L2>HL THEN L2=HL
1620 IF I=0 PRINT "BUFFER EMPTY" : GOTO 1690
1630   IF K<L1  THEN 1650
1640   CP=I: PRINT USING "##:";K;: PRINT T$(I)
1650   I=FP(I) : K=K+1
1655   IF K>L2 THEN 1680
1660   GOTO 1630
1680 CL=K-1
1690 GOTO 450   'RETURN FOR NEW CMD
1700 '----------------------------------------
1701 '     P COMMAND (PRINTS ON LINE PRINTER)
1702 '----------------------------------------
1710 I=FP(1) : K=1
1715 IF L2>HL THEN L2=HL
1720 IF I=0 PRINT "BUFFER EMPTY" : GOTO 1790
1725 PRINT "TURN ON PRINTER -- READY";:INPUT Z$
1730   IF K<L1 THEN 1750
1740   CP=I: LPRINT T$(I)
1750   I=FP(I) : K=K+1
1755   IF K>L2 THEN 1780
1760   GOTO 1730
1780 CL=K-1
1790 GOTO 450   'RETURN FOR NEW CMD
1799 '=====  BLOCKS RESERVED FOR FUTURE COMMANDS  =====
1801 '     M COMMAND (MOVE BLOCKS OF TEXT)
1901 '     E COMMAND (ECHO BLOCKS OF TEXT)
2001 '     R COMMAND (READ FILE INTO BUFFER AFTER CL)
2101 '     W COMMAND (WRITE BUFFER ONTO FILE)
2201 '     F COMMAND (FIND LINE WITH /TEXT/)
2301 '     C/ COMMAND (CHANGE /TEXT1/ TO /TEXT2/)
```

```
10000 REM*** SUBROUTINE TO SIMULATE 'LINE INPUT L$' ***
10010 B$=""
10020 V9$=INKEY$: IF V9$="" THEN 10020
10025 IF B$<>"" PRINT CHR$(24);
10030 IF ASC(V9$)=13 THEN 10050
10035 PRINT V9$;: PRINT CHR$(95);
10040 B$=B$+V9$: GOTO 10020
10050 PRINT " ":RETURN
```

### Solution to Project in Section 4.6

**Project** *EDIT5000*

This program is an extension of EDIT1000. It adds several new commands: T(ransfer), C(hange)/old/new/, and F(ind).

```
10 ' ****************************************************
20 ' *     EDIT5000  (LINE EDITOR PROGRAM, PHASE 5)      *
30 ' ****************************************************
110 DEFINT A-Z                        'ALL VARIABLES TO BE INTEGER
120 CL=0 : HL=0                       'CL IS CURRENT LINE #
130 CLEAR 5000                        'HL IS HIGHEST LINE #
140 DIM T$(100), FP(100)              'T$( ) IS TEXT ARRAY BUFFER
190 CLS                               'CLS MEANS CLEAR SCREEN
191                                   'CLEAR 5000 MEANS RESERVE
192                                   '5000 BYTES FOR STRINGS
200 '---------------------------------------
201 '     GIVE DIRECTIONS; OPEN FILES
202 '---------------------------------------
210 GOSUB 610:PRINT "CURRENT LINE NUMBER NOW = 0.  TO APPEND TEXT"
215 PRINT "AT LINE # 1 RESPOND   CMD? A   AFTER? 0"
216 PRINT "..............................................."
300 '---------------------------------------
301 '     INITIALIZE LINKED LIST
302 '---------------------------------------
310 AV=2 : FP(1)=0                    'AV PTS TO TOP OF AVAIL LIST
320 FOR I=2 TO 99                     'FP(1) PTS TO TOP OF TEXT LIST
325   FP(I)=I+1                       '0 SIGNALS END OF LIST
330 NEXT I
340 FP(100)=0
400 '---------------------------------------
401 '     INPUT COMMAND
402 '---------------------------------------
450 PRINT"CMD? ";:LINE INPUT L$      'L$ HOLDS CMD CODE
455 L1=CL : L2=CL                     'DEFAULT VALUES
```

```
500 '---------------------------------------
501 '     INTERPRET COMMAND
502 '---------------------------------------
510 IF L$="I" THEN 1010
511 IF L$="H" GOSUB 610: GOTO 450
512 IF L$="T" THEN 32000
515 IF L$="V" PRINT USING"##:";CL;:PRINT T$(CP):GOTO 450
520 IF LEFT$(L$,2)="C/" THEN 2310
525 IF LEFT$(L$,2)="F/" THEN 2210
529 '
530 IF L$="A"  V$="AFTER" ELSE IF L$="G"  V$="WHERE" ELSE V$="FROM"
532 PRINT TAB(10);CHR$(27);V$;:INPUT L1: L2=L1
534 IF L$="A" THEN 1310
536 IF L$="G" THEN 1610
539 '
540 PRINT TAB(20);CHR$(27);"TO";:INPUT L2
542 IF L$="L" THEN 1610
544 IF L$="D" THEN 1510
546 IF L$="C" THEN 1410
548 IF L$="P" THEN 1710
590 PRINT "NO SUCH COMMAND" : GOTO 450
601 '
602 '........... HELP SUBROUTINE ...........
610 PRINT"AFTER 'CMD?' TYPE A(PPEND), C(HANGE), D(ELETE), G(O TO)"
620 PRINT"H(ELP), I(NFO), L(IST), P(RINT), T(RANSFER), OR V(ERIFY)."
640 PRINT"RESPOND TO OTHER ? PROMPTS WITH A LINE #.  RESPONDING WITH"
650 PRINT"'ENTER' GIVES CURRENT LINE #. USE I CMD TO FIND CURRENT #."
690 RETURN
1000 '---------------------------------------
1001 '     I COMMAND (INFORMATION)
1002 '---------------------------------------
1010 PRINT "CURRENT LINE IS #";CL;"    HIGHEST LINE IS #";HL
1090 GOTO 450    'RETURN FOR NEW CMD
1300 '---------------------------------------
1301 '     A COMMAND (APPEND AFTER LINE L1)
1302 '---------------------------------------
1310 PRINT USING"##>";L1+1;: LINE INPUT B$
1312 IF LEN(B$)=0 THEN B$=" "
1315 IF B$="." THEN 1390
1320 IF AV=0 PRINT "BUFFER FULL": GOTO 1390
1325 CL=L1+1
1330 I=1 : K=1
1335 IF FP(I)=0 OR K=CL THEN 1350        'I IS LINE POINTER
1340   I=FP(I) : K=K+1 : GOTO 1335       'K IS LINE COUNTER
1345 '
1350 J=AV : AV=FP(J)
1355 FP(J)=FP(I)
1360 FP(I)=J
1365 T$(J)=B$
1370 CP=J : HL=HL+1                'CP IS PTR TO CURRENT LINE
1375 IF CL=HL THEN HP=CP           'HP IS PTR TO HIGHEST LINE
1380 L1=CL : GOTO 1310
1390 GOTO 450    'RETURN FOR NEW CMD
1400 '---------------------------------------
1401 '     C COMMAND (CHANGE LINES L1 TO L2)
1402 '---------------------------------------
1410 CL=L1
1415 IF L2>HL PRINT "2ND LINE TOO HIGH":GOTO 450
1420 I=1 : K=1
1425 IF FP(I)=0 OR K=CL THEN 1440
1430   I=FP(I) : K=K+1 : GOTO 1425
1435 '
```

```
1440 IF FP(I)=0 PRINT "NO SUCH LINE" : GOTO 1490
1445   CP=FP(I)
1450   PRINT USING "##:";CL; : PRINT T$(CP)
1452   PRINT USING "##>";CL; : LINE INPUT T$(CP)
1455 IF CL<L2 THEN CL=CL+1 : GOTO 1420
1490 GOTO 450   'RETURN FOR NEW CMD
1500 '---------------------------------------
1501 '      D COMMAND (DELETE LINES L1 TO L2)
1502 '---------------------------------------
1510 CL=L1
1511 IF L2>HL THEN L2=HL
1512 FOR C=0 TO L2-L1
1515 I=1 : K=1
1520 IF FP(I)=0 OR K=CL THEN 1540
1525   I=FP(I) : K=K+1 : GOTO 1520
1530 '
1540 IF FP(I)=0 PRINT "NO TEXT TO DELETE": GOTO 1585
1545    J=FP(I)
1550    FP(I)=FP(J)
1555    FP(J)=AV
1560    AV=J
1565    HL=HL-1
1570    CP=I
1575   IF HL=CL THEN HP=CP
1580 NEXT C
1585 CL=L1-1: IF CL<0 THEN CL=0
1590 GOTO 450   'RETURN FOR NEW CMD
1600 '---------------------------------------
1601 '      L COMMAND (LIST LINES L1 TO L2)
1602 '---------------------------------------
1610 I=FP(1) : K=1
1615 IF L2>HL THEN L2=HL
1620 IF I=0 PRINT "BUFFER EMPTY" : GOTO 1690
1630   IF K<L1  THEN 1650
1640   CP=I: PRINT USING "##:";K;: PRINT T$(I)
1650   I=FP(I) : K=K+1
1655   IF K>L2 THEN 1680
1660   GOTO 1630
1680 CL=K-1
1690 GOTO 450   'RETURN FOR NEW CMD
1700 '---------------------------------------
1701 '      P COMMAND (PRINTS ON LINE PRINTER)
1702 '---------------------------------------
1710 I=FP(1) : K=1
1715 IF L2>HL THEN L2=HL
1720 IF I=0 PRINT "BUFFER EMPTY" : GOTO 1790
1725 PRINT "TURN PRINTER ON -- READY";:INPUT Z$
1730   IF K<L1 THEN 1750
1740   CP=I: LPRINT T$(I)
1750   I=FP(I) : K=K+1
1755   IF K>L2 THEN 1780
1760   GOTO 1730
1780 CL=K-1
1790 GOTO 450   'RETURN FOR NEW CMD
1799 '=====  BLOCKS RESERVED FOR FUTURE COMMANDS  =====
1801 '      M COMMAND (MOVE BLOCKS OF TEXT)
1901 '      E COMMAND (ECHO BLOCKS OF TEXT)
2001 '      R COMMAND (READ FILE INTO BUFFER AFTER CL)
2101 '      W COMMAND (WRITE BUFFER ONTO FILE)
2201 '      F COMMAND (FIND LINE WITH /TEXT/)
2210 LP=LEN(L$)-3
2215 P$=MID$(L$,3,LP): IF LP=0 P$=" "
```

```
2220 I=1  : K=1
2225 I=FP(I)
2230 IF I=0 PRINT "NOT FOUND": GOTO 450
2235   FOR Q=1 TO LEN(T$(I))
2240     IF MID$(T$(I),Q,LP)=P$ THEN 2255
2245   NEXT Q
2250   K=K+1 : GOTO 2225
2255 CP=I : CL=K
2260 PRINT USING"##:";CL;:PRINT T$(CP):GOTO450
2301 '     C/ COMMAND (CHANGE /TEXT1/ TO /TEXT2/)
2310 R$="": P=3
2311 IF LEN(L$)<3 PRINT "TYPE IN FORM C/OLD/NEW/": GOTO 450
2312 IF RIGHT$(L$,1)<>"/"PRINT "NEED SLASH AT END":GOTO450
2313 IF RIGHT$(L$,2)="//"PRINT"NEED TEXT BETWEEN LAST 2 SLASHES":GOTO450
2315 X$=MID$(L$,P,1)
2320 IF X$="/" THEN 2330
2325   R$=R$+X$: P=P+1: GOTO 2315
2330 N$=""
2335 FOR Q=P+1 TO LEN(L$)-1
2340   N$=N$+MID$(L$,Q,1)
2345 NEXT Q
2350 LT=LEN(T$(CP)): LR=P-3
2360 FOR Q=1 TO LT
2365   IF MID$(T$(CP),Q,LR)=R$ THEN 2375
2370 NEXT Q
2375 IF Q>LT PRINT R$;" NOT FOUND": GOTO 450
2380 T$(CP)=LEFT$(T$(CP),Q-1)+N$+RIGHT$(T$(CP),LT-Q-LR+1)
2385 PRINT USING"##:";CL;: PRINT T$(CP)
2390 GOTO 450
31997 '---------------------------------------
31998 '     T COMMAND (TRANSFER SCREEN IMAGE TO PRINTER)
31999 '---------------------------------------
32000 N9=15359
32005 PRINT CHR$(27);:PRINT TAB(40);" "
32010 FOR L9=1 TO 16
32020 L9$=""
32030 FOR W9=64 TO 1 STEP -1
32040 IF PEEK(N9+W9) > 32 THEN 32060
32050 NEXT W9
32060 FOR K9=1 TO W9
32070 L9$=L9$+CHR$(PEEK(N9+K9))
32080 NEXT K9
32090 LPRINT L9$
32100 N9=N9+64
32110 NEXT L9
32115 CLS
32120 GOTO 450
```

# APPENDIX A

**Summary of Level II BASIC**

Variable Names (examples)

| | | | | |
|---|---|---|---|---|
| A | Z | A1 | Z9 | |
| AC | ZT | B(I) | X(I,J) | |
| N(X(I)) | Q(R(I),C(J)) | S(X,Y,Z) | A$ | A! |
| B$(I) | BC(I,J) | D2$(R,C,L) | A% | A# |

Variable Types (see DEFINT, DEFSNG, etc.)

| | Type | Example Values |
|---|---|---|
| A | Undeclared (numeric) | 1, −1, .99, 75.95 (same as single precision, below) |
| A$ | String | "HI!", "ANYTHING (UP TO 255 CHARACTERS)−−+=!@#%&:;?" |
| A% | Integer | 1, −1, 99, −32768, +32767 |

| | | |
|---|---|---|
| A! | Single Precision | 1.0, −1.0, .999999, −1.701411E+38, 804.123 |
| A# | Double Precision | 1.0, −1.0, −1.701411832544556D+38, 804.123456123456 |

Operators and Relations

| | | |
|---|---|---|
| + | > | AND |
| − | < | OR |
| * | = | NOT |
| / | >= | |
| ↑ | <= | |
| | < > | |

Commands (ln=line number, inc=increment, var=variable, exp=expression, n=number)

| Command | Explanation | Example |
|---|---|---|
| AUTO ln, inc | Automatic line numbering:<br>To get 10, 20, 30 . . .<br>To get 5, 10, 15, 20 . . .<br>To get 100, 102, 104, . . .<br>To exit from AUTO press BREAK. | <br>AUTO<br>AUTO 5,5<br>AUTO 100, 2 |
| CLEAR n | Resets all variables to zero or null. Specifies bytes of memory to be reserved for string storage. (Default is 50 bytes.) | CLEAR<br>CLEAR 500<br>10 CLEAR 200 |
| CLOAD "filename" | Erases program in memory. Searches for and loads file stored on a cassette. Only first letter of name is used. If no name is given first file encountered is loaded. | CLOAD<br>CLOAD"1"<br>CLOAD"A"<br>CLOAD"ANIMAL" |
| CLOAD?"filename" | Does not erase memory. Searches for and compares the named file with the file currently in memory. | CLOAD?<br>CLOAD?"A" |
| CONT | Continues execution of a program after STOP statement or hitting BREAK. | CONT |
| CSAVE "filename" | Stores the program currently in memory on cassette tape, labels it with the filename. | CSAVE"A" |

| | | |
|---|---|---|
| DELETE ln-ln | Erases program lines from memory.<br>To erase a range of lines-<br>To erase all lines up to 50-<br>To erase line just entered- | DELETE 10<br>DELETE 10-50<br>DELETE -50<br>DELETE. |
| EDIT ln | Puts the computer in edit mode, allowing use of the subcommands: L, X, I, A, E, Q, H, nD, nC, nSc, and nKc. Hit shift ↑ to exit from a subcommand. Hit ENTER or type E to exit from edit mode. | EDIT 100 |
| LIST ln-ln | Displays program lines currently in memory. Use shift @ to halt the display of a long program, hit any key to resume.<br>To display line just entered-<br>To display whole program- | LIST 10<br>LIST 10-100<br>LIST 50-<br>LIST -100<br>LIST.<br>LIST |
| LLIST ln-ln | Same as LIST but output is sent to the printer. | LLIST 10<br>LLIST 10-100<br>LLIST 50-<br>LLIST -100<br>LLIST.<br>LLIST |
| NEW | Erases all program lines. Sets variables to zero or null. | NEW |
| RUN ln | Executes the program, starting at that line number. If no line number given, it starts at the beginning. Also used as a statement. Resets all variables (does a CLEAR). | RUN 100<br>RUN<br>500 RUN 10 |
| SYSTEM | Puts the computer in system mode. Allows loading and running of machine language programs. | SYSTEM |
| TROFF | Turns off the Trace function. | TROFF |
| TRON | Turns on the Trace function. Programs run subsequently will display currently executed line number in addition to program's output. Also used as a statement. | TRON<br>RUN<br><br>10 TRON<br>...<br>50 TROFF |

| Statement | Explanation | Example |
|---|---|---|
| CLEAR | See *Commands* | |
| CLS | Clears the video screen | 35 CLS |
| DATA item list | Allows you to store data items within the program. See READ. | 1000 DATA "APPLES", "LB.",.49 |
| DEFINT letter range | Variable names starting with letters in the range given will be treated as integer type. | 10 DEFINT A-L<br>10 DEFINT A,B,C |
| DEFSNG letter range | Variable names starting with letters in the range given will be treated as single precision type. | 10 DEFSNG M-Q<br>10 DEFSNG M,N |
| DEFDBL letter range | Variable names starting with letters in the range given will be treated as double precision type. | 10 DEFDBL R-W<br>10 DEFDBL R |
| DEFSTR letter range | Variable names starting with letters in the range given will be treated as string type. | 10 DEFSTR X-Z<br>10 DEFSTR X,Y,Z |
| DIM var (n,n,...) | Reserves memory space for arrays. Default is 11 elements, 0-10. | 10 DIM A(100), B(10,20), C$(100)<br>15 N = 20<br>20 DIM D(N,50) |
| ELSE | See IF ... THEN ... ELSE | |
| END | Optional. Terminates execution of a program normally. | 9999 END |
| ERL | See *Special Functions.* | |
| ERR/2+1 | See *Special Functions.* | |
| ERROR code | Simulates the specified error during execution. Used to test an ON ERROR GOTO statement. | 150 ERROR 1 |

| | | |
|---|---|---|
| FOR...TO...STEP | Sets up a loop. The program statements following it are repeated for as many steps (increments) as it takes to get from the initial value of the counter variable to the final value. A NEXT statement is required at the end of the sequence of statements. Loop executes at least once. | 10 FOR I=1 TO 10<br>10 FOR J=0 TO 100 STEP 5<br>10 FOR K=50 TO −50 STEP −1<br>10 FOR X=1.5 TO 15.5 STEP .5<br>10 FOR Q=1 TO N STEP R+2<br>10 FOR T=Q TO P−1 |
| GOSUB ln | Branches to the line number given but 'remembers' where it was. Subroutine at that line number should end with a RETURN statement causing a branch back to the statement following the GOSUB. | 10 GOSUB 1000<br>20 (rest of the program)<br>...<br>999 END<br>1000 (subroutine)<br>1020 RETURN |
| GOTO ln | Branches to the line number given. Used as a command to run a program without resetting all variables. | 50 GOTO 5<br>GOTO 10 |
| IF...THEN...ELSE | Conditional statement. Computer tests a logical or relational expression; if true it does the THEN part(s). If false, it does the ELSE part(s) (optional). It then goes on to the following instruction. | 10 IF X<Y THEN 100<br>10 IF B$="Y" THEN Z=1 :PRINT "YES"<br>10 IF Y=X THEN 100 ELSE 1000<br>10 IF Y>M THEN M=Y ELSE N=Y:D=C |
| INPUT item list | Allows person to input values from the keyboard. Stops execution and prints ?. Items can be typed in separated by commas or ENTER. | 10 INPUT X<br>10 INPUT A$,X, B$, Y<br>10 INPUT "TYPE YOUR NAME";N$ |
| INPUT#−1, item list | Allows values to be read from a tape cassette. | 10 INPUT#−1,X, Y,Z,P$,D$,Q$ |

| | | |
|---|---|---|
| LET var=value | Assignment statement. LET is optional. Stores value in variable name mentioned. | 10 LET Z=10026.359<br>10 LET B$="AMOUNT"<br>10 X=X+1<br>10 Y=Y−(Y *.1)<br>10 S=S+A |
| LPRINT item list | Like PRINT but output is sent to the printer. LPRINT @ does not work but all other versions of PRINT statements do. See PRINT. | 10 LPRINT "SUMMARY"<br>20 LPRINT X, Y, X + Y |
| NEXT var | Required at the end of a sequence of statements to be repeated by a FOR loop. Counter variable name is optional. | 10 FOR X=2 TO 102 STEP 2<br>20 (statements)<br>...<br>50 NEXT X |
| ON ERROR GOTO ln | Allows you to write a special routine at the given line number, ending it with RESUME. This will be executed when an error in execution occurs—instead of the usual BREAK. | 5 ON ERROR GOTO 100<br>10 (rest of program)<br>...<br>100 (special routine)<br>120 RESUME |
| ON var GOSUB ln, ln,... | Like ON var GOTO but control returns after executing the subroutine to the statement following the ON var GOSUB. | 10 ON Y GOSUB 1000, 2000, 3000<br>20 (rest of program) |
| ON var GOTO ln, ln,... | Allows branching to one of many line numbers, depending on the value of var. If var=1, the first line number is used, if var=2, the second, etc.; var can be an expression. | 10 ON X GOTO 100, 200, 300 |
| OUT port, value | Outputs a value to the specified port, which is connected to a peripheral device. Ports are numbered 0-255, value can be 0-255. See *Special Functions,* INP (port). | 250 OUT 12,10<br>250 OUT 255,A |
| POKE address,value | Loads a value (from 0-255) into the given address. See *Special Functions*, PEEK (address). | 500 POKE 32765,211<br>500 POKE 32764,B |

| | | |
|---|---|---|
| PRINT item list | Prints an item or list of items on the display. Does calculations. Commas cause items to be spread out, semicolons keep them close together. A semicolon at the end will cause the next thing printed to begin where the last PRINT statement left off, otherwise the next line is cleared and used. | 10 PRINT 10279+64823<br>10 PRINT"HI!"<br>10 PRINT X,Y,Z<br>10 PRINT X+3*X<br>10 PRINT "AMT:";X, "DISC:";Y<br>10 PRINT "X+Y="; |
| PRINT @ pos,item | Like PRINT but specifies where on the display screen printing is to start. 0 is the upper left corner position, 1023 is the lower right corner. Use a semicolon at the end to prevent clearing the next line. | 10 PRINT @ 510, "MIDDLISH"; |
| PRINT TAB(n) item | Like PRINT but moves cursor to a specified position on the current line (0,255). If the value is greater than the line length on the display or printer, the next line will be used. | 50 PRINT TAB(5)<br>50 PRINT TAB(X) "TABLE"<br>50 PRINT TAB (T*5) Z |
| PRINT USING string;item list | Allows you to specify a format using a string. Special characters (#, **, $$, +, −, !, % %, and ↑↑↑↑) are used within the string to show fields to be filled with variables. | 10 A$= "$$##,###.#####"<br>20 PRINT USING A$;A,B<br><br>10 PRINT USING "% %";B$<br><br>10 C$="NAME IS %  %"<br>20 PRINT USING C$;N$ |
| PRINT#−1, item list | Allows values of the given variables to be recorded on a tape cassette. | 10 PRINT#−1,A,B,C,D$,E$,F$ |
| RANDOM | Causes the random number function RND to produce a new set of random numbers on successive runs. | 5 RANDOM<br>10 X = RND(0) |

| | | |
|---|---|---|
| READ item list | Assigns values from DATA statements to the given variables. Keeps track of which values have already been read; always attempts to read the next unused value unless a RESTORE statement is encountered. | 10 READ A$,X,B$,Y<br>...<br>1000 DATA "APPLE",.25, "ORANGE",.50<br>1001 DATA 0,0,0,0 |
| REM | Remark statement. Ignored by the computer. Used to explain parts of a program. Abbrev. ' . | 10 REM---MAIN LOOP---<br>20 '----INITIALIZE VARIABLES---- |
| RESTORE | Causes the next READ statement to start over at the beginning of the data. See READ. | 20 IF A$="0" THEN RESTORE |
| RESUME | See ON ERROR GOTO. | 100 RESUME<br>100 RESUME NEXT<br>100 RESUME ln |
| RETURN | Ends a subroutine; sends control back to the statement following the GOSUB. | ...<br>100 (subroutine)<br>120 RETURN |
| RUN | See *Commands*. | |
| STOP | Interrupts execution and prints a BREAK IN ln message. To restart type CONT. | 500 STOP |
| THEN | See IF...THEN...ELSE | |
| TROFF | See *Commands*. | |
| TRON | See *Commands*. | |

*String Functions*

| Function | Explanation | Example/Result Returned |
|---|---|---|
| ASC(string) | Returns the ASCII code (in decimal) for the first character of the string. | PRINT ASC("A")<br>65<br>PRINT ASC("AARDVARK")<br>65<br>T$="A":PRINT ASC(T$)<br>65 |
| CHR$(exp) | Returns the character which has the ASCII code given. If the code is one for a graphics character or a control function, that function is performed. | PRINT CHR$(65)<br>A<br>10 CLS: PRINT CHR$(23)<br>(changes display to double-sized letters) |
| FRE(string) | Returns the number of bytes of string storage space currently available. Argument in parenthesis has no meaning, it's a dummy. | PRINT FRE(D$)<br>25 |
| INKEY$ | Scans the keyboard, returns one character if a key is being pressed, or a null string if no key is being pressed. ENTER is not needed. | 10 A$=INKEY$<br>20 IF A$="" THEN 10<br>30 IF A$="B" PRINT "BANG!" |
| LEFT$(string,n) | Returns a substring of the string, n characters long, starting at the left. | A$="PEARTREE"<br>PRINT LEFT$ (A$,4)<br>PEAR<br><br>PRINT LEFT$ (A$,LEN(A$)−2)<br>PEARTR |

| | | |
|---|---|---|
| LEN(string) | Returns the number of characters in the string. See LEFT$. | PRINT LEN ("PEARTREE")<br>8 |
| MID$(string,p,n) | Returns a substring of the string, n characters long, starting with the pth character. If n is not given, the rest of the string is returned. | A$= "RENOVATION"<br>PRINT MID$(A$,3,4)<br>NOVA<br>PRINT MID$(A$,3)<br>NOVATION |
| RIGHT$(string,n) | Returns a substring of the string, n characters long, ending at the right. | PRINT RIGHT$ (A$,3)<br>ION |
| STR$(exp) | Makes a string which looks like the given numeric expression. | A=−5.2376<br>B$="////"<br>PRINT B$+STR$(A) +B$<br>////−5.2376//// |
| STRING$(n,x) | Produces a string of x's, n characters long. x can be a quoted character or a number. If x is a number, CHR$(x) is printed n times. | PRINT STRING$ (10,"?")<br>?????????? |
| VAL(string) | Returns the number represented by the string. Trailing characters are ignored. | PRINT VAL ("123 FOURTH ST.")<br>123<br>PRINT VAL ("1.32E+07")<br>1.32<br>PRINT VAL ("75.95")+5<br>80.95 |

*Numeric Functions*

| | | |
|---|---|---|
| ABS(x) | Absolute value of x. x may be a numeric variable, expression, or constant. | PRINT ABS (−23), ABS(49.2)<br>23 49.2 |
| ATN(x) | Returns angle whose tangent is x (in radians). Multiply by 57.29578 to get degrees. | PRINT ATN(15)<br>1.50423 |
| CDBL(x) | Converts x to a double-precision number. | PRINT CDBL (1)/3<br>.3333333333333333 |
| CINT(x) | Converts x to an integer. x must be between −32768 and +32767. Returns the greatest integer less than x. | PRINT CINT (327.549); CINT (−3.2)<br>327 − 4 |

| | | |
|---|---|---|
| COS(x) | Cosine of angle x (in radians). | PRINT COS(2.2)<br>−.588501 |
| EXP(x) | Returns $e^x$, the inverse of LOG. | PRINT EXP(1.5)<br>(4.48169) |
| FIX(x) | Makes x an integer by deleting the fractional part. | PRINT FIX (1234.5678), FIX(−2.34)<br>1234 −2 |
| INT(x) | Converts x to an integer. x is not limited to −32768 to +32767. The greatest integer less than x is returned. | PRINT INT (98765.43), INT(−2.34)<br>98765 −3 |
| LOG(x) | Returns log(x), the inverse of EXP. | PRINT LOG (4.48169)<br>1.5 |
| RND(0) | Returns a random number between 0 and 1. Unless the statement RANDOM is used, the numbers generated may be the same on successive runs of the program. | PRINT RND(0)<br>.325471 |
| RND(n) | Where n is an integer from 1 to 32767, returns a random integer between 1 and n inclusive. | PRINT RND (6);RND(6); RND(6)<br>4 6 1 |
| SGN(x) | Returns −1 if x is negative, 0 if x is 0 and +1 if x is positive. | PRINT SGN (−35);SGN(0); SGN(456)<br>−1 0 +1 |
| SIN(x) | Sine of angle x (in radians). To convert x in degrees use X * .0174533 | PRINT SIN(.8)<br>.717356 |
| SQR(x) | Square root of x. | PRINT SQR(25)<br>5 |
| TAN(x) | Tangent of angle x (in radians). | PRINT TAN(2)<br>−2.18504 |

*Special Functions*

| | | |
|---|---|---|
| ERL | Returns the line number in which an error has occurred. Used with ON ERROR GOTO. | 10 PRINT "ERROR AT LINE";ERL |

| | | |
|---|---|---|
| ERR/2+1 | Like ERL but ERR returns a value related to the code of the error; ERR/2+1= true error code. | 10 PRINT "ERROR #"; ERR/2+1 |
| INP(port) | Returns a value (1 byte) from the given port. | 100 P=INP(230) |
| MEM | Returns the number of unused and unprotected bytes in memory. | PRINT MEM<br>1256 |
| PEEK(address) | Returns the value (decimal form) stored at the given address. | PRINT PEEK (15360)<br>65 |
| POINT(x,y) | Returns −1 if the specified graphics block is "on", returns 0 if it is "off". The upper left graphics block has coordinates 0,0, the lower right block has 127,47. | 100 IF POINT (50,60) THEN 500 |
| POS(x) | Returns the current cursor position on the display screen (0−63). Argument is a dummy, use any number. | 100 PRINT TAB (POS(0)+5) "NEXT WORD"; |
| RESET(x,y) | See SET. | |
| SET(x,y) | Turns on the graphics block specified by the coordinates. To turn if off use RESET. | 50 SET(A,B)<br>60 RESET(A,B) |
| USR(x) | Calls a machine language subroutine (written by the user) and passes the argument to it. If not needed, the argument is a dummy. Only one such subroutine is allowed in Level II BASIC. | 45 R = USR(N)<br><br>In disk BASIC use<br>45 R=USR1(N) |
| VARPTR(var) | Returns an address which can be used to locate a variable in memory. | 10 K = VARPTR (A$) |

## SUMMARY OF TRS-80 DISK-EXTENDED BASIC

*Constants*

| | | |
|---|---|---|
| &Hdddd<br>&Oddddd | Hexadecimal and octal constants (base 16 and base 8) can be assigned and used directly. | PRINT &H5200,&O5100<br>POKE&H3C00,42<br>10 FOR I=(&H3C00) TO(&H3FFF) STEP(&H40) |

*Commands*

| Command | Explanation | Example |
|---|---|---|
| (Many new commands are available in the TRSDOS mode; only three are included here: BASIC, BASIC *, and BASIC2. All other commands shown are used in disk BASIC.) | | |
| BASIC | Loads Disk Extended BASIC into memory. Uses most of Level II BASIC routines and adds others. Takes up 5.8K bytes of memory. | BASIC |
| BASIC * | Like BASIC but previously loaded BASIC program is not lost. | BASIC * |
| BASIC2 | Gives you Level II BASIC. | BASIC2 |
| CLOAD | Same as Level II except type CMD"T" first, CMD"R" when finished. No filename is allowed. | CLOAD |
| CLOAD? filename | Same as Level II except type CMD"T" first, CMD"R" when finished. It will only work with programs CSAVEd through Disk BASIC. | CLOAD? |
| CSAVE filename | Same as Level II except type CMD"T" first, CMD"R" when finished. Filename is required. | CSAVE "1" |

| | | |
|---|---|---|
| CMD"D" | Loads and executes the TRSDOS DEBUG program. Does not effect your BASIC program. To re-enter BASIC, type G. | CMD"D" |
| CMD"R" | Starts the clock, enables interrupts. | CMD"R" |
| CMD"S" | Returns to the operating system command mode (TRSDOS). Does not re-initialize. BASIC program is lost unless BASIC * is used to reenter. | CMD"S" |
| CMD"T" | Stops the clock, disables interrupts. Use before any BASIC tape input/output operation. | CMD"T" |
| KILL"filename" | Deletes the file from the disk; filename can include :0, :1, or :2 for the specific disk drive. Final " is optional. | KILL"GRAPH1:1" |
| LOAD"filename",R | Loads a file from the disk, erases the current file in memory, clears all variables, closes all files. R is optional, if used, files are left open and program is run. Used as a statement to chain programs. | LOAD "GRAPH2",R<br>LOAD "ANIMAL"<br>10 LOAD "MARK3",R |
| MERGE"filename" | Like LOAD but current program is not erased. Current lines will be erased if merged file has same line numbers. Merged file must be in ASCII format. | MERGE "SUBR1" |
| RUN"filename",R | Like LOAD...R loads and runs file. R is optional, if used files are left open. Used as a statement. | RUN"FANCY"<br>RUN"FAR1",R<br>10 RUN"STAR" |
| SAVE"filename",A | Saves BASIC programs on disk. A is optional, it requests ASCII format rather than compressed format. | SAVE"ANIMAL"<br>SAVE"SUBR1:1",A |

*Statements*

| Statement | Explanation | Example |
|---|---|---|
| CLOSE n, n, ... | Closes access to a file. If no buffer numbers are mentioned, all open files are closed. See OPEN. | 9998 CLOSE 1,2,3<br><br>9998 CLOSE |
| DEF FNname (vars)=exp | Defines a function which you create and then use like a BASIC function. Name is any valid BASIC variable; it designates the type of value returned. Must have at least one variable argument but it can be a dummy. | 10 DEF FNC#(A)=1/(A*A) |
| DEF USRn = exp | Designates the entry address for a USRn subroutine. | 100 DEF USR3 =&H7D00 |
| ERROR | Same as Level II except it writes out the error message. | 150 ERROR 1 |
| FIELD n, m AS var$, q AS var$,... | Assigns field sizes (in bytes) to file buffer. | 100 FIELD 1, 255 AS A$<br>100 FIELD 1, 16 AS A$, 25 AS B$,... |
| GET n, m | Read a record from disk, random access mode. n is buffer channel number, m is record number. If no record number, the current record will be read. | 100 OPEN"R",1, "DEMO"<br>110 FIELD 1, 127 AS A$, 127 AS B$<br>120 GET 1,50<br>130 GET 1 |
| INPUT#n,var, var,... | Reads values from disk into variables mentioned. n is the buffer channel previously assigned to a sequential file whose format is known. | 130 INPUT#1, A,B,C<br>130 INPUT#1, A$,B$,C$<br>130 INPUT#1, X,X$,Y,Y$ |
| LINE INPUT "prompt";var$ | Like INPUT, but no ? is displayed, and all characters typed in are accepted and stored in one string variable. ENTER terminates the input string. Prompt is optional. | 100 LINE INPUT ":";A$(I)<br>100 LINE INPUT B$ |

| | | |
|---|---|---|
| LINE INPUT#n, var, var,... | Reads a line of data, sequential mode. Ignores usual formats. Reads up to 255 characters into one variable. | 100 LINE INPUT #1, TEMP$ |
| LSET var$ = exp$ | Places a value (converted to a string) in the specified buffer field var$, starting at the left filling in with blanks to the right. For random access files. | 100 LSET A$ = "FERDINAND"<br>110 LSET B$ = MKS$(12345.6) |
| MID$(var$,p,n)= exp$ | Allows you to replace part of a string, var$, starting at the pth character and continuing for n characters, with another string, exp$. If n is omitted the length of exp$ or the remaining part of var$ is used. The length of var$ remains the same. | A$ = "GUTTERSNIPE"<br>MID$(A$,2,9)="*********"<br>PRINT A$<br>G*********E |
| OPEN"m",n, "filename" | Opens a disk file for access (creates the file if necessary). "m" is I for sequential input mode, O for sequential output mode, or R for random I/O mode. File is associated with buffer channel number n (from 1-15). | 100 OPEN"I",1, "NAME"<br>100 OPEN"O",1, "MESSAGE"<br>100 OPEN "R",1, "INFO" |
| PRINT#n,exp; exp;... | Writes values onto a disk file, sequential mode. Analogous to PRINT. n is a channel already OPENed. | 110 PRINT#1, A,B,C<br>110 PRINT#1,X,X$,Y,Y$ |
| PRINT#n,USING var$;exp;exp;... | Similar to PRINT USING; var$ contains a format string. | 110 PRINT#1, USING A$;Y;Z |
| PUT n, m | Writes a record to disk, random access mode. The reverse of GET. | 120 PUT 1,50 |
| RSET var$=exp$ | Places a value (converted to a string) in the specified buffer field var$, ending at the right and filling in with blanks to the left. For random access files. | 100 RSET A$= "FERDINAND"<br>100 RSET B$=MKS$(12345.6) |

*String Functions*

| Function | Explanation | Example/Result Returned |
|---|---|---|
| INSTR(p,var1$, var2$) | Searches through string variable var1$ to see if it contains var2$. Starts searching at pth character. If p is not given, 1 is assumed. Returns the position number where var2$ is found to have started, or a 0 if not found. | 15 X=INSTR (M$, "SAM") |
| *Numeric Functions* | | |
| USRn(x) | Similar to USR except 10 subroutines (0 to 9) are allowed. | 110 DEFUSR1=&H5000<br>...<br>550 HL=USR1(X) |
| *Special Functions* | | |
| CVD(var$) | Converts the value found in a buffer field from string to double-precision number form. Inverse of MKD$. | 10 GET 1<br>20 A#=CVD(A$) |
| CVI(var$) | Converts the value found in a buffer field from string to integer form. Inverse of MKI$. | 10 GET 1<br>20 A%=CVI(A$) |
| CVS(var$) | Converts the value found in a buffer field from string to single-precision number form. Inverse of MKS$. | 10 GET 1<br>20 A!=CVS(A$) |
| EOF(n) | Returns a 0 (false) when the end-of-file record has not yet been read, and −1 (true) when it has. n is the buffer number of an open file. | 500 IF EOF(5) THEN 9998 |
| LOF(n) | Returns the number of the highest numbered record in the file. | 50 FOR I=1 TO LOF(1)<br>... |

| | | |
|---|---|---|
| MKD$(exp#) | Converts a double-precision number into the form of an 8-byte string. Inverse of CVD. | 500 LSET A$= MKD$(A#)<br>510 PUT 1 |
| MKI$(exp$) | Converts an integer into the form of a two-byte string. Inverse of CVI. | 500 LSET A$=MKI$(A%)<br>510 PUT 1 |
| MKS$(exp!) | Converts a single-precision number into the form of a four-byte string. Inverse of CVS. | 500 LSET A$=MKS$(A!)<br>510 PUT 1 |
| TIME$ | Returns a string composed of the date and time currently stored in the real time clock memory area. This is set using TRSDOS commands TIME and DATE, or by using POKE. | PRINT TIME$<br>12/10/79 12:30:45<br><br>10 T$=TIME$ |

# APPENDIX B

## ASCII Codes

Most modern computers use the American Standard Code for Information Interchange (abbreviated ASCII) to represent characters such as letters, numbers and special symbols inside the machine. There are 128 ASCII codes used for this purpose. They are written as the decimal numbers 0 to 127 in BASIC. In machine language programming they are usually represented by the hexadecimal numbers 0 to 7F. Hexadecimal (also called "hex") numbers use the 16 digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The hex number A is the same as 10 in decimal, B is 11, and so on up to F which is 15. To go higher than this several hex digits are used, with powers of 16 understood as multipliers. For example, 7F in hex translates into the decimal number $7*16^1+F*16^0=112+15*1=127$.

The first 31 codes are used for control purposes. The TRS-80 uses thirteen of these as shown in our first table. These control functions can be called upon from BASIC by using the CHR$ (D) function, where D is the decimal ASCII code. For example to obtain double sized characters you use the BASIC statement PRINT CHR$ (23). To move the cursor up you use PRINT CHR$ (27), and so on.

## ASCII Control Codes Used on the TRS-80

| Hex | Decimal | Result |
|---|---|---|
| 8 | 8 | Move Cursor Left & Erase Character |
| D | 13 | Carriage Return |
| E | 14 | Cursor On |
| F | 15 | Cursor Off |
| 17 | 23 | Double Sized Characters |
| 18 | 24 | Move Cursor Left |
| 19 | 25 | Move Cursor Right |
| 1A | 26 | Move Cursor Down |
| 1B | 27 | Move Cursor Up |
| 1C | 28 | Move Cursor to Top Left |
| 1D | 29 | Move Cursor to Start of Line |
| 1E | 30 | Erase to End of Line |
| 1F | 31 | Erase to End of Screen |

The ASCII codes from 32 to 127 are used to represent printer characters. To see what characters the codes produce on a specific printer, you can run the following program. The code 32 means print a space, so nothing will show on paper. Our run was on a printer that had both upper and lower case letters.

```
5 CLEAR 500
10 REM ----- ASCII  (HEX AND DECIMAL ASCII PRINTING CODES) ---
20 DIM H$(127),C$(16)
30 FOR K=0 TO 15: READ C$(K): NEXT K
40 DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
50 FOR D=0 TO 127
60    H1=INT(D/16): H2=D-H1*16
70    H$(D)=C$(H1)+C$(H2)
80 NEXT D
90 LPRINT "              ASCII PRINTING CODES.   H = HEX, D = DECIMAL"
95 LPRINT" "
100 FOR I=1 TO 3:LPRINT "   H    D   CHR$(D)   ";:NEXT I: LPRINT
110 LPRINT "------------------------------------------------------------"
120 G$="   %%  ###    !       "
130 FOR D=32 TO 63
140   FOR I=0 TO 64 STEP 32
150   LPRINT USING G$; H$(D+I), D+I, CHR$(D+I);
160   NEXT I: LPRINT
170 NEXT D
```

```
RUN

            ASCII PRINTING CODES.   H = HEX, D = DECIMAL

  H      D    CHR$(D)       H      D    CHR$(D)       H      D    CHR$(D)
--------------------------------------------------------------------------
  20     32                 40     64     @           60     96     `
  21     33     !           41     65     A           61     97     a
  22     34     "           42     66     B           62     98     b
  23     35     #           43     67     C           63     99     c
  24     36     $           44     68     D           64    100     d
  25     37     %           45     69     E           65    101     e
  26     38     &           46     70     F           66    102     f
  27     39     '           47     71     G           67    103     g
  28     40     (           48     72     H           68    104     h
  29     41     )           49     73     I           69    105     i
  2A     42     *           4A     74     J           6A    106     j
  2B     43     +           4B     75     K           6B    107     k
  2C     44     ,           4C     76     L           6C    108     l
  2D     45     -           4D     77     M           6D    109     m
  2E     46     .           4E     78     N           6E    110     n
  2F     47     /           4F     79     O           6F    111     o
  30     48     0           50     80     P           70    112     p
  31     49     1           51     81     Q           71    113     q
  32     50     2           52     82     R           72    114     r
  33     51     3           53     83     S           73    115     s
  34     52     4           54     84     T           74    116     t
  35     53     5           55     85     U           75    117     u
  36     54     6           56     86     V           76    118     v
  37     55     7           57     87     W           77    119     w
  38     56     8           58     88     X           78    120     x
  39     57     9           59     89     Y           79    121     y
  3A     58     :           5A     90     Z           7A    122     z
  3B     59     ;           5B     91     [           7B    123     {
  3C     60     <           5C     92     \           7C    124     |
  3D     61     =           5D     93     ]           7D    125     }
  3E     62     >           5E     94     ^           7E    126     ~
  3F     63     ?           5F     95     _           7F    127
```

# INDEX


ABS(X), 47, Appendix A
AMORT, 158
amortization table, 158-161
AND, 77
ANSWER, 32
application software, 6
apostrophe (REM), 78
*APRIL1*, 45
array, one-dimensional, 36, 79
array, two-dimensional, 39, 80
ARITH, 67-70
arithmetic expression, 19, 32
*ARROW2*, 141
ASC(X$), 81, Appendix A
ASCDEMO, 82
ASCII codes, 337
ASCII file, 208
assembly language, 243-245

BABYQ, 122
*BABYZAP*, 136
BASIC, 8
BASIC interpreter, 6
BASIC summary, 319
binary search, 89, 92-94
bit, 196
block, 61
BONJOUR, 8
BREAK key, 15, 16
buffer pointer names, 217
BUGPROG, 84
byte, 196

*CARPET1*, 25
*CARPET2*, 25
(*CATERER*), 168
channel/buffer number, 214
(*CHECKBAL*), 188
CHR$(23), 18, 106
CHR$(X), 82, Appendix A
*CIPHER*, 115
cipher, 120
CLEAR key, 18
CLEAR, 83, Appendix A
CLOAD, 10, 17
CLOSE, 219, Appendix A
CLS, 18, Appendix A
code, 120
colon, 76
comma (see PRINT)
command, 17
commands, BASIC, 320
compiler, 242
Computer House, 12
concatenation, 80
condition, 29, 31
conditional branching, 29
constant, 20
control codes, 338
control-C, 15
corrections, 17
COS(X), 50, Appendix A
*CRAPS*, 48

cryptogram, 120
CSAVE, 9, 17
cursor control, 171-174
CVI(X$), 218, Appendix A
CVS(X$), 218, Appendix A

DATA, 26, 70, 105, 110, 114, 119, 166, Appendix A
data base, 163-166, 210
data files, 210
data layout, 165
data structures, 61
*DATARIT2*, 70
*DATARIT3*, 71
*DATARIT4*, 71
*DATARITH*, 70
debugging, 83
decimal codes, ASCII, 338
DEFINT, 182, Appendix A
DICE, 47
DIM, 38, Appendix A
direct mode computing, 83
disk files, 210
disk I/O buffer, 216
disk operating system (DOS), 193, 197
disk-extended BASIC, 193, 331-336
*DOGDAYS*, 42
*DOGSALES*, 40
dot matrix printer, 198
DRILL, 34
dummy variable, 72
dummy READ, 105

E notation, 22
(*EDIT7000*), 249
EDIT1000, 234
*EDIT2000*, 241
*EDIT5000*, 248
editor, 230
(*ELBLASTO*), 141
END, 69, Appendix A
(*ENERGY*), 28
ENTER key, 9, 17
ESTIMATE, 164
execute, 15
extended BASIC, 76
extended IF, 77
external storage, 196

FAMOUS, 43
FANCY, 53
FIELD, 217
fields, printing, 20
FILEBOX, 210
*FILEBOX2*, 222
*FILEBOX3*, 224
FILEDEMO, 219
files, 210
*FINAN2*, 162
*FINANCE*, 161
firmware, 6
flag, 106
floppy disk, 11, 210-229
flow charts, 30, 31
FOR, 34, Appendix A
functions, BASIC, 47, 327-331

GET, 217
GOTO, 15, Appendix A
*GOTCHA*, 17
GOWRONG1, 35
*GOWRONG2*, 35
*GOWRONG3*, 35
graphics, alphanumeric, 50, 123
graphics, point, 137
GROCERY1, 23
*GROCERY2*, 28
GUESS1, 87
*GUESS2*, 92
*GUTENTAG*, 17

hard copy printer, 197
*HARDSALE*, 175
hash coding, 224-228
HASHDEMO, 226
(*HASHFILE*), 227
HEIGHT, 24
hex codes, ASCII, 338
hexadecimal numbers, 243, 245
HYACINTH, 13

*ICECREAM*, 38
IF ... THEN, 29, Appendix A
IF ... THEN ... ELSE, 77, Appendix A
INPUT, 23-25, Appendix A
inputting strings, 46
INT(X), 47, 72, Appendix A
interface, 197
interpreter, 61, 242

keyboard, 14

LEFT$(A$,M), 80, Appendix A
LEN(X$), 81, Appendix A
LET, 21, 32, Appendix A
letter-quality printer, 198
LINE INPUT, 197, 241, Appendix A
line numbers, 17
linked lists, 230
LIST, 13, 17, Appendix A
LOAD (CLOAD), 17
LOAN1, 149
LOAN2, 155
LOF(X), 220, Appendix A
LOG(X), 95, Appendix A
logical record 221-225
loop, 35
LPRINT, 175, Appendix A
LRDEMO, 81
LSET, 218, Appendix A

machine code, 242, 245
machine language, 242, 245
macro flow diagram, 180, 181
magnetic disk, 11
mass storage, 196
master file, 164
matrices, 80
memory, 79, 195
menu, 161, 176
MERGE, 208
MID$(A$,P,N), 81, Appendix A
minidisk, 11
MKI$(X), 218, Appendix A
MKS$(X), 218, Appendix A
MLDEMO, 243
module, 1, 180
MONEY, 176
*MONEY2*, 188
(*MONEYDISK*), 228
MONYLIST, 199
*MULT*, 73
multiple statements on one line, 76

nested FOR loops, 35
NEW, 12, 17, Appendix A
NEXT, 34, Appendix A
NOT, 77
n-tuple, 163

*OKPROG*, 85
ON K GOSUB, 53, 74, Appendix A
ON K GOTO, 52, 73, Appendix A
one-dimensional array, 36
OPEN, 216, Appendix A
operators, BASIC, 320
OR, 77

parallel interface, 198
parentheses, 21
PEEK(X), 207, Appendix A
peripheral, 193, 246
physical record, 211, 221-225
plan A, vi
plan B, vi

*POEM*, 44
POKE, 243, Appendix A
PRINT, 12, 17, 19, Appendix A
PRINT @, 106, 110, 113,
  Appendix A
PRINT USING, 82, Appendix A
printer, 197
PRINTPI, 19
program, 4, 6, 8, 17
program stub, 180
PRUDEMO, 83
pseudo code, 132
PUT, 217, Appendix A
Pythagorean theorem, 133

Radio Shack TRS-80, Model I, 7, 197
Radio Shack TRS-80, Model II, 247
RAM, 195
RANDOM (RANDOMIZE), 73,
  Appendix A
random numbers, 47, 73
READ, 26-28, 70, 71, 105, 110, 114,
  119, Appendix A
READDATA, 26
READY, 10
record, 210
REM (remark statement), 69,
  Appendix A
report generating, 176
RESET(X,Y), 137, Appendix A
RESTORE, 70, 71, Appendix A
*RETIRE*, 31
RETURN key, 9
*RICHQUIK*, 22
RIGHT$(A$,N), 80, Appendix A
RND(0), 47, 72, 73, 111, 112,
  Appendix A
*RNDARITH*, 72
ROM, 195
RS-232-C standard, 198-199
RUN, 9, 17, Appendix A

SALESLIP, 169
SALEDATA, 37
*SAM*, 44
SAVE, 151, Appendix A
SAVE (CSAVE), 17, Appendix A
SCRAM1, 96
*SCRAM2*, 110
*SCRAM3*, 113
scroll up, 33
sector, 215
semicolon (see PRINT)
sequential search, 93
serial interface, 198
SET(X,Y), 137, Appendix A
SETPLOT, 137
SIN(X), 50, 134, Appendix A
SNAPSHOT, 207
software, 4
solutions to projects, 253-316
source program, 242
*SQCUBE*, 34
*SQROOT*, 75
*SRCH1*, 93
*SRCH2*, 94
*SRCH3*, 94
STARS, 52
statement, 8, 17
stepwise refinement, 132
STOP, 29, Appendix A
string, 80
string array, 79
string variable, 43
structured program design, 31, 59, 63
structured programming, 31, 63
stub, 180
subscripted variables 36
SURVEY, 21
system software, 6

table, 36
table, one-dimensional, 79
TAB(X), 9, 17, 50, Appendix A
*TABGRAPH*, 50
*TAXRPT*, 206
text editor, 230
*TEXTFORM*, 249
text formatter, 230
*THROW A PARTY*, 54
tolerance, 74
transaction data, 164, 176
TRIGPIX, 139
two-dimensional arrays, 39-42
two-dimensional table, 80

VAL(X$), 81, Appendix A
VALDEMO, 81
variable, 20, 36
variable name, 20
variable name, long, 79
*VARIETY1*, 74
*VARIETY2*, 74
vector, 36, 79
video output, freeze, 33
VOTER, 29

WHILE, 90
word processing, 230

**A treasure chest of applications for your TRS-80! Plus ideas for creating your own programs!**

## About the Authors

**Thomas A. Dwyer** is Professor of Computer Sciences at the University of Pittsburgh, Pittsburgh, Pennsylvania. He earned his Ph.D. degree at Case Institute of Technology, and has taught at both the university and high school levels, with interests in microcomputers and the innovative application of computers to education. He was Director of Project Solo and Soloworks, experiments in computing for secondary school students. He is the author of numerous articles on personal computers as well as the books, *A Guided Tour of Computer Programming* (1973) and (with Ms. Critchfield) *A Computer Resource Book, Algebra* (1975) and *BASIC and the Personal Computer* (1978).

**Margot Critchfield** is a doctoral student in Foundations in Education at the University of Pittsburgh. Ms. Critchfield studied art at the Cleveland Institute of Art and anthropology and education at the University of Pittsburgh, where she earned her B.A. and M.Ed. degrees. Ms. Critchfield served as Senior Research Associate for Project Solo and the Soloworks Lab, and has authored several articles on computers in education, computer graphics, and computer art. She is presently Senior Research Associate for a new project in the educational use of microcomputer networks.

Radio Shack A Division of Tandy Corporation