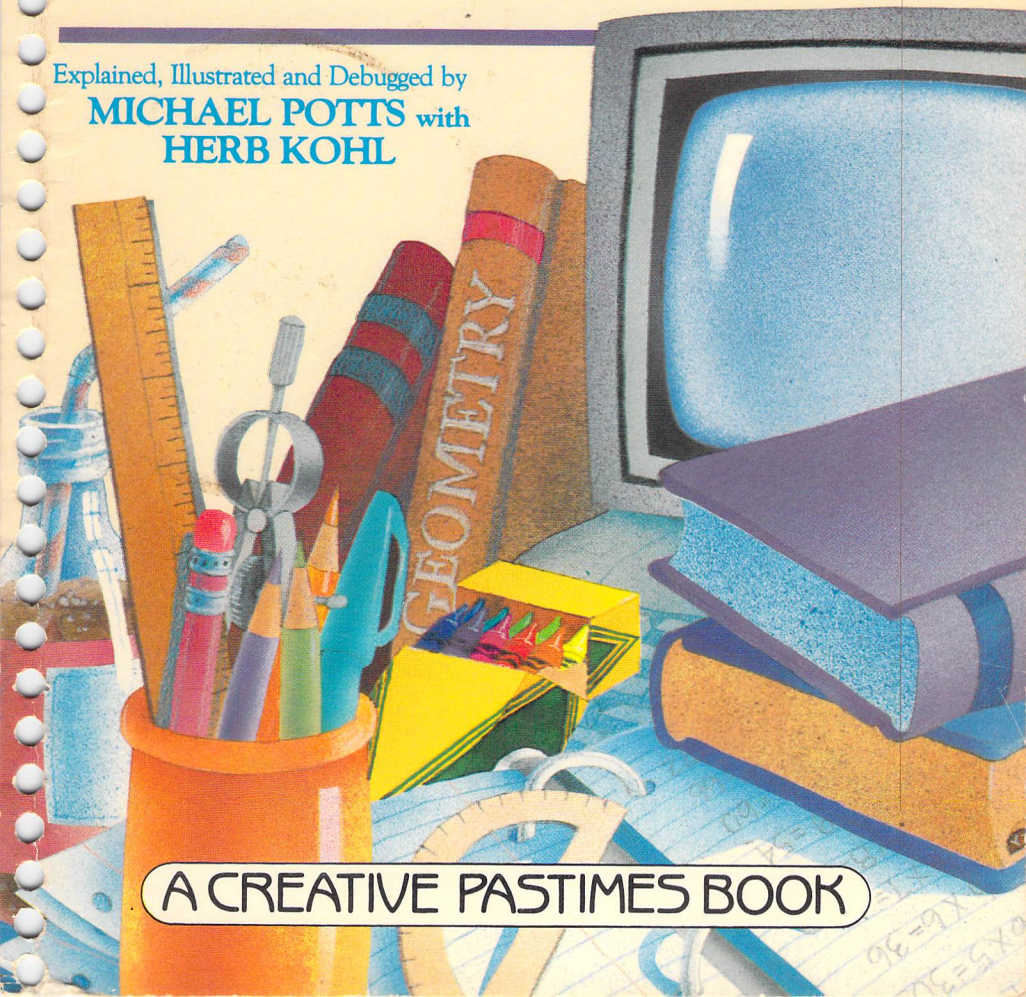
# Homework Helper

## USEFUL PROGRAMS FOR COMMODORE® COMPUTERS

Explained, Illustrated and Debugged by
MICHAEL POTTS with
HERB KOHL




A CREATIVE PASTIMES BOOK

This book belongs to

_____

# Homeworker Helper for the Commodore 64®

useful programs
for the Commodore 64
explained, illustrated,
and debugged by

**MICHAEL POTTS**

with

Herbert Kohl

A Creative Pastimes Book
Reston Publishing Company, Inc.
Reston, Virginia
A Prentice-Hall Company

ISBN: 0-8359-2861-6

Interior design and production by Karen Winget.



10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

# contents

# acknowledgments

Thanks to Herb Kohl for his belief that this book could be done, and to Rochelle Elkan, without whose patient and thoughtful collaboration this book could not have been done.

# preface

The four programs in the first section of the book are meant for younger students—they introduce the computer's keyboard and screen and help build symbol recognition and logic skills. But we can't expect young ones to program before they can read. An older person—"the programmer," maybe seven or older—will need to help. The first programs are carefully designed to help the novice over some of the humps all programmers encounter when they start, and to proceed with good programming habits from the very first.

The six programs in the second section are for more advanced computer students; even so, programmers younger than age twelve will be able to write these programs, which will then be of genuine use in their studies. Some of these programs are more advanced in structure and concept, so as you build these programs, you gain more insight into the ways professional programmers develop working programs.

The last section contains six programs that will be of use to anyone who wishes to use the computer as a tool for organizing and simplifying daily life. And the writing of these programs will expose the programmer to some of the more advanced techniques available to a programmer working on the Commodore 64.

In this book you learn to use the computer while building useful tools—the computer is present in your learning as *subject,* as *means*, and as *tool.* I have been working with microcomputers in classrooms, using that approach, since they first appeared there in the late 1970s. I have watched seven-year-olds use this approach to learn programming. For many students, the computer is as natural a part of the learning environment as the television and the textbook.

"Yes, but does it do anything *useful*?" is a question many home computer owners hear. With these programs on your computer, the answer is clearly Yes. There are programs in this book that anyone, even big people, can use for practical work. And again: learning the native language of a computer, in order to teach it to do your work, is the most useful learning of all.

I have written this book with a clear purpose always in mind: learning *and* computers are fun when embarked upon in the right spirit.

Michael Potts

# a message to parents and teachers

With this book, you can turn a Commodore into a Homework Helper. The programs in the *first section* help pre-school and elementary students (age four to nine) practice elementary concepts—counting, symbols, and spelling. Very young children can play these games, and learn with the computer. (The bigger person who helps type these programs in will learn a lot about programming in BASIC—more about that later.) The intelligence games in the *second section* are for the use of older students—ages seven through thirteen—in polishing these skills further, but there are learning challenges here for most adults as well. The study tools in the *advanced section* help you illustrate, organize, graph, and calculate.

This is not a read-only book: you should work with it and a Commodore computer. It doesn't matter if you've never touched a computer—or even a typewriter keyboard—before. On the way through this book, you learn to program in BASIC. When you are done with this book, you will have a well-trained silicon-based homework assistant, *and* you will know how to use BASIC to get computers to do what you want.

Most books that teach programming remind me of high school language textbooks: lots of word lists and verb conjugations, but no feel for the language or the people that speak it. In this book I try to interweave computer language with a plain-English explanation. Doing this book is the equivalent of living for two weeks with a family in the BASIC interpreter!

Many computers live in closets because owners find little useful to do with them. This book tackles that problem head on: people, students particularly, can use the programs in this book to go the extra distance needed to be an exceptional student.

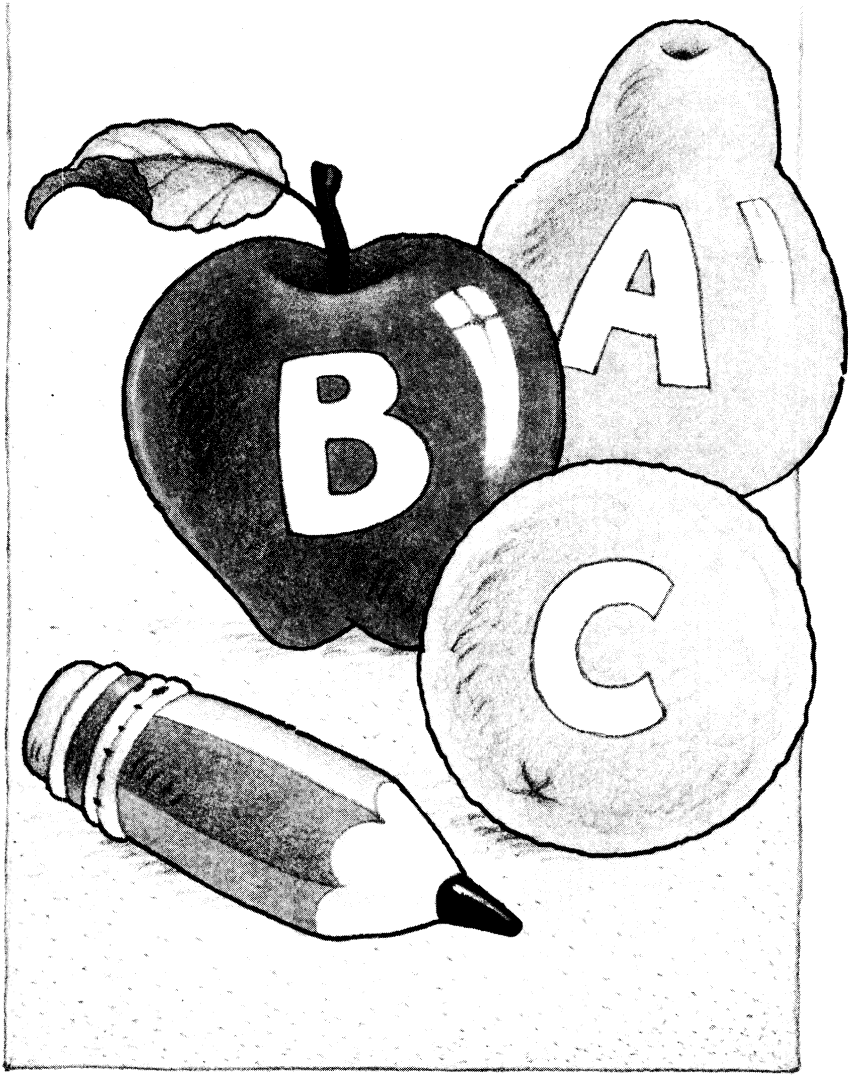
If you just bought a Commodore computer or are thinking about buying one to help your children with school, this book will build a useful partnership between your children and the computer.

If you bought (or are considering) a Commodore to teach yourself programming, this book will help you build the tools you will need to turn the computer into a working partner.

If you are a teacher with a Commodore in your classroom, this book should reside beside the computer as an invitation to your students to make the computer do useful work. Take the computer and book home on weekends and write the programs; when you are done, you will be able to answer most of your students' (answerable) questions, and you will know enough to turn the computer into your classroom partner.

If you work with words and numbers, this book will help you turn your Commodore computer into a willing, skilled, tireless, accurate, and speedy assistant. Machines should work, and people should think—but to make machines work, you need to know how to ask. BASIC, the native language of most small computers, is easy to learn: even my youngest students have mastered it, working on tools like the ones in this book—tools to help with their work. This book makes computers a relevant and useful study.

# early childhood games



Learning with computers takes a number of forms; learning *to program* a computer is an experience much like learning a foreign language. To me, programming is negotiation with a Silicon Intelligence—the single-minded, literal, and logical presence within my machine. If I tell it to tie itself in knots or chase its tail, it does it unquestioningly and with lightning speed. Over the years, I have learned that even the simplest program benefits from my logical approach—or suffers when I am not sure what I want to do. Beginning programmers, especially younger ones, often bite off projects much larger than they can chew, and find programming quite frustrating. I am reminded of George Washington Carver's story of his dialogue with the Almighty: "I want to know Everything about Everything," George said, and the answer came back, "How about *everything* about the *peanut,* George. It's more your size."

The programs in this section are easy to write, with just enough challenge for beginning programmers. They're about the right size.

# NUMS

a symbol recognition game

## DISCOVERING THE COMPUTER

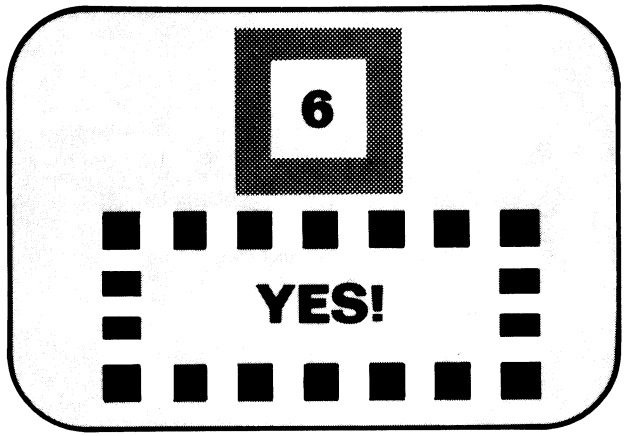
NUMS is a very simple program, and even very simple programs have their challenge. Computers can be demanding partners. They are extremely literal-minded. Therefore, a good strategy for getting the job done is needed.

For this program to be useful as more than a programming project, you will need a young person—aged 2 to 5—to play the game when you have finished it.

The purpose of the game is to help build a quick, accurate association between the number keys on the computer and the symbols on the video screen. Many younger children find the keyboard, with all its symbols and keys, quite bewildering. Making friends with ten keys helps engage these young ones.

For the programmer, the purpose is to work with the computer with a job in mind and finish with a tool usable by a young child. The skills learned here will be useful when it comes to more complicated (and interesting) challenges in the future.

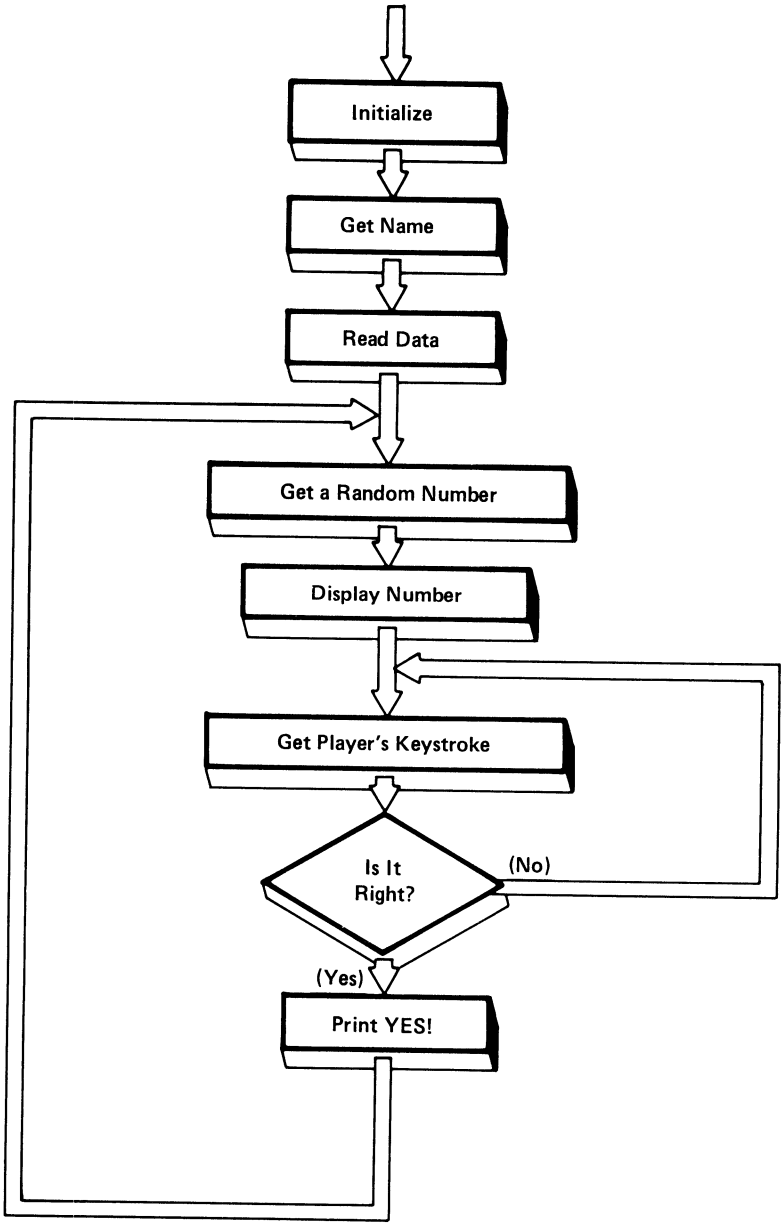
In NUMS, the computer randomly chooses a number (from an easily changed group defined within the program), and the player presses the corresponding key. (When you get it, a colorful YES flashes on the screen.




Younger children (aged 2-5) may be able to recognize only the numerals from one to five at first but will quickly learn the rest playing this game.

## WRITING THE PROGRAM

Here's a flow chart and program listing for NUMS. If they make no sense to you, don't worry. There's a step-by-step explanation on page 8. Programming in BASIC, the native language of your Commodore, is easy, because you can build programs a few lines at a time and test them to make sure they work. We will use that system—professionals call it "modular programming." It isn't entirely fair to make you look at the whole program at once, but here it is:


Initialize
Get Name
Read Data
Get a Random Number
Display Number
Get Player's Keystroke
Is It Right?
(No)
(Yes)
Print YES!

```
1 REM                                   * NUMS *
2 DIM A$(100):GOSUB 900
5 DATA "1","2","3","4","5","6","7","8","
9","0","END"
6 K$="                    ":M$="
  "
10 PRINT:PRINT "HELLO.":PRINT "MY NAME I
S NUMS."
20 INPUT "WHAT IS YOUR NAME";N$:N=0
30 READ A$(N):IF A$(N)="END" THEN 50
40 N=N+1:GOTO 30
50 LN=0:HN=N-1
60 GOSUB 1300:REM GETS A RANDOM NUMBER
70 GOSUB 900:REM  CLEARS THE SCREEN
80 PRINT "[RV][RED]          "
81 PRINT K$;"[RV] [RV]      [RV] "
82 PRINT K$;"[RV] [RV]      [RV][RED] "
83 PRINT K$;"[RV] [RV]      [RV] "
84 PRINT K$;"[RV]          "
90 Z=ASC(A$(R)):POKE 1523,Z
100 GET Q$:IF Q$="" THEN 100
110 IF Q$=A$(R) THEN 140
120 GOTO 100
130 PRINT
140 PRINT
150 PRINT M$;"[RV][RED] [GRN] [RED] [GRN
] [RED] [GRN] [RED] [GRN] [RED] [GRN] [R
ED] [GRN] [RED] "
160 PRINT M$;"[RV][GRN] [RV]            [
RV] "
170 PRINT M$;"[RV][RED] [RV]  Y E S !  [
RV][RED] "
180 PRINT M$;"[RV][GRN] [RV]            [
RV] "
```

```
190 PRINT M$;"[RV][RED] [GRN] [RED] [GRN
] [RED] [GRN] [RED] [GRN] [RED] [GRN] [R
ED] [GRN] [RED] "
200 FOR T=1 TO 1000:NEXT
290 GOTO 60
900 PRINT CHR$(147);N$
910 FOR N=1 TO 9:PRINT:NEXT
920 PRINT K$;
930 RETURN
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

Subroutines can be used from anywhere in a program or borrowed for several programs. In later programs you'll begin to recognize "old friends" among the program lines. You will learn to borrow them from earlier programs, so you won't have to type them in more than once. That's what computers are for: machines should work; people should think.

So let's get down to it, and start at the end!

The last line of the program is a one-line subroutine that makes up a random number R between a Low Number LN and a High Number HN. You will need this line in several programs, but you need to type it once. If you build the program a piece at a time, and test each piece as you build it, you know for sure that your program works. If you type a program like this:

```
100 LN=0:HN=9
110 GOSUB 1300:PRINT R,
120 GOTO 110
1300 R=INT(LN+(HN-LN+1)):RETURN
```

and RUN it, your video screen will fill with randomly chosen numbers between 0 and 9—unless you made a mistake. If it doesn't work, you only have four lines of code to debug. Lines 100, 110, and 120 are temporary (called a "stub" by programmers): they test the random number subroutine at line 1300 to assure that it is working right.

A subroutine to clear the screen is a logical second step. If you now type in

```
1 K$="                "[16 spaces]
110 GOSUB 900:GOSUB 1300:PRINT R
900 PRINT CHR$(147);N$
910 FOR N=1 TO 9:PRINT:NEXT
920 PRINT K$;
930 RETURN
```

and RUN your program, the computer will flash a series of single random numbers near the center of the computer's memory. When you type the new version of line 110, it replaced the earlier version in the computer's memory.

### *Commodore 64 notes*

Saving Your Work

It would be a good idea to practice saving your program while it's short. If you (or the computer) should happen to make a mistake, you'd only have to type 8 lines over again. If you have a tape system, make sure your recorder is ready, and type

```
SAVE"RANCLR"
```

If you have a disk system, type

```
SAVE"RANCLR",8
```

When the computer replies "OK", check the light on the front of the disk subsystem. If it isn't flashing, you know your work is protected against loss. You could turn the machine off now.

To get the program back again, you type LOAD instead of SAVE.

On the Commodore, you can use a filename only once for saving with a disk system. If you try to save a file with a name that is already on the disk, the disc drive's red light will flash, indicating an error. It's good practice to save files with consecutive names, like NUM1, then NUM2, then NUM3. Keep track of your file names.

## THE BEGINNING OF NUMS

Now that the end of the program is written, and you know the two subroutines do what they are told, you can write the beginning lines.

Line 2 gets the computer ready to remember 101 symbols (don't forget 0)—more than needed but when you have 64K of memory, you'd just as well use it! Line 5 contains the symbols to be flashed

on the screen; the symbols should be in quotes, and the last datum has to be "END." Of course, you can change this line to contain any symbols you want to play with.

Lines 10 and 20 find out the player's name. Lines 30 and 40 are a "loop" that reads the data in line 5 and stores it in an array called A$—I say it "A-string"—testing every one as it goes to see if it's the END. Line 50 sets the low number and high number for the random number making subroutine. It is the last line in the set-up part of the program.

## THE MAIN PROGRAM

The main program starts at line 60: it sends for a random number; line 70 clears the screen and prints the player's name in the upper left-hand corner. Skip the fancywork in lines 80-84 for now, unless you know how to make those funny shapes on the screen. We'll come back to them when everything else is working right.

Line 90 has two tricky things in it. The first one, ASC(A$(R)), looks up the Rth symbol in the list—which is called an "array"—and figures out the number the computer uses—called the ASCII code—to represent it. Next, the line has a POKE in it. A POKE finds a particular place in the computer's memory and pokes a number into it. Memory location 1523 is right in the center of the video-screen memory, and Z is the ASCII code for our random symbol. If you type

```
POKE 1523,65
```

from your keyboard (no line number, please) a capital A will miraculously appear in the center of the video screen. But that's enough magic for now.

In line 100, the computer awaits a character, any character, from the keyboard. If it doesn't get one—if Q$="", which is called "the null string" because there isn't anything between the quotes—it waits some more. IF it's the right key, line 110 sends it on, but if it isn't line 120 sends it back to wait until the right key is pressed.

Skip lines 150-190 for now, but type in a temporary line 170 like this:

```
170 PRINT "Y E S !"
```

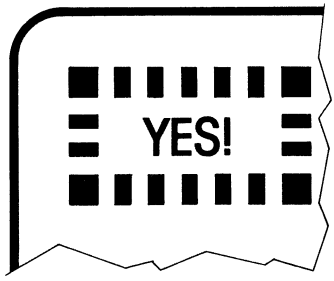
Line 200 is an "empty loop"—it doesn't do anything but count to 1000, so the display stays long enough for you to read it. Line 290 sends the computer back to the beginning of the main program loop at line 60.

Try it out: type RUN. If it doesn't work, find out why. This is the fun part: debugging.

## SOME FANCYWORK

When you have exterminated any bugs, you have time to dress up the program for use. If you know how to edit—look in the manual, because you need to know soon—add a 13-space-long M$ to line 1.

Now for the graphics. In the program listing, color and video control codes look like blocks with symbols in them. You get those blocks into your program by holding down the control key (or the Commodore key) and pressing a color key in the top row. Holding down <CTRL> then pressing the <9> deposits a reversed-video-R, which turns on reverse video, so spaces appear on your screen as blocks of color. Holding <CTRL> and <3> prints a reversed-British-pound symbol, which changes the color to red. Line 80 consists of those two characters and 7 spaces—just press the space bar. <CTRL><0> prints as a reversed underline, and turns the reverse video off on the screen; in lines 81 and 83 it makes a blue space in the middle of a box around the center of the screen, to set off the displayed symbol. Line 82 adds a <Commodore key><7> to make the symbol light blue—the best color for display clarity. Line 84 uses this character, too, so that anything printed by the computer is readable. We did all this to make a red box appear around the symbol we want the player to find. One consolation: once it is typed into a program, you don't have to type it again.



That is, if you saved it! What if the power went off now? This program is not quite finished, but good program practice says: SAVE IT anyway. If you have a tape system, type

```
SAVE "NUM1"
```

and follow the directions. If you have a disk system, type

```
SAVE "NUM1",8
```

Now if the power goes out, you're covered.

The last fancywork is a colorful border around the "Y E S!". The only new character is a <CTRL><8>, which makes alternate blocks yellow. Of course, if you have other favorite colors than red and yellow, you can use them.

An editing trick: line 190 is almost the same as line 150. You can LIST 150, then move the cursor up onto the listed line, change the 5 in the line number to a 9, move your cursor to the right, and insert a <Commodore key><7>, then press return, and you've got the new line at almost no cost to yourself. Of course, if that seems too complex, you can type the lines from scratch.

Now you get to debug the new parts—I had to fiddle with my colors to make them work the first time. Then BE SURE TO SAVE! It's a finished program now, so call it NUMS when you save it. Test your work, and when you're sure it's right, find someone to play it.

When players get tired of the number from zero to nine, you can change line 5's data. Here are examples.

The most-used keys:

```
5 DATA "E","T","O","A","I","N","R","S","END"
```

The home row for touch typing.

```
5 DATA "A","S","D","F","J","K","L",";","END"
```

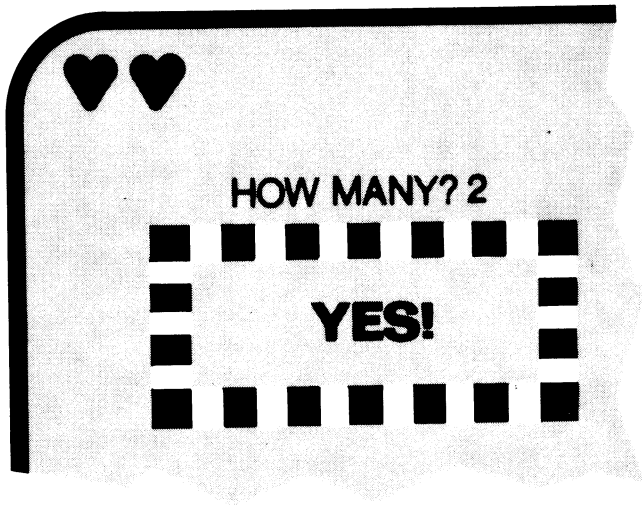
The beginning of the alphabet.

```
3 DATA "A","B","C","D","E","F","G","H","I"
4 DATA "J","K","L","M","N","O","P","Q","R"
5 DATA "S","T","U","V","W","X","Y","Z","END"
```

You can use any keyboard symbols you want. Just make sure that END is the last entry in your DATA.

# COUNTEM

a counting game

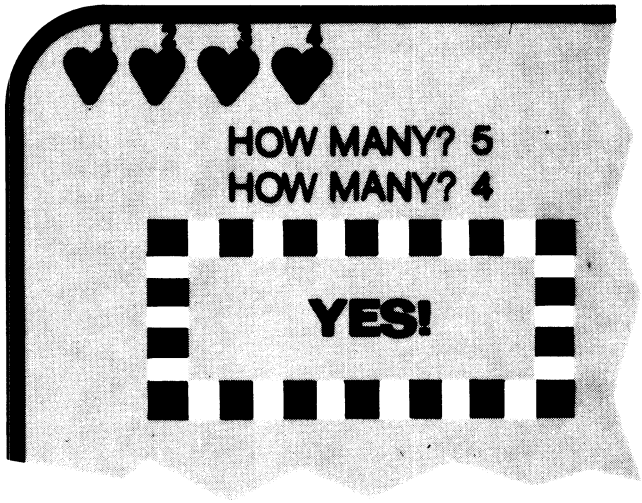



## A COUNTING-SKILLS PROGRAM

This program was inspired by the Count, a TV character that delights in counting things—a delight shared by many young children. As soon as young players have found some of the keys they need on the computer, they are ready to strengthen associations between the keys and numbers.

For the programmer, COUNTEM is interesting because it introduces the concept of "borrowing from yourself," a strategy that saves a lot of work and confusion: if you can reuse a piece of a program (a module) that is known to work, the programming task is much easier.

What happens when the player gets a wrong answer? Computer programs that TEST ONLY say "WRONGO!" (and some even make an impolite noise!). I like programs that help the player LEARN. The computer, and the programmer, should be smart enough to offer help. Frequently, the help is nothing more than a quick reminder about how to count:




COUNTEM draws a random number of hearts on the screen, then invites the child to count them. (The programmer chooses the range of numbers to count.)

A simple switch, and the objects to be counted flash on the screen for only a moment; even adult players can challenge and build their pattern-recognition skills!

```
1 REM                                   * COUNTEM *
2 C$=CHR$(147)+D$:M$="                                  "
3 FOR N=1 TO 4:D$=D$+CHR$(13):NEXT
5 C$=CHR$(147)+D$:M$="                                  "
8 PRINT C$;M$;"* COUNT 'EM *";D$
10 INPUT "WHAT IS THE LOWEST NUMBER";LN
20 INPUT "        THE HIGHEST NUMBER";HN
30 GOSUB 1300:PRINT C$;:FOR N=1 TO R
40 PRINT "  S";
50 NEXT:PRINT D$
60 PRINT M$;"HOW MANY? ";:GOSUB 100
70 IF Q=R THEN 140
80 PRINT "[CD][CD][CD][CD] ";:FOR N=1 TO
 R:PRINT N;:NEXT
90 PRINT D$:GOTO 60
100 GET Q$:IF Q$="" THEN 100
110 IF Q$<"1" OR Q$>"9" THEN 100
120 Q=VAL(Q$):PRINT Q:RETURN
140 PRINT
150 PRINT M$;"[RU][RED] [GRN] [RED] [GRN
] [RED] [GRN] [RED] [GRN] [RED] [GRN] [R
ED] [GRN] [RED] "
160 PRINT M$;"[RU][GRN] [RU]            [
RU] "
170 PRINT M$;"[RU][RED] [RU]  Y E S !  [
RU][RED] "
180 PRINT M$;"[RU][GRN] [RU]            [
RU] "
190 PRINT M$;"[RU][RED] [GRN] [RED] [GRN
] [RED] [GRN] [RED] [GRN] [RED] [GRN] [R
ED] [GRN] [RED] "
200 FOR T=1 TO 1000:NEXT
290 GOTO 30
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

## WRITING THE PROGRAM

This program uses some routines borrowed from NUMS. Wouldn't it be nice if you didn't have to type them in again? Well, you don't!

*Commodore 64 notes*

Start out with NUMS in your computer's memory, and type

```
LIST 140-200
```

The computer will list the code for the colorful YES! block. Then type

```
LIST 1300
```

You'll see the random number generator on your screen. Now type

```
NEW
```

and NUMS will be erased from the computer's memory, but *not from the screen!* To get the lines into your new program, use the cursor control keys to move the cursor into each line you want to borrow, and press <ENTER>.

For COUNTEM you need to borrow lines 140, 150, 160, 170, 180, 190, 200, and 1300. Now LIST your program. The trickiest parts are already done.

## THE PROGRAM'S INTRODUCTION

Lines one and two set up a "Clear the Screen" string, called C$, to erase old data and place the cursor near the screen's center. D$ sends the cursor down four lines at one fell swoop. M$ is a margin, so that the writing isn't crowded against the left side of your screen. Line 3 uses both of these *screen formatting strings* to display the program's name.

Lines 10 and 20 get the Lowest and Highest Numbers for the random number maker. When you play this program, remember to keep the numbers adjusted to the player. If you use numbers less than 1 or bigger than 9, the program will crash.

## THE MAIN PROGRAM

Line 30 sends out to our old friend the Random Number Maker for a random number R, then prints R hearts on the screen. You make the heart in line 40 by holding the shift key and pressing (S).

Lines 50 and 60 move the cursor down a bit and ask "HOW MANY?," then send the program to the keyboard subroutine at line 100 for an answer.

The keyboard subroutine that starts at line 100 "looks" at the keyboard with the GET Q$. If nothing is there—no key has been pressed—then Q$ is the null string (Q$="") and the program goes back and looks at the keyboard again. And again and again . . . .

Until someone presses a key. Then line 110 checks to see if it's a legal key—one of the numbers between 1 and 9. If it's not legal the computer ignores it and goes back to waiting for someone to press a key.

If it is a legal keystroke, the keyboard subroutine figures out the number value for the key, prints the symbol, and sends control back to line 60, the line that called the subroutine.

Line 70 checks to see if Q (the value of the key pressed) is the same as the number of hearts. If it is, off to the colorful YES! box at line 140.

If not, line 80 moves the cursor back to beneath the hearts, and numbers them to help the player count. Line 90 moves the cursor back to the HOW MANY? then sends the computer back to line 60 for a new answer.

You need only one more line: line 290 sends the computer back to the beginning of the main program after the YES! display.

Time to test the program, then put it to use strengthening number concepts.

A challenge for people who know how to count: set the low number at 5 and the high number at 9, then try to *estimate*—counting is *no fair*—the number of hearts. After a little practice, you'll be amazed at how quick you get.

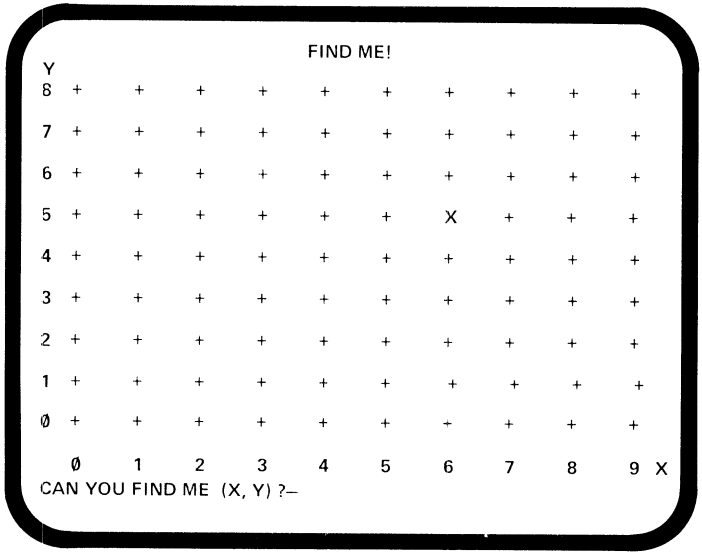
To speed things up a bit, change line 50 by adding a timing loop like this:

```
50 NEXT:FOR T=1 TO 50*R:NEXT:PRINT C$;D$;D$
```

If that goes by too fast, change the 50 to a larger number.

# FINDME

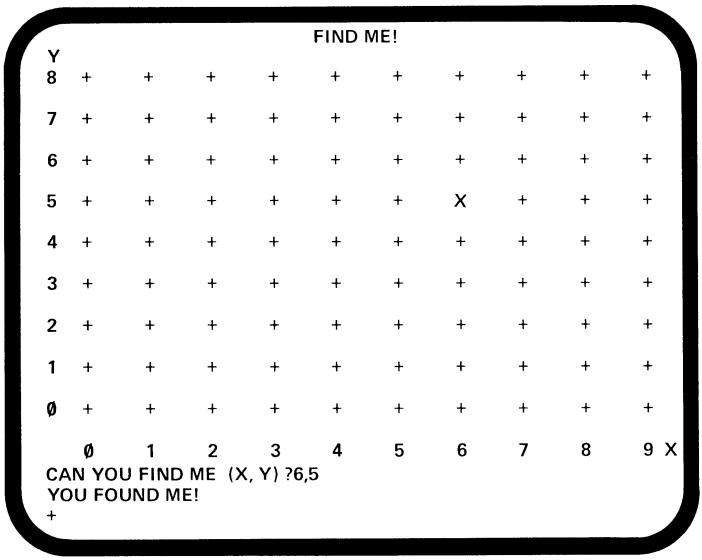
coordinate geometry game

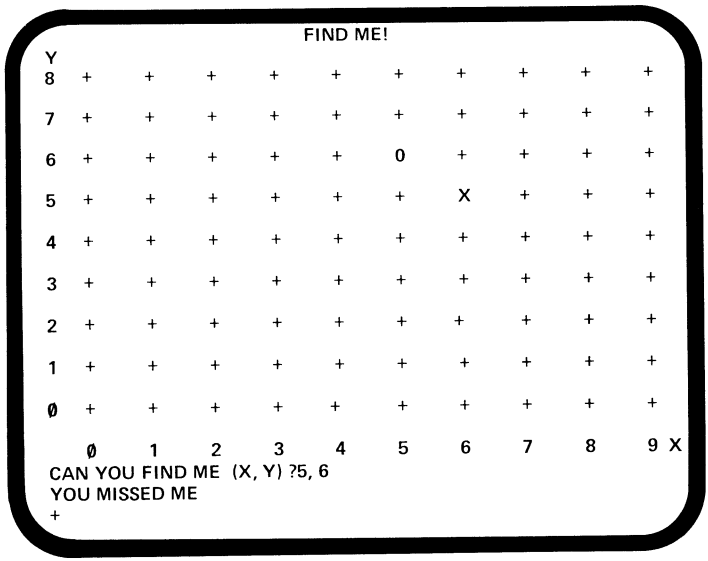



## HIDE-AND-SEEK WITH A COMPUTER

Now that numbers are second nature, how about something more difficult? Two numbers! Coordinates are two numbers that define a location in a two-dimensional space, and this game plays with that idea. Ordered pairs—you always type the horizontal, or X-coordinate, first—are an important mathematical notion that many children never wholly grasp. This game is good practice for several classic games, including Battleship and Star Trek.

The main program for FINDME is very simple to write and debug. For more challenge, there is a second version that employs the computer's sound capabilities and demands a bit more of the programmer's—and player's—skill.


FIND ME!
Y
8 + + + + + + + + + +
7 + + + + + + + + + +
6 + + + + + + + + + +
5 + + + + + + X + + +
4 + + + + + + + + + +
3 + + + + + + + + + +
2 + + + + + + + + + +
1 + + + + + + + + + +
Ø + + + + + + + + + +
Ø 1 2 3 4 5 6 7 8 9 X
CAN YOU FIND ME (X, Y) ?6,5
YOU FOUND ME!
+


FIND ME!
Y
8 + + + + + + + + + +
7 + + + + + + + + + +
6 + + + + + 0 + + + +
5 + + + + + + X + + +
4 + + + + + + + + + +
3 + + + + + + + + + +
2 + + + + + + + + + +
1 + + + + + + + + + +
Ø + + + + + + + + + +
Ø 1 2 3 4 5 6 7 8 9 X
CAN YOU FIND ME (X, Y) ?5, 6
YOU MISSED ME
+

FINDME makes up coordinates—pairs of numbers, like 2,3—and plots the corresponding point with an X on the screen. The player finds them.

A tricky variant—Sound FINDME—doesn't mark the point, but gives you musical clues to help you find it: the lower the note, the lower the number.

## WRITING THE PROGRAM

With the exception of the business at lines 1200-1210, this is an easy program.

```
1 REM                                   * FINDME *
2 C$=CHR$(147):VB=1024:LN=0
10 PRINT C$;"      FIND ME!":PRINT " Y"
20 FOR R=8 TO 0 STEP -1:PRINT R;
30 FOR C=0 TO 9:PRINT "+  ";:NEXT C:PRIN
T:PRINT
40 NEXT R:PRINT "  ";:FOR C=0 TO 9:PRINT
 C;:NEXT:PRINT "X"
50 HN=9:GOSUB 1300:X=R:HN=8:GOSUB 1300:Y
=R
60 VX=X:VY=Y:Z=86:GOSUB 1200
70 PRINT:PRINT "CAN YOU FIND ME (X,Y)";
80 INPUT QX,QY:IF QX=X AND QY=Y THEN 100
90 GOTO 200
```

Skip lines 100-290 for now, and deal with this mystery:

```
1200 VP=VB+40*(16-2*(VY-1))+3*(VX+1)
1210 POKE VP,Z
1290 RETURN
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

You recognize our old friend at line 1300—the Random Number Maker. It has to work twice as hard in this game, because every answer needs *two* random numbers.

But what is all that other nonsense? Read on.

Memory-mapped Video

Like many home computers, the Commodore 64 reserves an area of memory for the symbols on the video display. The Commodore display is 40 characters wide and 25 lines tall, so there are exactly 1000 possible characters; the computer keeps them in memory, starting at address 1024—the Video Base in line 2 of FINDME.

You can easily play with the memory map. Clear the screen with <SHIFT><CLR>, and then type

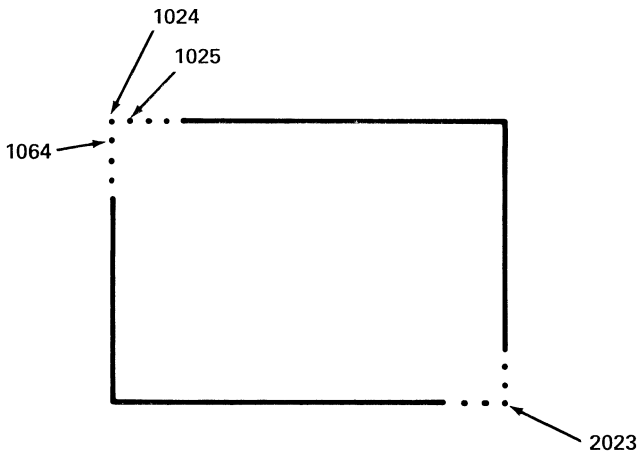
```
POKE 1024,65(RETURN)
```

The letter A should appear in the upper left-hand corner of your screen—the home position, or beginning of video memory.

---

FINDME works now, up to the point where you give it a pair of numbers. If it doesn't, work on it until it does. Debugging is one of the primary skills of the programmer: you must learn to think linearly, like the machine. Exactly where does it stop doing the right thing? What was it supposed to do? What did it do? When you can answer that, list the code and *exterminate the bug!*

Now that the grid is drawn and the X plotted correctly, add the lines that check the player's guess.




SCREEN CHARACTER MAP

```
100 PRINT "YOU FOUND ME!";
180 FOR T=1 TO 2000:NEXT:GOTO 10
200 PRINT "YOU MISSED ME";
210 UX=QX:UY=QY:Z=87:GOSUB 1200
220 FOR T=1 TO 1000:NEXT
230 FOR N=1 TO 13:PRINT "[CR] [CR]";:NEX
T
240 PRINT "[CD][CD]";:GOTO 70
```

## THIS IS HOW THE PROGRAM WORKS

The main program begins at line 10 by clearing the screen and typing the game's name. Lines 20, 30, and 40 draw a grid on the screen. Line 50 gets random coordinates for the hider, and line 60 hides her.

Lines 70 and 80 ask the player to locate the hider, then check to see if the location is correct. If the player gets the right answer, lines 100 and 110 display "YOU FOUND ME" for a decent period before clearing the screen and circling back to line 10 for another spot. If the player is wrong, line 90 sends program control to line 200.

Line 200 prints "YOU MISSED ME" and line 210 plots the spot the player entered for comparison. The most common error is giving the Y answer first; when the wrong point is plotted on the screen, it's easy to see the reversal. Line 220 leaves the message on the display for a moment, then line 230 wipes it out. Line 240 circles back to line 80 for another try.

## ADVANCED PROJECT: SUPER-FINDME

In this program, the computer doesn't put an X to mark the hider's spot, but issues two pairs of tones to help the player find it. The tones are hints: the closer together the tones get, the closer you are to the hider.

SUPER-FINDME blends the FINDME program with another program, or routine, called TWONOTE:

```
1900 REM                              * TWONOTE *
1910 N1=1:N2=2:GOSUB 2000
1920 GOSUB 2100
1930 INPUT N1,N2:GOTO 1920
2000 S=54272:FOR L=S TO S+24:POKE L,0:NE
XT:K=1
2010 POKE S+5,9:POKE S+6,0:POKE S+24,15
2030 READ HF(K):READ LF(K):K=K+1:IF K<11
 THEN 2030:RETURN
2050 DATA 4,48,4,180,5,71,6,71,7,12
2060 DATA 8,97,9,104,10,143,12,143,14,24
2100 POKE S+1,HF(N1):POKE S,LF(N1):POKE
S+4,33
2110 FOR T=1 TO 125:NEXT
2120 POKE S+4,32:FOR T=1 TO 32:NEXT
2130 POKE S+1,HF(N2):POKE S,LF(N2):POKE
S+4,33
2140 FOR T=1 TO 250:NEXT
2150 POKE S+4,32:FOR T=1 TO 32:NEXT
2160 FOR L=S TO S+4:POKE L,0:NEXT
2190 RETURN
```

The first lines (1900-1920) are called a "programming stub," a sort of test-bed to make sure the program works right before it gets incorporated in a larger program. They initialize the computer by sending out to line 2000, where the sound generators are set up, then send to the sound routine for two notes (line 1920), then ask you for two other notes. Be sure the numbers you supply for line 1930 are in the range from 0 to 9.

Don't worry if you don't understand the pokes in this program—that is, if it works. For a discussion of the sound generator, see your Commodore manuals.

When you are sure that TWONOTE is working (and saved in case something goes wrong), you are ready to merge the module with FINDME.

Borrowing Lines

To merge TWONOTE into FINDME to make SoundFINDME, make sure TWONOTE is in the computer's memory by typing

```
LIST
```

Now type

```
NEW
```

and then load FINDME from disk or tape. When the computer says READY, move your cursor to the home position of the screen by pressing {HOME}. You should see the cursor block flashing on the 2 in line 2000 of the listing of TWONOTE, which is still on the screen, even though you NEWed program memory and loaded another program. Now press {RETURN} once in every line of TWONOTE remaining on the screen—that should be 13 times.

If you list your program now, you should find it is 13 lines longer—the sound module from TWONOTE has been appended to FINDME.

---

There are several changes necessary to make the sound module work. First, you need to add

```
:GOSUB 2000
```

to line 2, to initialize the sound generator. The subroutine for "displaying" the hider isn't at line 1200 anymore; we use the sound subroutine at line 2100, so line 60 needs to change, too.

```
1 REM                                * SFINDME *
2 C$=CHR$(147):VB=1024:LN=0:GOSUB 2000
10 PRINT C$;       FIND ME?":PRINT " Y"
20 FOR R=8 TO 0 STEP -1:PRINT R;
30 FOR C=0 TO 9:PRINT "+  ";:NEXT C:PRIN
T:PRINT
40 NEXT R:PRINT "  ";:FOR C=0 TO 9:PRINT
 C;:NEXT:PRINT "X"
```

```
50 HN=9:GOSUB 1300:X=R:HN=8:GOSUB 1300:Y
=R
60 N1=X+1:N2=Y+1:GOSUB 2100
70 PRINT:PRINT "CAN YOU FIND ME (X,Y)";
80 INPUT QX,QY:IF QX=X AND QY=Y THEN 100
90 GOTO 200
```

There is some added code between lines 100 and 180, the "Correct" Fanfare:

```
100 PRINT "YOU FOUND ME!";
110 N1=X+1:N2=N1:GOSUB 2100:N1=Y+1:N2=N1
:GOSUB 2100
120 N1=3:N2=5:GOSUB 2100:GOSUB 2100
130 N1=7:N2=9:GOSUB 2100
170 QX=0:QY=0
180 FOR T=1 TO 500:NEXT:GOTO 10
```

and similar changes between 200 and 290:

```
200 PRINT "YOU MISSED ME";
210 N1=QX+1:N2=X+1:GOSUB 2100
220 N1=QY+1:N2=Y+1:GOSUB 2100
230 FOR N=1 TO 13:PRINT "[CR] [CR]";:NEX
T
240 PRINT "[CD][CD]";:GOTO 70
```

The new program should run now. Test it out. There is a full listing of SFINDME in the appendix for troubleshooting purposes.

An interesting footnote: you have written a program (SFINDME) that works very nicely without the video screen. (Of course, you can't turn the TV off, because the computer relies on it for the sounds as well as the picture.) Would it be possible to build a computer that didn't use the video at all? How would a computer for blind people work?

# ONECLAP

## a word-discovery game

"Is this a word, Mommy?" asks the young child, just learning to build words out of sounds. Exploring the way letters combine into words is an exciting quest for children aged five through seven. But English is an illogical language, and many words that should be spelled one way, *duz* for example, are spelled in quite another, *does*. It takes a lot of memory to know the spellings of the most misspelled words. But that's no reason why your computer can't know and help a young child discover a smaller class of words. At this age, children are fascinated by truly simple words like CAT, PET, and SIT: a letter or two, then a vowel (A, E, I, O or U) and a final consonant. I call these One-Clap words, because when you sing or say them, clapping your hands as you go, these words get only one clap, where other words get more: computer, for example, gets three claps.

This program is an ambitious programming project for the beginning programmer. It is nothing particularly complicated, but a number of modules must cooperate to do the program's work. Good luck! (Accurate typing helps, too . . . .)

```
ONE CLAP

CAT

CAT IS ON MY LIST
```

```
ONE CLAP

DUZ

DUZ IS NOT ON MY LIST
```

ONECLAP is really two games in one program. In the first game, the player presses any alphabet key, and the computer shows the letter inside the box on the screen. When the player presses <RETURN>, the computer replaces the letter with a short word containing that letter. The computer searches for a matching word randomly, so pressing the same letter several times will give the player several words containing the chosen letter. Playing with the computer in this way, the player learns the words in the computer's vocabulary.

When the player is ready to move on, typing any two (or more) letters sends the computer to the second game (or "mode" as they say in the computer world). The child types in a simple word, and the computer compares it to its vocabulary. If the computer finds the word, it flashes the YES! block; if not, it confesses that the word "is not on my list" and makes a note of it for future reference by someone with a bigger vocabulary.

Children quickly learn the "one clap" concept by clapping along with the sounds of their speaking or singing. A "one-clap" word has only one syllable, but that's a hard term to explain to younger players. This program uses an even narrower definition: *a one-clap word has only one syllable, no fewer than three nor more than four letters, and its second-to-the-last letter is A, E, I, O, or U.* It turns out that there are hundreds of such words—see how many you can find—and they comprise a large majority of most children's first written words. This program only supplies a few, but children delight in finding more. If the programmer *maintains* the program by adding newfound "good words" after each session, the program becomes a fascinating mirror of the player's burgeoning written vocabulary.

## WRITING THE PROGRAM

Several new concepts make their appearance in this program. We can build the program one module at a time, and the whole thing shouldn't take more than an evening.

Start out by borrowing a few lines from earlier programs—specifically, the box and colorful YES! from NUMS, lines 80-84 and 130-170. If you don't remember how to borrow, refer to the notes in the last chapter.

After borrowing program lines from another program, or if you need to renumber a few lines to accommodate something new, all you do is LIST the lines needing change, then move the cursor to their line numbers, correct them, press <RETURN>, and it's done!

A step at a time: for this program, we need to move the lines that make the red box at center-screen from 80-84 (where they reside in NUMS) to lines 300-340. Start out by typing

```
LIST 80-84
```

The complicated graphic lines scroll onto the screen. Be glad you won't have to retype them! Depress the <SHIFT> key and press the <CURSOR-UP> button on the right side of the keyboard's bottom line until the cursor is over the 8 in the first line's 80. You need space for one more character, so continue to depress the shift key and press the <INST> (that means "insert") key in the keyboard's upper-right corner. A space appears. Now type

```
300{RETURN}
```

The cursor drops to the next line—line 81—where you repeat the process by making it into line 310 . . . and so forth, until all five lines are renumbered.

When you LIST your program again, you will find that those lines now exist in BOTH places. Better get rid of lines 80-84—you do that by typing their line numbers on a clear line.

---

Highlights from the program: the first lines prepare the computer to work with the special kind of program we are writing. DIM A$(4,100) creates an array of memory locations (like pigeon-holes) 5 holes wide (0, 1, 2, 3, and 4) by 101 holes high, to contain the program's vocabulary. If you RUN the program now, lines 20, 30, and 40 would send the computer off on wild goose chases, because the lines at 3000, 300, and 400 haven't been written.

```
1 REM                                   * ONECLAP *
2 GOSUB 900:M$="                        ":K$=M$+"
  "

5 DIM A$(4,100):MO$="ANY LETTER":MO=0
10 PRINT:PRINT "HELLO":PRINT "MY NAME IS
 ONECLAP.":PRINT
20 INPUT "WHAT IS YOUR NAME";N$:PRINT:N=
0:GOTO 3000
```

```
30 GOSUB 300:IF LEN(T$)>1 THEN 60
40 GOSUB 400:K=K+1:IF K<10 THEN 30
50 MO$="ONECLAP":MO=1:GOSUB 300
60 F$=MID$(T$,LEN(T$)-1,1):GOSUB 500:W=0
:IF F=-1 THEN 250
70 IF A$(F,W)=T$ THEN 100
80 W=W+1:IF W<W(F)+1 THEN 70
90 GOTO 250
```

The lines beginning at 100 let the player know when he's typed a good word. You should have borrowed lines 130-170 from NUMS already.

```
100 PRINT "[CD][CD][CD]":PRINT K$,"$," I
S ON MY LIST.":GOSUB 800
110 NR=NR+1:PRINT:PRINT:IF T$="QUIT" THE
N 700
120 PRINT
130 PRINT M$;"[RU][RED] [GRN] [RED] [GRN
] [RED] [GRN] [RED] [GRN] [RED] [GRN] [R
ED] [GRN] [RED] "
140 PRINT M$;"[RU][GRN] [RU]              [
RU] "
150 PRINT M$;"[RU][RED] [RU]  Y E S !  [
RU][RED] "
160 PRINT M$;"[RU][GRN] [RU]              [
RU] "
170 PRINT M$;"[RU][RED] [GRN] [RED] [GRN
] [RED] [GRN] [RED] [GRN] [RED] [GRN] [R
ED] [GRN] [RED] "
180 PRINT:PRINT M$;"                    MORE
?";
190 GET Q$:IF Q$="" THEN 190
```

Line 200 and beyond take care of words not on the list.

```
200 IF Q$="N" THEN 700
210 GOTO 50
220 B=B+1:IF B<11 THEN 290
230 GOTO 700
250 PRINT "[CD][CD][CD]":PRINT K$;T$;" I
S NOT ON MY LIST.":GOSUB 800
260 B$(B)=B$(B)+T$+" ":IF LEN(B$(B))<36
THEN 290
270 B=B+1:IF B<11 THEN 290
280 GOTO 700
290 GOTO 50
```

The lines beginning at 300 clear the screen, draw the input box (which you borrowed from NUMS), and send out for the player's keystrokes. (That long series of blocks in line 350 moves the cursor back up into the center block. You get there by pressing the <CURSOR-RIGHT> key 22 times, the <SHIFT><CURSOR-UP> key twice, the <CTRL><0> once to turn reverse video off.)

```
300 GOSUB 900:PRINT "[RV][RED]        "
310 PRINT K$;"[RV] [RV]      [RV] "
320 PRINT K$;"[RV] [RV]      [RV][RED] "
330 PRINT K$;"[RV] [RV]      [RV] "
340 PRINT K$;"[RV]         "
350 PRINT "[CR][CR][CR][CR][CR][CR][CR][
CR][CR][CR][CR][CR][CR][CR][CR][CR][CR][
CR][CR][CR][CR][CR][CD][CD][RV]";
360 GOSUB 1000
390 RETURN
```

Lines 400-490 match single keystrokes to words containing those letters (from the first part of the game.)

```
400 F$=T$:GOSUB 500:IF F<0 THEN 420
410 HN=W(F):GOSUB 1300:W=R:R=F:GOTO 480
420 HN=4:GOSUB 1300:W=0
430 IF LEFT$(A$(R,W),1)=T$ THEN 480
440 IF RIGHT$(A$(R,W),1)=T$ THEN 480
450 W=W+1:IF W<W(R)+1 THEN 430
460 R=R+1:W=0:IF R<5 THEN 430
470 R=0:GOTO 430
480 T$=A$(R,W):PRINT "[CR] [CR]";T$
490 GOSUB 800:RETURN
```

The lines from 500 to 590 set up a variable F to work with the vowels in the words.

```
500 F=-1:IF F$="A" THEN F=0
510 IF F$="E" THEN F=1
520 IF F$="I" THEN F=2
530 IF F$="O" THEN F=3
540 IF F$="U" THEN F=4
590 RETURN
```

You can skip the scoring section (lines 700-990) for now, and proceed to the keyboard subroutine beginning at line 1000. Line 1050 is another cursor positioning command: four <CURSOR-RIGHT>s to get the message outside the box, then the text, and then 18 <SHIFT><CURSOR-LEFT>s to get back to the right place. If you're wondering what CHR$(13) is, it's the BASIC language equivalent of the <RETURN> key; line 1090, for instance, translates, "If the Input-string is not equal to <RETURN> then print it and add it to the Text-string."

```
1000 T=0:T$="":REM                    * KEYB *
1010 GET I$:T=T+1:IF T<250 THEN 1060
1020 IF MO=0 AND LEN(T$)=1 THEN 1050
1030 IF MO=1 AND LEN(T$)=3 THEN 1050
1040 GOTO 1060
1050 PRINT "[CR][CR][CR][CR]PRESS <RETUR
N>[CR][CR][CR][CR][CR][CR][CR][CR][CR][C
R][CR][CR][CR][CR][CR][CR][CR][CR]";:T=0
1060 IF I$="" THEN 1010
1070 IF MO=1 AND LEN(T$)<3 THEN 1090
1080 IF I$=CHR$(13) THEN RETURN
1090 IF I$<>CHR$(13) THEN PRINT I$;:T$=T
$+I$
1100 T=0:GOTO 1010
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

You can test this module directly after you have it typed: type

```
GOSUB 1000
```

Nothing exciting happens—the computer just sits there and hums. Type something in, like

```
HELLO, COMPUTER
```

and press <RETURN>. The computer responds with READY. Nothing very interesting? Ask the computer to print T$—you can take a short-cut and type

```
? T$
```

The computer should respond by typing your message again—the keyboard subroutine displayed your keystrokes, while capturing them in the Test-string.

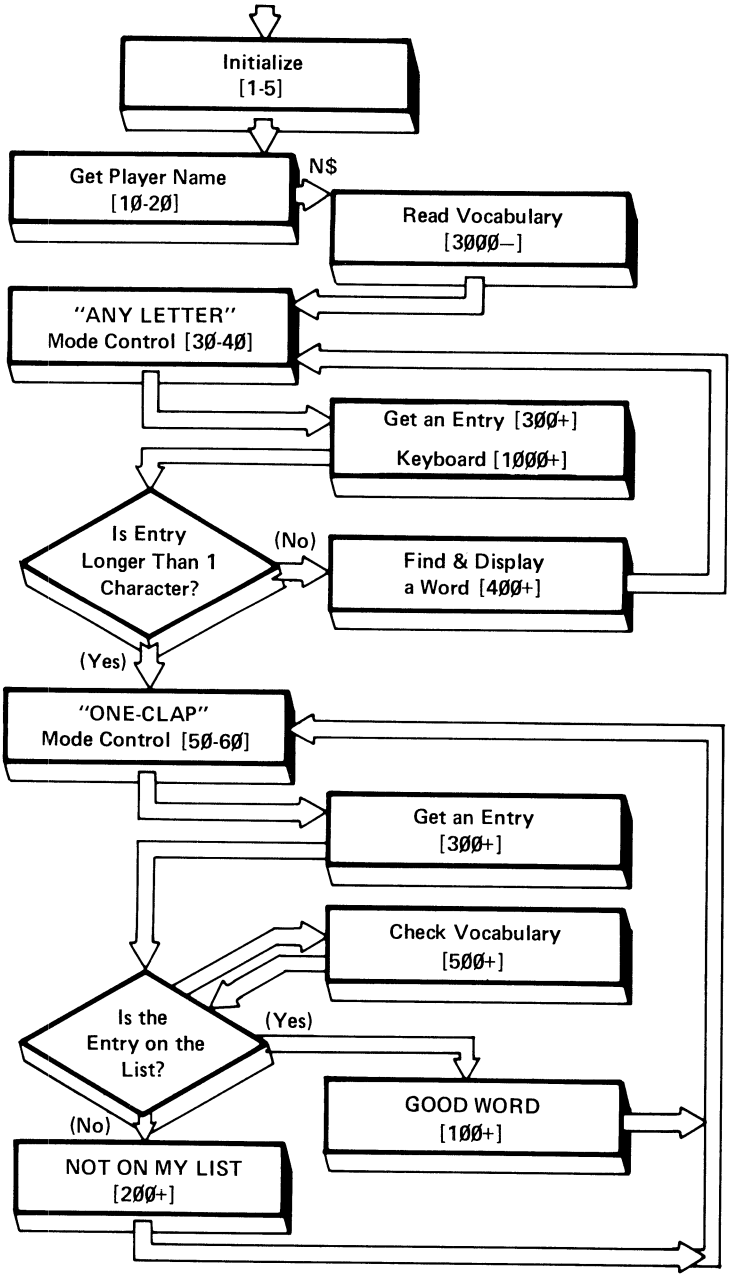
Line 1300 is our old friend the Random Number Maker, borrowed already from NUMS.

Lines 3000-3200 read the vocabulary from the data lines 9000-9990 into the array called A$(V,W) where V is the vowel number (starting with 0 for A) and W is the word number. After the last word for each vowel is stored, the computer keeps track of how

many words it has for each vowel in an array of numbers called W(V). This module is designed to read as many as 101 words for each vowel from the data, provided, of course, that the A words come before the E words, and the very last data entry is ZZZ.

```
3000 PRINT "EXCUSE ME, ";N$
3010 PRINT "I'M READING MY LIST NOW."
3020 PRINT:W=0
3030 GOSUB 3200:IF MID$(A$,2,1)="E" THEN
 3050
3040 A$(0,W)=A$:W=W+1:GOTO 3030
3050 W(0)=W-1:W=1::A$(1,0)=A$
3060 GOSUB 3200:IF MID$(A$,2,1)="I" THEN
 3080
3070 A$(1,W)=A$:W=W+1:GOTO 3060
3080 W(1)=W-1:W=1::A$(2,0)=A$
3090 GOSUB 3200:IF MID$(A$,2,1)="O" THEN
 3110
3100 A$(2,W)=A$:W=W+1:GOTO 3090
3110 W(2)=W-1:W=1::A$(3,0)=A$
3120 GOSUB 3200:IF MID$(A$,2,1)="U" THEN
 3140
3130 A$(3,W)=A$:W=W+1:GOTO 3120
3140 W(3)=W-1:W=1::A$(4,0)=A$
3150 GOSUB 3200:IF A$="ZZZ" THEN 3170
3160 A$(4,W)=A$:W=W+1:GOTO 3150
3170 W(4)=W-1
3190 GOTO 30
3200 READ A$:PRINT M$;A$:RETURN
```

Long program! But now it's ready to test. On the next page you will find a programming tool, called a flow chart, to help you debug this program. Follow the lines on the chart while the computer follows the program logic. If the computer does something unexpected, you'll know where to look for a problem. Computers are literal to a fault: if a single character is wrong, the computer will complain, or will do the wrong thing.


Initialize
[1-5]
Get Player Name
[1Ø-2Ø]
N$
Read Vocabulary
[3ØØØ–]
"ANY LETTER"
Mode Control [3Ø-4Ø]
Get an Entry [3ØØ+]
Keyboard [1ØØØ+]
Is Entry
Longer Than 1
Character?
(No)
Find & Display
a Word [4ØØ+]
(Yes)
"ONE-CLAP"
Mode Control [5Ø-6Ø]
Get an Entry
[3ØØ+]
Check Vocabulary
[5ØØ+]
Is the
Entry on the
List?
(Yes)
GOOD WORD
[1ØØ+]
(No)
NOT ON MY LIST
[2ØØ+]

Commodore BASIC lets you leave out almost all spaces, so you can save memory. (Other BASICs don't let you do that.) The problem with leaving spaces out, though, is that it makes the lines hard to read. I left the spaces in to help you understand the programs, but you can leave them out if you want.

---

Once you have ONECLAP performing the way it should, you can add lines 700-990, which let the player QUIT the game and see the words the computer doesn't have on its list.

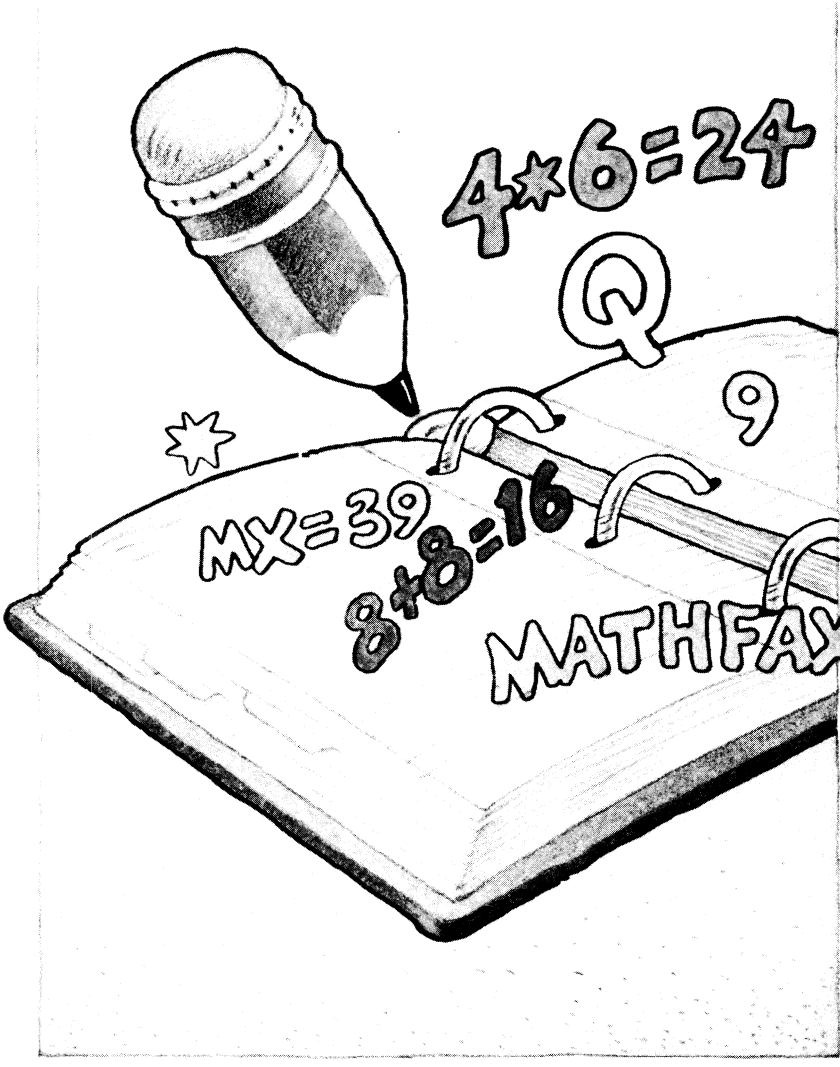
There is a full listing of ONECLAP in the appendix.

```
700 PRINT C$:PRINT K$;"ONE CLAP":PRINT
710 PRINT N$;" FOUND";NR;"GOOD WORDS."
720 IF B$(0)="" THEN 790
730 PRINT:PRINT "WORDS NOT ON MY LIST"
740 FOR N=0 TO B-1:PRINT B$(N):NEXT
790 END
800 FOR T=0 TO 999:NEXT:RETURN
900 PRINT CHR$(147);"[RU]";N$;" "
910 FOR N=1 TO 3:PRINT:NEXT
920 PRINT K$;MO$:PRINT:PRINT K$;
990 RETURN

9000 DATA BAT,CAT,EAT,FAT,HAT,MAT,PAT,RA
T,SAT,TAT,VAT,WAX
9010 DATA BET,GET,JET,LET,MET,NET,PET,SE
T,VET,YET
9020 DATA BIT,FIT,HIT,KIT,PIT,SIT,TIT,WI
T,ZIP,QUIT
9030 DATA COT,DOT,GOT,HOT,JOT,LOT,NOT,PO
T,ROT,SOT,TOT
9040 DATA BUT,CUT,GUT,HUT,JUT,NUT,PUT,RU
T
9990 DATA ZZZ
```

# school skills

intelligence games



The next six programs take the computer from the crawling and walking stage to the running and jumping stage. There should be something interesting and useful for anyone old enough to press the keys. There is one game that will have the quickest of mathematicians scribbling away with pencil and paper along with the rest of us: it's possible, but tough.

The computer is a smart blackboard: write a "little program" that teaches some specific lesson quickly. These six programs polish various school skills—logic, memory, spelling, math, history, and word problems—as painlessly as possible.

The programming involves several new techniques. As you construct these programs, you will see the reasons for building certain parts first, like building the roof and walls to keep the rain off, but only after the foundation and floor are solid.

# PATTERN

## deduce mathematical patterns

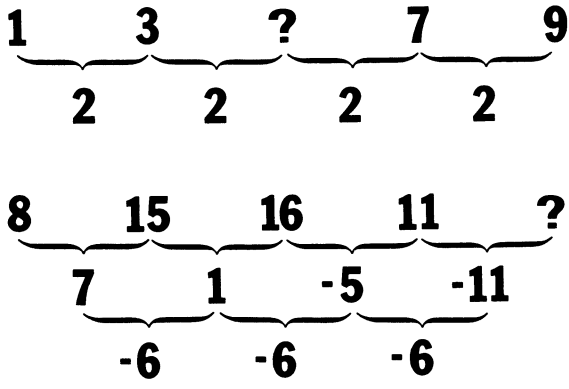
```
   PATTE  S
  PATTE N
 PATTE NS
PATTER
PATTER S
PATTERN
PATTERNS-a patterns recognition game
```

This makes up mathematical puzzles like

```
1 3 ? 7 9          and    8 15 16 11 ?
(series one)              (series two)
```

The player's job: decipher the rule and supply the unique missing number. Such puzzles often find their way into intelligence tests and math textbooks; the ability to deduce the answers quickly is often taken for intelligence.

Some patterns, of course, are easier than others: the first series is the odd numbers, and the missing member is 5—a six-year-old can easily find that. The second one is not quite so easy—the answer is zero.




There is a systematic method for solving all but the alphabetic puzzles with pencil and paper. Write out the series, then bracket each pair of numbers and note the difference between them. Then bracket each pair of differences and again calculate the differences. Repeat the process until you recognize the pattern, and you'll be able to work backwards to find the missing number. Good Luck!


PATTERN is the computer version of an age-old number game: it presents a series of numbers with some mathematical rule governing their relationship, and the player tries to find the rule and supply the missing number. This program uses five different *algorithms* (fancy computer talk for rules) to generate the sequences: additive, multiplicative, incremental, exponential, and alphabetic, to give them properly mathematical names. Mixing them up randomly, this game can sharpen the number skills of almost anyone. Does practice make perfect?

The program begins by asking you to select a difficulty level from 0 (easy) to 9 (hard). The complexity of each series is based on this level. The computer presents a series with one missing member, which you calculate and type in. If you are right, the computer makes up another series, but if you're wrong, it shows you the same series with a different missing number.

## WRITING THE PROGRAM

PATTERNS lends itself to modular construction, because it involves several different methods for deriving number series—each method gets its own set of lines. Start with a framework that includes two different series-generation algorithms, make sure they work, and then add more modules.

The program uses our old standby, line 1300: the Random Number Maker. You can copy it from another program following the procedure in COUNTEM. The main framework of the program goes like this: Line 5 sets the low number variable LN for the Random Number Maker. Line 60 gets the difficulty level; line 70 calculates the high number for randomly choosing the series algorithm, which is then chosen in line 80.

```
1 REM                                   * PATTERN *
5 LN=1



60 PRINT:INPUT "EASY (0) OR HARD (9)";DF
70 PRINT:HN=INT(DF/2)+1:IF HN>5 THEN HN=
5
80 GOSUB 1300:ON R GOTO 100,200,300,400,
500
```

Lines 1000-1090 are a very simple "keyboard" to get the player's answer. Lines 1100-1190 leave lots of space for polishing up the scoring routine—see the suggestions at the end of this section.

I believe we have seen line 1300 before.

```
1000 PRINT
1010 INPUT "WHAT IS MISSING";Q$
1020 IF Q$="" THEN 1090
1030 IF ASC(Q$)>64 THEN 1090
1040 Q=VAL(Q$)
1090 RETURN
1100 PRINT "YOU ARE RIGHT."
1190 GOTO 70
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

The rest of the program makes up and prints the series. The easiest module is additive, and it serves as a pattern for "knocking off" the other algorithms.

```
100 HN=4*(DF+1):GOSUB 1300:S=R
120 HN=2*(DF+1):GOSUB 1300:I=R
130 HN=2:GOSUB 1300:L=3+R
140 HN=L:GOSUB 1300:M=R
150 FOR N=1 TO L
160 IF N=M THEN PRINT " ? ";:GOTO 180
170 PRINT S+I*(N-1);
180 NEXT:A=S+I*(M-1):GOSUB 1000:IF Q=A T
HEN 1100
190 GOTO 140
```

Line 100 sends out for a random Starting number for the series; 120 sends for an Increment (the amount to increase each time a new series number is calculated). Line 130 gets the length of the series, and line 140 decrees which of the series members will be Missing. The lines from 150 to 180 are a loop that checks to see if this member is Missing or printed (160), calculates and prints the latter (170), and loops back if the series has not reached its allotted length—that's the FOR (line 150) . . . NEXT (180) loop. When you understand what a loop does, you have perceived the power of the computer.

It's easy to *duplicate* lines in Commodore BASIC. All you do is LIST the original lines, move the cursor onto the line you want to duplicate, change its line number (and anything else you need to change to make it exactly right). Is everything exactly right? Excellent: the last thing you do before leaving the line is press <RETURN>. That snap-shoots another line into the computer's program memory.

---

You can test-drive the program as it stands (but be careful, you'll crash if you try any difficulty over 2!) but I wanted something a little more interesting, so I added another module: the multiplicative:

```
200 HN=4*SQR(DF+1):GOSUB 1300:S=R
220 HN=2*(DF+1):GOSUB 1300:I=R
225 HN=INT(DF/2):GOSUB 1300:I2=R
230 HN=2:GOSUB 1300:L=3+R
240 HN=L:GOSUB 1300:M=R
250 FOR N=1 TO L
260 IF N=M THEN PRINT " ? ";:GOTO 280
270 PRINT S*I*N;
280 NEXT:A=S*I*M:GOSUB 1000:IF Q=A THEN
1100
290 GOTO 240
```

If you have the feeling you have seen those lines before, you are remembering the module we just finished. You won't be surprised by the next two modules, either.

```
300 HN=DF:GOSUB 1300:S=R
320 HN=DF+1:GOSUB 1300:I=R
325 HN=INT(DF/2):GOSUB 1300:I2=R
326 IF DF/2=INT(DF/2) THEN 330
327 I2=I2-HN:IF I2=0 THEN I2=1
330 HN=2:GOSUB 1300:L=3+R
340 HN=L:GOSUB 1300:M=R
350 I1=I:FOR N=1 TO L
```

```
360 IF N=M THEN PRINT " ? ";:A=S+I1*(N-1
):I1=I1+I2:GOTO 380
370 PRINT S+I1*(N-1);:I1=I1+I2
380 NEXT:GOSUB 1000:IF Q=A THEN 1100
390 GOTO 340
```

This algorithm has two increments (and gets particularly nasty if you let negative numbers in: that's what those mean-looking lines at 325, 326, and 327 accomplish. When you get this module working, play around with the numbers and relationships of this section (if you're interested in becoming a mathematician.)

And now for something utterly different: letters! Words are a kind of series, and the logic is certainly different, especially in English. The module beginning at line 400 shifts the program from the coolly numeric to the jungle of textual calculation: strings.

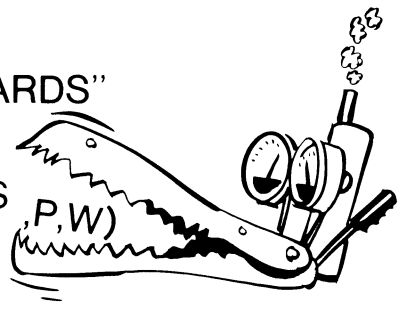
```
400 IF FR=0 THEN 1600
410 HN=FR:GOSUB 1300:A$=A$(R-1):P=1
420 L=LEN(A$):HN=L:GOSUB 1300:M=R
430 HN=2:GOSUB 1300:I=R:IF I=1 THEN 450
440 I=-1:P=L
450 IF P=M THEN PRINT " ? ";:Z$=MID$(A$,
P,1):GOTO 470
460 PRINT " ";MID$(A$,P,1);" ";
470 P=P+I:IF P=0 OR P>L THEN 490
480 GOTO 450
490 GOSUB 1000:IF Q$=Z$ THEN 1100
495 IF I<0 THEN 440
496 P=1:GOTO 450
```

The first line sends to line 1600 only the first time this module gets called—those lines are next. The next three lines are familiar. Line 440 sets things up to step backwards through a word—like this: S D R A W K C A B. The computer language we are using, BASIC, has a very powerful way of playing with words. We'll see more of that later. MID$—pronounced *mid-string*—cuts a *sub*string out of a string, like this:

A$ = "BACKWARDS"

W$ = MID$(A$,P,W)

W$ = "KWA"



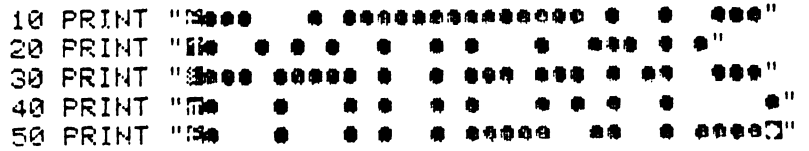
```
1600 DIM A$(100):N=0
1610 READ A$(N):IF A$(N)="END" THEN 1690
1620 N=N+1:GOTO 1610
1690 FR=N:GOTO 410
9000 DATA BIRTHDAY,WASHINGTON,FISH
9010 DATA COMPUTER,HORSE,BOAT
9990 DATA "END"
```

There are dozens more possible algorithms that could be included in this game—and lots of memory in the computer, too—but I propose to give you only one more: exponents.

```
500 HN=SQR(DF):GOSUB 1300:S=R
520 HN=2:GOSUB 1300:I=R+1
530 HN=2:GOSUB 1300:L=3+R
540 HN=L:GOSUB 1300:M=R
550 FOR N=1 TO L
560 IF N=M THEN PRINT " ? ";:A=S*M^I:GOT
O 580
570 PRINT S*N^I;
580 NEXT:GOSUB 1000:IF Q=A THEN 1100
590 GOTO 540
```

(The world of numbers is so much cleaner to program in!)

A bit of fancywork to dress up the beginning of the program, and we can wrap this one and move along to another kind of patterning.



The heiroglyphics just inside the quote in each line are <CTRL><number> keys, which make the following characters print in that color. Different combinations look better on different video screens, so experiment.

There are many more interesting relationships and interrelationships for making this kind of puzzle, and finding and defining them is a superior kind of intelligence-building game.

# REMEMBER

polish memory skill

## CAN YOU TYPE WHAT YOU SEE?

Another trait frequently identified as intelligence is the ability to glimpse random strings of symbols, like

```
7 8 3 5 2          or          7 : A ) Q # 2
```

then reproduce them: sort of rote-memory muscle-building. Of course some people will be able to handle much longer strings than others. In writing Computer Aided Instruction, the concept of a program's "ceiling" defines the point past which learners cannot benefit from a program. This program is a good example of a game without a ceiling: the lower levels of difficulty present short strings for a long interval, but the highest levels are much too long and fast for me! In other words, this game should challenge anyone.

REMEMBER times and scores the play—interesting computer functions for any kind of gaming program. The methods and modules that time and score can be easily transported to other programs.

REMEMBER devises strings of numbers, letters, and typographical symbols, flashes them on the screen for a measured period, then clears the screen and invites the player to reproduce the string. Playing difficulty ranges from trios of numbers shown for several seconds (level 0) up to nine characters displayed for less than a second (level 9). You must pay constant, focussed attention to the screen, and your typing must be error-free.

The program is written "open-ended" so that you can add an elaborate timing and scoring section.

## WRITING THE PROGRAM

The Random Number Maker appears once again in this program; you might like to borrow it from a program already in memory.

Lines 5-30 get the computer ready for work: getting the player's difficulty level and calculating the time delay (20), the length of the strings to be devised, and setting the high number in accordance with the difficulty level. The FOR . . . NEXT loop from line 40 to 80 puts the test string together—line 60 adds a space and a number produced by STR$(R) each time around, line 70 takes care of letters and symbols.

*Commodore 64 notes*

Programming with Cursor Motion Keys

The graphics blocks in line 6 are reverse-video-Q—you get them by typing a <CURSOR-DOWN> character just inside the quotes. The reversed-circles in line 200 are produced by the <SHIFT><CURSOR-UP>. There is a chart of symbols in the appendix if you need further data.

You may have noticed that sometimes the cursor keys move the cursor in a program line, and sometimes they print characters. The rule: if you type a " symbol, all following cursor-motion keys will be entered as characters (until you press another " or <RETURN>). When you are editing a line, the cursor-keys may start overwriting existing characters after the first change. If this isn't what you meant to happen, press <RETURN> and then move your cursor back into the line to fix the damage—one set of changes at a time.

The <DELete/INSert> key behaves the same way.

---

The rest of REMEMBER manages the video display:

```
100 PRINT C$;DF;M$;T$
110 FOR T=0 TO TD:NEXT
120 PRINT C$;M$;"?    ";:Q$="":T=0:GET I$
200 PRINT:PRINT:PRINT"EXACTLY?"
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

Lines 130-190 are a "keyboard" routine that captures the player's keystrokes and compares them with the computer's string after each one. As soon as the player has completed entering the symbols—assuming they are exactly right—the program passes into the scoring process. If the player isn't precise, a <RETURN> submits the incorrect string for evaluation. The delay at line 290 maintains the rhythm of the program—it is exactly as long as the viewing cycle in line 110.

Once the program is performing properly, you can add a scoring routine:

```
210 IF Q$="" THEN 300
220 LT=LEN(T$):LQ=LEN(Q$):SC=0:FOR N=2 T
O LT STEP 2
230 IF N>LQ THEN 260
240 IF MID$(T$,N,1)=MID$(Q$,N,1) THEN SC
=SC+1
250 NEXT:PRINT "SYMBOLS RIGHT:";SC
260 SC=INT((SK*L)/((T*TK)*(SC/LT)))+1
280 PRINT "TIME:";T*TK;"SECONDS":PRINT "
SCORE";SC:TS=TS+SC:NQ=NQ+1:TT=TT+T*TK
290 FOR T=1 TO TD:NEXT:GOTO 40
300 PRINT C$;"REMEMBER...":PRINT
310 PRINT "TOTAL SCORE        ";TS
320 PRINT "STRINGS ATTEMPTED ";NQ
330 PRINT "AVERAGE SCORE/$    ";TS/NQ
340 PRINT "TOTAL SECONDS      ";TT
350 PRINT "AVERAGE  TIME/$    ";TT/NQ
390 FOR T=1 TO 3*TD:NEXT:GOTO 40
```

Lines 200 to 250 count the number of correct symbols-in-position. Line 260 calculates a "score" based on time and the correctness of the entry, and line 270 displays the results and keeps track of the score.

To see the scoreboard, press <RETURN> only (lines 300-390).

For the timer to work, you'll have to make these additions to the keyboard routine:

```
130 GET I$:T=T+1
150 IF I$="" THEN 130
170 IF I$=CHR$(13) THEN PRINT:GOTO 210
180 Q$=Q$+" "+I$:PRINT I$;" ";
190 IF Q$<>T$ THEN 130
```

## ADVANCED TOPICS

The timer is tricky because its count is governed by the number of computer cycles inside the timing loop—in this program, that's how long it takes the computer to do all the steps between lines 130 and 190. The TIME constant TK may have to be calibrated for the computer's "seconds" to correspond to "real time." The easiest way to do this is by carefully clocking the time between the appearance of the ? on the video screen—when the clock starts—and your last keystroke. Adjust TK until the computer time is accurate.

An adventurous programmer could adapt the scoring module to several of the other programs in this book. Recalibration of the clock would be required.

# SPELL

a spelling driller

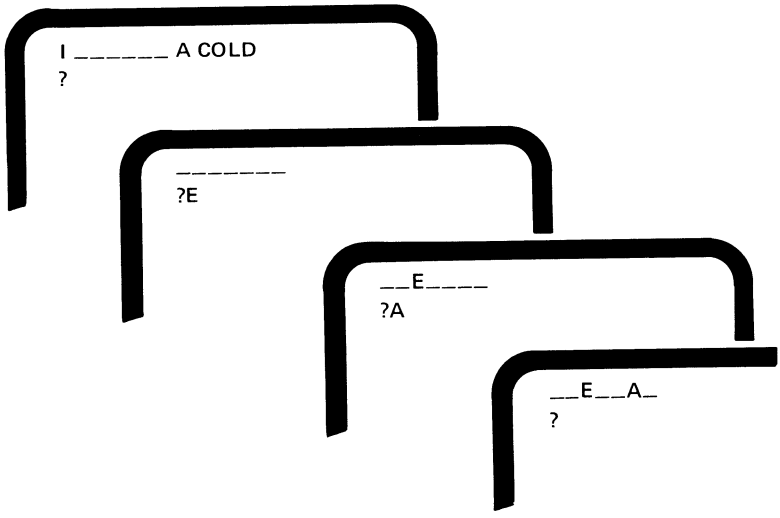
Spelling, at least in English, is not very different from remembering random strings of letters. There are two ways the computer helps. One way is with spelling-drillers, like the program in this chapter, that present words-to-be-learned in an interesting and instructive way.

Another way is through *spelling checkers,* programs that comb through a written work for unrecognizable words, then offer the writer a chance to survey the dictionary and, if necessary, correct the spelling. (This book was checked for mistakes using a program called *ProofReader.* See the comments about Word Processing in the Appendix.) Is it possible that the computer will liberate future children from spelling drillers?

Everybody learns in a slightly different way. This program uses four different ways to try to find everyone's good side: first, you type your list (and, if desired, a set of "hints"); next, you play REMEMBER (from the last chapter); then you supply the correct spelling for each hint; finally, you play Hang-Man with the spelling list.

For the programmer, this program is an easy challenge, because several different modes must work together within a single program.

```
ENTER WORDS                ZZZ TO QUIT
1: CAUGHT
HINT: I CAUGHT A COLD
2: COUGH
HINT: HE HAS A BAD COUG
  *** ARE YOU SURE ?
2: COUGH
HINT: HE HAS A BAD COUGH
3: CATCH
HINT: DON'T CATCH MY COLD
4: FIRE
```



SPELL uses four different practice strategies to help learn spelling. The first method is the straightforward entry of the spelling list. After entering the words, you are asked for hints for each of the words on the list—a short sentence or phrase for each word. (You can skip the hint-entry by pressing <RETURN> instead of a hint for the first word.)

```
CATCH   PLAY ----- WITH ME.
CAUGHT   I ------ A COLD.
```

When all the words are entered, SPELL goes into the "Flash" mode; it flashes the word on the screen for a moment, then clears the screen, and you type the word in. If you are right, you are informed; if not, the correct spelling is shown. When you have gotten ten words right, the program promotes you.

The "Clue" mode shows the clue, and you are invited to provide the missing word. Again, if you get it right, the computer tells you, but if you miss, the computer shows the right word. When you have gotten twenty more words right you go on.

The "Hang-Man" mode is familiar to all: the computer presents you with the right number of blanks, and you press the letters you think belong in the word. If you are right, the computer puts the letter in the right place, or places, if the letter occurs more than once in the word; but if you're wrong, the computer keeps track. You only get twice as many wrong guesses as there are letters in the word, so proceed thoughtfully.

## WRITING THE PROGRAM

SPELL can be put together a mode at a time. The input section and the word flasher look like this:

```
1 REM                                        * SPELL *
2 DIM A$(100),B$(100),C$(100)
5 C$=CHR$(147)+"[CD][CD][CD][CD][CD][CD]
[CD]":PRINT C$:M$="                          ":PRINT
 M$;"SPELL":PRINT
8 R$(1)="RIGHT!![CR][CR]":R$(2)="CORRECT
  ":R$(3)="YOU GOT IT!"
9 TD=1000:LN=1:LD=9
```

```
10 GOTO 500
20 FOR N=1 TO ND:A$=A$(N)
30 PRINT A$:B$(N)=LEFT$(A$,LD):LT=LEN(A$
)-LD:C$(N)=RIGHT$(A$,LT)
40 IF RIGHT$(B$(N),1)=" " THEN B$(N)=LEF
T$(B$(N),LEN(B$(N))-1):GOTO 40
50 NEXT:FOR T=0 TO TD:NEXT
60 HN=ND:GOSUB 1300:PRINT C$;M$;"   ";B$(
R)
70 FOR T=0 TO TD:NEXT:PRINT C$;M$;:INPUT
 Q$
80 IF Q$=B$(R) THEN 100
90 PRINT:PRINT M$;"SORRY, IT'S ";B$(R):W
R=WR+1:GOTO 130
100 HN=3:GOSUB 1300:PRINT:PRINT M$;R$(R)
110 RI=RI+1
130 FOR N=0 TO TD:NEXT:IF RI<10 THEN 60
```

The reverse-video Qs in line 5 are <CURSOR-DOWN>s. The variables in lines 8 and 9 are the messages that flash when you get it right, the Time Delay, the Lowest Number (for our old reliable Random Number Maker), and the Length of the Data. Please adjust any of these (except LN) to please yourself. There's an unattached wrong-answer counter in line 90. Could you hang anything on it?

You need two more routines to make the program work.

```
500 PRINT "ENTER WORDS
ZZZ TO QUIT":N=1:H=1
510 PRINT N;":";:INPUT B$(N):IF B$(N)=
"ZZZ" THEN 590
520 IF H=0 THEN 580
530 INPUT "HINT:";C$(N)
540 IF C$(N)="" THEN H=0:GOTO 580
550 P=1:LB=LEN(B$(N)):L=1+LEN(C$(N))-LB
560 IF MID$(C$(N),P,LB)=B$(N) THEN 600
```

```
570 P=P+1:IF P>L THEN PRINT "ARE YOU
SURE?": GOTO 510
575 GOTO 560
580 N=N+1:GOTO 510
590 ND=N-1:GOTO 20
600 L$=LEFT$(C$(N),P-1):R$=RIGHT$(C$(N)
,1+LEN(C$(N))-(P+LB))
610 A$="":FOR B=1 TO LB:A$=A$+"-":NEXT:
C$(N)=L$+A$+R$
620 PRINT B$(N)
690 GOTO 580
```

Test the "Flash" mode, and make sure it's working right. When building a complicated program, you want to make sure that everything works perfectly before adding more complexity.

Adding the Clue mode requires only a few more lines:

```
150 HN=ND:GOSUB 1300:PRINT C$;"    ";C$(R
)
160 PRINT:PRINT M$;:INPUT Q$:GOTO 80
```

This module hitchhikes on the "Right!!" messages system set up for the "Flash" mode.

### *Who is in charge here?*

When testing one of the advanced modes, it tries my patience to play through earlier modes. So I write temporary *detours* into the program to proceed directly to my desired module. For example, you go straight to Hang-Man by inserting this line

```
55 RI=99:GOTO 200
```

You can take the line out for a full-playing game at any time.

The Hang-Man module is a bit more demanding:

```
140 IF RI>30 THEN 200


200 WG=0:RG=0:HN=ND:GOSUB 1300:PRINTC$;M
$;
210 T$=B$(R):LT=LEN(T$):FOR Z=1 TO LT
220 PRINT "- ";:D$(Z)=MID$(B$(R),Z,1)
230 NEXT:PRINT:PRINT M$;"GUESS  ?";
240 GET I$:IF I$="" THEN 240
250 PRINT I$;:Z=1
260 IF I$=D$(Z) THEN 310
270 Z=Z+1:IF Z<=LT THEN 260
280 PRINT "[CR] [CR]";:IF RF=1 THEN RF=0
:GOTO 240
290 WG=WG+1:IF WG<2*LT THEN 240
300 PRINT:PRINT "YOU LOSE":GOTO 130
310 PRINT:PRINT "[CD][CD]";M$;:IF Z=1 TH
EN 330
320 FOR P=1 TO Z-1:PRINT "[CR][CR]";:NEX
T
330 PRINT D$(Z);:D$(Z)="":RG=RG+1:RF=1
340 IF RG=LT THEN PRINT:PRINT:GOTO 100
350 PRINT:PRINT M$;"[CR][CR][CR][CR][CR]
[CR][CR][CR][CR]";:GOTO 270
360 GOTO 240
```

Lines 210 and 220 place the right number of blanks on the screen. Your guess (in line 230) gets checked against all the letters in lines 260, and if it isn't found, your guess is erased (280—the graphics strings are <SHIFT><CURSOR-LEFT>, then a <space> to clear out your guess, then another <SHIFT><CURSOR-LEFT> to reposition the cursor.), a strike is noted against you, and if you've missed too many (290), you lose. If, however, your guess fits into

the word, the code starting at line 310 moves the cursor to the right place(s) in the display line, prints your guess, and returns for another.

It's easy to put the data inside the program or save it on a disk or tape file. The program is less flexible, but in many ways it is easier to use. The program in TIMELINE deals with data in the first way; after reading it, you could change SPELL ever so slightly to work the same way. And SORT would allow you to store several lists and call them into SPELL. If you are careful to structure each datum so the first nine characters were the spelling word (or spaces), and the rest of the data is the hint (like this):

```
CAUGHT    I ------ A COLD
COUGH     HE HAS A BAD ------
```

you could substitute the following for the program lines beginning at 500:

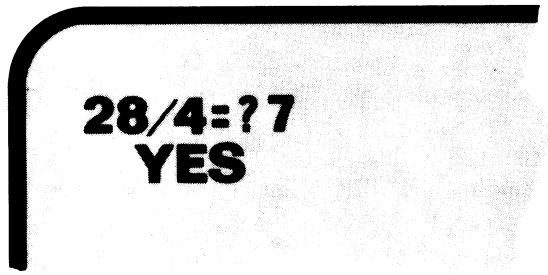
```
500 INPUT "FILENAME TO LOAD";F$
510 F$=F$+",SEQ,R":OPEN 2,8,4,F$:N=1
530 INPUT#2,A$(N):IF A$(N)="ZZZ" THEN ND
=N-1:CLOSE2:GOTO 20
540 N=N+1:GOTO 530
```

and add this code at the beginning:

```
10 GOTO 500
20 FOR N=1 TO ND:A$=A$(N)
30 PRINT A$:B$(N)=LEFT$(A$,LD):LT=LEN(A$
)-LD:C$(N)=RIGHT$(A$,LT)
40 IF RIGHT$(B$(N),1)=" " THEN B$(N)=LEF
T$(B$(N),LEN(B$(N))-1):GOTO 40
50 NEXT:FOR T=0 TO TD:NEXT
```

# MATHFAX

practice math facts




## A DRILLING GAME FOR FACTS

"Two and two are four; four and four are eight. . . ." The mathematical facts of life are never easy, and everyone has a pet bugaboo (for me, it's eight times seven.) This program helps you perfect your grasp of those facts through drill-and-practice.

When we finally "know our math facts," it is usually the product of several learning systems working together: rote memory, pattern recognition, and familiarity from practice. This program helps people learn (and can be found in innumerable versions on every computer in the world) because it combines all three avenues to help us learn.

For the programmer, this is an interesting program because it offers a chance to do something often done, but with our own style. Some programmers strive to write *short* code—could this program be written in one line? in five? Other programmers work to make their programs "friendly" by error-trapping and hand-holding. I have spent many lines on tailoring each game to the player's abilities and offering graphic help when the answer is wrong.

MATHFAX quizzes you on your pluses, minuses, timeses, and divided-byes. It even offers a little help if you get it wrong.

```
*MATH FACTS*

1=ADD 2=SUBTRACT 3=MULTIPLY 4=DIVIDE ? 4
HIGHEST NUMBER FOR ADDING? 10
FOR MULTIPLYING?  7
```

You start out by tailoring the drill to your abilities: some players know more facts than others. You choose the highest operation—if you choose 3 for MULTIPLY then you get Adding and Subtracting too—and the highest number for adding. If you choose multiplying or dividing, you supply the highest number there, too. Some children will have mastered all their pluses to 7, for example, and will know some of the simple timeses, too—to 3, possibly. You can tailor the quiz to challenge a learner at almost any level. Adults might wish to improve their speed by setting the high numbers to 20 for addition and 13 for multiplication—that would give most everyone a workout.

```
6 * 4 = ? 24

      YES

28 / 4 = ? 8
NO
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
```

Once you've defined the drill, you are given random problems to solve. If you answer right, the computer responds "YES" but if you are wrong, the computer draws a picture that should help you figure out the right answer for next time.

## WRITING THE PROGRAM

As usual, we can sneak up on this program, and get it working a section at a time. Since the program deals with four operations, there are four different problem-makers. The main program and the addition-maker go like this:

```
1 REM                                * MATHFAX *
5 C$=CHR$(147)+"[CD][CD][CD][CD][CD][CD]
[CD][CD]":M$="                    "
10 PRINT C$;M$;"* MATH FACTS *":LN=0:PRI
NT
20 INPUT "1=ADD 2=SUBTRACT 3=MULTIPLY 4=
DIVIDE";HO:HO=HO-1
30 INPUT "HIGHEST NUMBER FOR ADDING";HA:
IF HO<2 THEN 50
40 INPUT "            FOR MULTIPLYING";HM
50 HN=HO:GOSUB 1300:OP=R+1:PRINT C$;M$;:
ON OP GOTO 60,60,90,90
60 HN=HA:GOSUB 1300:N1=R:GOSUB 1300:N2=R
:ON OP GOTO 70,80
70 PRINT N1;"+";N2;"= ";:A=N1+N2:GOTO 13
0
80 PRINT N1+N2;"-";N1;"= ";:A=N2:GOTO 13
0
90 HN=HM:GOSUB 1300:N1=R:GOSUB 1300:N2=R
:ON OP GOTO 0,0,100,110
```

You need something to deal with right and wrong answers:

```
130 INPUT Q:IF Q<>A THEN 160
140 PRINT:PRINT M$;"         YES"
150 FOR T=0 TO 999:NEXT:GOTO 50
```

You need a Random Number Maker—it is exactly like the one now appearing in programs from cover to cover of this book:

```
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

Test the program, but be sure not to ask for anything harder than addition yet. And *don't make any mistakes!* The program isn't ready.

With that section working, we can add the other three operations:

```
100 PRINT N1;"*";N2;"= ";:A=N1*N2:GOTO 1
30
110 IF N2=0 OR N2=0 THEN 90
120 PRINT N1*N2;"/";N1;"= ";:A=N2:GOTO 1
30
```

Dividing by zero is risky business—in fact, zero is tricky anywhere in division, so line 110 rejects any problems with zeroes in them.

Test again—using all the operations—but *you still can't make any mistakes.* If you do, the programs crashes. Oh, go on, make a mistake, just to see what happens.

The capabilities of the Commodore 64 are hardly touched by this little program, so let's add a little bit of help. Most people don't like tests unless right answers and maybe even hints are supplied when you miss.

```
160 PRINT:PRINT M$;"          NO":ON OP GO
TO 170,165,180,180
165 NT=N1+N2:GOTO 175
170 NT=N2
175 PRINT M$;:GOSUB 200:PRINT M$;:NT=N1:
GOSUB 200:GOTO 150
180 N2=0
190 IF N2<N1 THEN PRINT M$;:NT=N2:GOSUB
200:N2=N2+1:GOTO 190
195 GOTO 150
200 NP=0
210 IF NP<NT THEN PRINT "Q";:NP=NP+1:GOT
O 210
220 PRINT:RETURN
```

For addition and subtraction, two lines of beads are drawn on the screen . To get the right answer in addition, you count all the beads. In subtraction, you only count the beads that aren't paired in two lines of beads. For multiplication, a rectangle of beads is drawn. To get the right answer in multiplication you count all the beads—you may know a shortcut, called a "times-table." In division, you know the number of beads in one side, and the total number of beads, so you just need to count the side you don't know.

To do all this counting, you may need to lengthen the delay in line 150, or change line 210 like this:

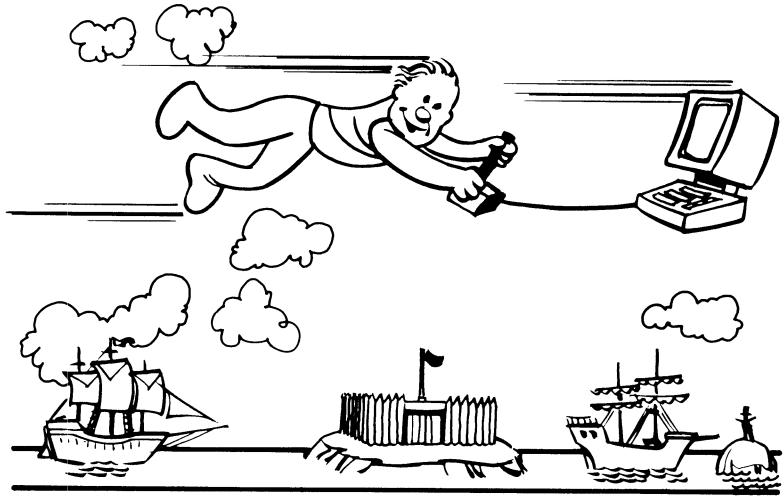
```
210 PRINT:NEXT:INPUT "OKAY";Q$:GOTO 150
```

## ADVANCED STUDIES

A timer and a scorekeeper would make this program more challenging for those of us who feel confident of our math facts. It would be an easy matter to borrow them from REMEMBER. You would need to change the "keyboard" at line 130 to one that uses GET I$ and includes a timing loop.

# TIMELINE

historical events




## A TIME MACHINE

The "mind of the computer" is sometimes compared to the human mind. The comparison is partly right. Microcomputers have tens of thousands of memory locations—the Commodore 64 has more than 64 thousand—lined up in one long row, like links in a chain. Each link can hold (and always does) one of 256 symbols. History is the story that ties a long row of events together. The microcomputer's memory can easily be made into a perfect model of events in history. This program roves over the terrain of dates and events like a time machine we get to drive!

Writing the program—and understanding the concepts behind the program—are the best part of this chapter. The part of the human mind lacking in the computer is the "relationship finder" we humans use to spot linkages between two apparently unrelated facts. The computer provides a smooth magic carpet to survey the facts, but it is not very good at noting interesting coincidences. My computer never said, "Did you know that the first steam-powered boat didn't run up the Hudson River until 32 years after Watt invented the steam engine? Is it my imagination or are things speeding up?"

For a machine, whether made of flesh and blood or silicon and gold, to make observations like that takes lots of smarts. Possibly someone who builds TIMELINE's simple magical historical carpet may later make key breakthroughs in teaching intelligence to machines.

```
                * TIMELINE *

        1000
Leif Ericson discovers America
        1522
first circumnavigation of the earth
        1607
Jamestown settled in Virginia
        1620
Pilgrims arrive at Plymouth Rock
        1756
French and Indian War begins
  -  -  -  -  -  -  -  -  -  -  -
- earlier     + later      What next?
```

TIMELINE provides a skeleton for a "terrain" of historical events for you to explore. You can see what happened last, happens next, what happened 10 years ago, what happens in 20 years—assuming you programmed the computer to know the answer. Now that I think of it, the computer wouldn't mind if you made up a History of the Future.

## WRITING THE PROGRAM

The program reads up to 101 dates and events and stores them in memory (lines 10 and 20). The main program begins at line 40, where it displays

```
1 dim n(100),n$(100):m$="
2 c$=chr$(147)+"           * TIMELINE *"+
chr$(13):print c$:n=1
10 read n(n),n$(n):if n$(n)="end" then 3
0
20 n=n+1:goto 10
30 ln=n-1:for n=1 to ln:print m$;n(n):pr
int n$(n):print:next:y=3
40 print c$:for n=y-2 to y+2:print m$;"-
";n(n);"-":print n$(n):print:next
50 print "-  -  -  -  -  -  -  -  -  -"
60 print "  - earlier  + later   What ne
xt?";:gosub 1000
65 if q$="+" then y=y+4:goto 80
70 if q$="-" then 90
75 y=y+1
80 if y>ln-2 then y=ln-2
85 goto 40
90 y=y-4:if y<3 then y=3
95 goto 40

1000 get q$:if q$="" then 1000
1010 return
```

Lines 60-70 find out what you want to do next (using a keyboard subroutine at line 1000). Lines 80 and 90 make sure you don't run out of facts, then loop you back to a new display in line 40.

There is lots of room for data in lines 100-999. Be sure you put your favorite facts in order.

If you don't have any favorite facts, here are a few of mine to help show you the format:

```
10 read n(n),n$(n):if n$(n)="end" then 3
0
20 n=n+1:goto 10
30 ln=n-1:for n=1 to ln:print m$;n(n):pr
int n$(n):print:next:y=3
40 print c$:for n=y-2 to y+2:print m$;"-
";n(n);"-":print n$(n):print:next
50 print "-  -  -  -  -  -  -  -  -  -"
60 print "  - earlier  + later   What ne
xt?";:gosub 1000
65 if q$="+" then y=y+4:goto 80
70 if q$="-" then 90
75 y=y+1
80 if y>ln-2 then y=ln-2
85 goto 40
90 y=y-4:if y<3 then y=3
95 goto 40
99 stop
100 data 1000,"Leif Ericson discovers Am
erica"
110 data 1522,"first circumnavigation of
 the Earth"
120 data 1607,"Jamestown settled in Virg
inia"
130 data 1620,"Pilgrims arrive at Plymou
th Rock"
140 data 1756,"French and Indian War beg
ins"
150 data 1775,"James Watt invents steam
engine"
160 data 1776,"Declaration of Independen
ce"
170 data 1778,"Captain Cook discovers Ha
waii"
180 data 1787,"U.S. Constitution signed"
190 data 1791,"Bill or Rights ratified"
```

```
200 data 1803,"Louisiana Purchase"
210 data 1807,"Robert Fulton's steamboat
 up the Hudson"
220 data 1834,"Charles Babbage's Analyti
cal Engine"
230 data 1846,"Potato Famine - U.S.-Mexi
co War"
240 data 1849,"California Gold Rush"
250 data 1858,"First Trans-Atlantic cabl
e"
260 data 1861,"Civil War begins"
270 data 1865,"Abraham Lincoln shot"
280 data 1869,"Golden spike: railroad ac
ross the U.S."
290 data 1888,"George Eastman invents th
e Kodak camera"
999 data 999,end
```

The last data line must end with "end," or else the computer will give you an "out of data" error.

That completes the program. Test, debug, and save it. Of course, it would be a simple matter to use a program like SSORT to enter data, sort it, and save it, then modify the program so it could be read from a storage device.

# MINI-WOMBATS

## for word problems

I want computer games to surprise and delight me. I invented Wombats* watching children "tricking" computers by typing in nonsense words and false names. Peals of laughter greeted

```
WELL DONE, BRTLSPYX!
NOW TRY 3 + 5 = ??
```

Mini-WOMBATS is a small—but expandable!—word-problem generating game intended for experienced and/or patient programmers. WOMBATS is a small expert system; its expertise lies in building simple English stories that ask number questions. Its stories include the player, a friend, and an object of her choice. Which points out a problem a computer might have telling stories: gender! It is OK for the player to make up joke names and objects, but the computer should speak as elegantly as it can. Pronouns and verbs should agree with their nouns. (In this program, some do, and some don't.)

The *string-handling* in this program is intense, hinting at the incredible complexity that *natural language programs* will need to understand the languages of humans.

The player chooses the names of the people and the things in the stories, and then the computer makes up number problems about them.

At the beginning of play, the highest level of operations (1 = add to 4 = divide), the highest number for addition, and, if needed, the highest number for multiplication, are chosen.

---

*The original version appeared in *Creative Computing* in October, 1981, page 216.

sample dialogue (player dialogue in italics):

```
Please tell me your name? Damiana

= + = + = + = + = + = + =
Sienna found a bag containing 7 wombats.
She already has 5 wombats at home. How
many does she have now? 12

 You got it!

 -- -- -- -- -- -- -- -- --
Name a person? bob
Person's names begin with a CAPITAL.
Remember the SHIFT key, and
please try again, Damiana.

 -- -- -- -- -- -- -- -- --
Name a person? Bob
Is Bob a boy or a girl? boy

 ++ ++ ++ ++ ++ ++ ++ ++ ++
Name an object? flat bed truck

 = + = + = + = + = + = + =
Bob had 13 flat bed trucks in his
pocket, and later won 15 more in a bet
with Damiana. How many did he have then?
28

Well done.

 = + = + = + = + = + = + =
Bob dug 4 flat bed truck traps in the
forest, and caught a total of 36 flat
bed trucks. On the average, how many
did he find in each trap? 9
```

## WRITING THE PROGRAM

Write this program in several sittings, carefully saving its parts after each session. The program is long and takes precise typing—kind of like putting a computer together from a kit.

The framework of the program begins with the introduction, which prepares the computer for the problems and tailors the problems to the player:

```
1 rem"                          * miniWOMBATS *
5 c$=chr$(147)+"[CD][CD][CD]":m$="

8 poke 53272,23:rem: sets lower case on
9 sx=1:p$="Sienna":j$="wombat"
10 print c$;m$;"* math facts *":in=0:pri
nt
20 input "1=add 2=subtract 3=multiply 4
divide";ho:ho=ho-1
30 input "highest number for adding";ha:
if ho<2 then 50
40 input "           for multiplying";hm
50 print c$
55 input "Please tell me your name";n$:i
f asc(n$)<192 then gosub 1120:goto 55
60 hn=3:gosub 1300:if r=3 then gosub 110
0
70 gosub 1300:if r=3 then gosub 1200
80 hn=ho:gosub 1300:op=r+1:if op>2 then
hn=hm:goto 90
85 hn=ha
90 gosub 1300:n1=r:gosub 1300:n2=r:on op
 goto 100,300,500,700
```

The graphics blocks in line 5 are <CURSOR-DOWN> characters to bring the text down a few lines. This program requires lowercase, so the POKE in line 8 makes sure they are on.

The addition problems are next. Lots of string handling.

```
100 hn=2:gosub 1300:on r+1 goto 110,140,
170
110 q$=p$+" found a bag containing"+str$
(n1)+" "+j$+"s."":gosub 1400
120 q$=q$+"already has"+str$(n2)+" "+j$+
"s at home. How many does ":gosub 1420
130 q$=q$+"have now":goto 290
140 q$=p$+" had"+str$(n1)+" "+j$+"s in "
:gosub 1480:q$=q$+"pocket, "
150 q$=q$+" and later won"+str$(n2)+" in
 a bet with "+n$+". How many did "
160 gosub 1420:q$=q$+"have then":goto 29
0
170 q$=p$+" receives 2 envelopes. One co
ntains"+str$(n1)+" "+j$+"s; the other"
180 q$=q$+" contains"+str$(n2)+". How ma
ny does ":gosub 1420
190 q$=q$+"have now":goto 290
290 a=n1+n2:goto 900
```

P$ is the name of the current Person, J$ is the name of the object. STR$(n1) changes a number into a text string—you may not see any difference, but the computer does.

The lines from 900 to 1090 deal with right and wrong answers:

```
900 t$=q$:print:print " = # = # = # = #
= # =":print
910 lt=len(t$):r$=left$(t$,39)
920 lr=len(r$):if lr=lt then 960
930 if right$(r$,1)=" " then 950
940 r$=left$(r$,lr-1):lr=lr-1:goto 930
950 t$=right$(t$,lt-lr):print r$:goto 91
0
960 print r$;"[CD][CD][CU][CU]";:input q
:if q=a then 1000
```

```
970 print:print "Woops! That's not right
."
980 print "The right answer is";a;"."
990 goto 60
1000 hn=3:gosub 1300:on r goto 1010,1020
,1030
1010 print "You got it!":goto 1090
1020 print "Well done.":goto 1090
1030 print "That's right!"
1090 goto 60
```

The routines that get new persons and objects go like this:

```
1100 print:print " -- -- -- -- -- -- --
-- --":print
1110 input "Name a person";p$:if asc(p$)
>192 then 1150
1115 gosub 1120:print:goto 1110
1120 print "Person's names begin with a
CAPITAL."
1130 print "Remember the SHIFT key, and"
:print "please try again, ";n$;"."
1140 return
1150 print "Is ";p$;" a boy or girl";
1160 sx=0:input sx$:if sx$="boy" then sx
=2
1170 if sx$="girl" then sx=1
1180 if sx>0 then return
1190 print "I'm sorry. I don't know any
questions":print "about ";sx$;"s."
1195 print "Please try again.":goto 1100
1200 print:print " ++ ++ ++ ++ ++ ++ ++
++ ++":print
1210 input "Name an object";j$
1290 return
```

Subroutines in the 1400s supply necessary pronouns, like this:

```
1400 if sx=1 then q$=q$+"She ":return
1410 q$=q$+"He ":return
1420 if sx=1 then q$=q$+"she ":return
1430 q$=q$+"he ":return
1440 if sx=1 then q$=q$+"hers ":return
1450 q$=q$+"his ":return
1460 if sx=1 then q$=q$+"her ":return
1470 q$=q$+"him ":return
1480 if sx=1 then q$=q$+"her ":return
1490 q$=q$+"his ":return
```

With these modules intact, you should be able to test the program, but only in first gear: addition. When you have everything operating correctly, we can move on to the three other operations.

Subtraction:

```
300 n2=n2+n1:hn=2:gosub 1300:on r+1 goto
 310,340,370
310 q$=p$+" hides"+str$(n2)+" "+j$+"s, a
nd you find"+str$(n1)+". "
320 q$=q$+"How many are still hidden":go
to 490
340 q$=p$+" had too many "+j$+"s and gav
e you"+str$(n2)+". Later, ":gosub 1420
350 q$=q$+"lost all of ":gosub 1440:q$=q
$+"and you gave"+str$(n1)
360 q$=q$+" back. How many do you still
have":goto 490
370 q$="Yesterday, "+p$+" bought"+str$(n
2)+" "+j$+"s, but this morning "
380 gosub 1420:q$=q$+"could only find"+s
tr$(n1)
390 q$=q$+". How many were missing":goto
 490
490 a=n2-n1:goto 900
```

Multiplication:

```
500 hn=2:gosub 1300:on r+1 goto 510,540,
570
510 q$=p$+" won"+str$(n1)+" coupons at t
he fair. ":gosub 1400
520 q$=q$+"exchanged each coupon for"+st
r$(n2)+" "+j$+"s. How many does "
530 gosub 1420:q$=q$+"have now":goto 690
540 q$=p$+" built a machine to make "+j$
+"s. It has made"+str$(n1)
550 q$=q$+" each day for"+str$(n2)+" day
s. How many has it made":goto 690
560 print r$;:input q:if q=a then 600
570 q$="A flying saucer deposits"+str$(n
1)+" silvery spheres in "+p$
580 q$=q$+"'s back yard."+str$(n2)+" "+j
$+"s jumped out of each one. How many "
590 q$=q$+j$+"s are rampaging around "+p
$+"'s house":goto 690
690 a=n1*n2:goto 900
```

Division:

```
700 if n1=0 then hn=hm:gosub 1300:n1=r:g
oto 700
710 n2=n1*n2:hn=2:gosub 1300:on r+1 goto
 720,750,780
720 q$=p$+" dug"+str$(n1)+" "+j$+" traps
 in the forest, and caught a total of"
730 q$=q$+str$(n2)+" "+j$+"s. On the ave
rage, how many did ":gosub 1420
740 q$=q$+"find in each trap":goto 890
750 q$=p$+" buys a packet with"+str$(n2)
+" "+j$+" seeds in it. The directions"
760 q$=q$+" tell ":gosub 1460:q$=q$+"to
plant"+str$(n1)+" in each hole. "
```

```
770 q$=q$+"How many holes should ":gosub
 1420:q$=q$+"dig":goto 890
780 q$=p$+" shared"+str$(n2)+" "+j$+"s w
ith"+str$(n1-1)+" of ":gosub 1480
790 q$=q$+"friends. How many did each ge
t":goto 890
890 a=n2/n1
```

I left one particularly glaring grammatical error in: "Bob had 1 flat bed trucks in his pocket. . . ." Could someone *please* write a few lines of code to trap that error?

There is room for adding more questions, and a simple score-keeping section, with a display of the score from time to time. The original version of the program also had a diagnostic section for teachers: the ability to print the problems as they were solved and summaries of problems solved, and an extensive HELP capability. Interested programmers should look up that classic October 1981 edition of *Creative Computing*.

# the computer as a tool

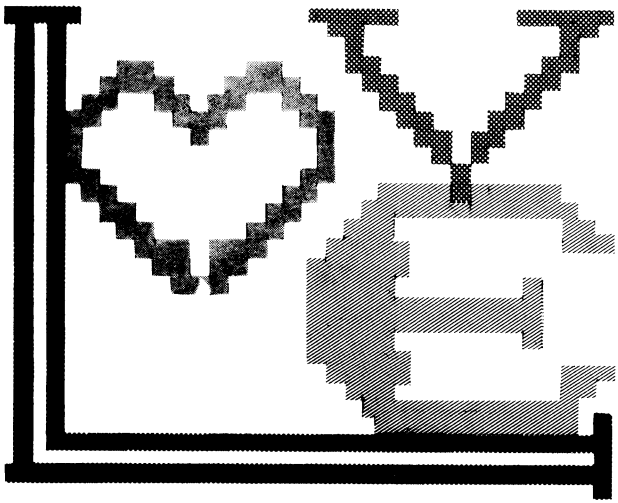
The last six chapters are about tools you can build to use your computer for more serious (and helpful) jobs: as a number- and data-cruncher.

The programming is no trickier than in earlier chapters, but some of the programs often consist of more modules, complex looping, and other precise code. If at first it makes no sense, read the chapter over again. Sleep on it. Understanding will dawn.

Be prepared to start thinking about all the great ways you can rewrite these programs to do all manner of other useful things. The nicest thing about computers is that they are willing to try anything twice.

# PAINT

## electronic crayons



The computer makes a wonderful set of electronic color crayons. And it's fun to see our own pictures on the TV screen for a change! Using this program, anybody can doodle.

For the programmer, there are two challenges: this program fiddles with color commands and saves and retrieves data with tape or disk. Fortunately, you can take on one challenge at a time.

They say a picture is worth a thousand words.

PAINT gives the player control over the color video capabilities of the Commodore, to create pictures and illustrations using colored squares. This turns out to be an enjoyable way to gain mastery over the Commodore's cursor control keys, while making pretty pictures.

When the program starts, your "brush" is the block in the middle of the "paper." You can move it up, down, left, and right, using the cursor keys in the lower right-hand corner of the keyboard. Notice that RIGHT and DOWN work fine just by pushing the keys, but UP and LEFT require you to hold down one of the <SHIFT> keys before pressing the cursor key. To get the "brush" where you want it takes practice.

Fortunately, you can practice moving the brush all you want without making a mark on the paper. When you're ready for paint, you press the space bar, and the brush begins depositing color wherever it moves. You can change the brush's color by pressing the COLOR keys at the top of the keyboard—they have numbers on top, but in this program it's the color names on their fronts that count. Color number 7, BLUE, is interesting: it's the background color, and can be used to erase mistakes. By the way: there is one unlabeled color, number 9, which paints a kind of orangey-brown on my TV.

After some practice, you will be able to interlace colors easily, turning your brush on and off using the space bar with precision. If you've spoiled a painting beyond retrieval, you can Zap it—just press Z and then Y.

You can Save a painting by pressing <S>. There is lengthy number-crunching involved, turning the colored picture into terms the computer knows how to save—it takes about a minute, so be patient. As soon as the crunching is done, the computer asks for a name for your picture.

## *Commodore 64 notes*

### Saving Files to Disk

Remember that the Commodore Disk System will not let you save anything with a filename that it finds already on the disk. This is to protect you from wiping out precious programs. You have to be very careful when you save something: you may *think* you saved it, but if the red light on the front of the disk drive is flashing, there has been a disk error, and your work probably didn't get saved.

If the light is flashing, and you are trying to save a painting, you should try it again—even though it takes a long time—using a new filename.

When choosing a name for a painting, program, or anything else you plan to save, it's a good idea to make the name as descriptive as possible, so when you look at the disk directory you will know what each file is. And another thing: when you plan to save several versions of a file,

number them—BOAT1, BOAT2, BOAT3—and keep track of the last number you used.

Saving Files to Tape

The programs in this book use the Input/Output commands for a disk drive system, but all can be adapted to work with tape storage, too. Tape commands are usually simpler than disk commands. Please check the details in your Commodore manuals.

---

If you have a painting saved, you can load it by pressing <L>. The program asks for the name of the file to load. (If you tell it a file that isn't there, it will give you an error message.)

You can put Text—words and symbols—into your painting by positioning your cursor, choosing the color for your text—white is most legible—and then pressing <T>. The brush continues to print each key you press until you press <RETURN>.

And you can Quit the program at any time by pressing <Q>.

## WRITING THE PROGRAM

The main framework of PAINT is quite simple.

```
1 REM                                    * PAINT *
5 C$=CHR$(147):PRINT C$;:X=20:Y=12:A=83
6 VB=1064:CB=55336:CF=1
7 DIM B$(24),C$(24)
10 GOSUB 1100:GOSUB 1000
20 IF I=17 THEN Y=Y+1:IF Y>24 THEN Y=24
30 IF I=145 THEN Y=Y-1:IF Y<1 THEN Y=1
40 IF I=29 THEN X=X+1:IF X>39 THEN X=39
50 IF I=157 THEN X=X-1:IF X<0 THEN X=0
60 IF I=32 AND PF=1 THEN PF=0:GOTO 70
65 IF I=32 THEN PF=1
70 IF I>47 AND I<58 THEN PC=VAL(I$)-1:CF
=PC:GOSUB 1100
80 IF I$="Q" THEN END

190 GOTO 10
```

The first few lines clear the screen, set the brush in the middle of the paper, and point out the Video Base and Color Base for the computer. The main program starts at line 10. The group of lines that all begin IF I= deal with each key pressed—lines 20-50—decode the cursor keys and change the X-Y position of the brush in accordance with the player's commands. Lines 60 and 65 toggle—switch on and off—the brush. Line 70 takes a number and translates it into a new Paint Color. Line 80 lets the player Quit.

```
1000 REM +KEYB
1010 GET I$:IF I$="" THEN 1010
1020 I=ASC(I$)
1030 IF PF=0 THEN A=OS:CF=OC
1040 IF PF=1 THEN A=160:CF=PC
1050 GOSUB 1100:A=83:CF=PC
1090 RETURN
1100 P=VB+40*(Y-1)+X:OS=PEEK(P)
1110 POKE P,A
1120 C=CB+40*(Y-1)+X:OC=PEEK(C)
1130 IF CF>-1 THEN POKE C,CF
1190 RETURN
```

Subroutine 1000 is the "keyboard," which gets the next command from the player. Subroutine 1100 places a brushstroke on the screen.

With this much of the program entered, you can make pictures, and make sure everything is working right so far. Remember: in programming, a keystroke out of place is like a needle in a haystack. The real fun is in getting clues from a program that is not quite working and eradicating those stubborn bugs.

## *Commodore 64 notes*

Memory-mapped Video, Part Two

In addition to the block of 1000 memory locations used for the symbols on the screen, the Commodore 64 also uses another block of 1000 memory positions to keep track of each position's *color.* The symbol memory begins at 1024 (in PAINT we leave the top line blank as a command line and start at 1064), and the color memory begins at 55296 (again, PAINT skips the top line and begins a 55336.) The color of a cursor position is

changed by POKEing a number from 0 to 15 into the correct memory location—the color codes can be found in your Commodore manuals.

The Commodore 64 apparently uses the "top four bits," part of the space dedicated to each color memory position, for its own purposes, so when a program tries to deduce the color of a screen position by PEEKing, it must first subtract 16 (over and over) until the number is within the range of real colors, between 0 and 15.

---

The save module contains some fancy business to accommodate the Commodore's storage scheme: the painting must be translated into a long string of numbers that can be stored like text.

```
300 UP=UB:CP=CB:PRINT "CRUNCHING.    ":FO
R ZL=1 TO 24
310 B$(ZL)="":C$(ZL)="":FOR ZC=0 TO 39
320 B$=CHR$(PEEK(UP)):IF B$="" THEN B$=
"@"
330 B$(ZL)=B$(ZL)+B$:UP=UP+1
340 CZ=PEEK(CP)
350 IF CZ>15 THEN CZ=CZ-16:GOTO 350
360 C$(ZL)=C$(ZL)+CHR$(CZ+65):CP=CP+1
370 NEXT
380 PRINT TAB(15);ZL;"[CR][CR][CR][CR][C
R]";
390 NEXT:PRINT:PRINT"[CD]";
400 INPUT "FILENAME FOR SAVE";F$
410 F$=F$+",SEQ,W":OPEN 2,8,4,F$
420 FOR ZL=1 TO 24
430 PRINT#2,B$(ZL):PRINT#2,C$(ZL)
440 NEXT:CLOSE2
450 PRINT "[CR]";:FOR N=1 TO 40:PRINT CH
R$(20);:NEXT
490 GOTO 10
```

The graphics blocks in lines 380 and 450 are <SHIFT><CURSOR-LEFT>; the block in 390 is <SHIFT><CURSOR-UP>.

Most BASIC languages "save space" by refusing to save spaces—what you get when you press the space bar—at the beginning of data to be saved. This is called "suppressing leading spaces." In PAINT, this amounts to ignoring all the unpaintd space on the left of a drawing, which tends to scramble the art. Line 320 changes all the spaces to @ symbols.

If you can Save, you should be able to Load, too:

```
500 INPUT "FILENAME TO LOAD";F$
510 F$=F$+",SEQ,R":OPEN 2,8,4,F$
520 PRINT CHR$(147);:FOR ZL=1 TO 24
530 INPUT#2,B$(ZL):INPUT#2,C$(ZL)
540 NEXT:CLOSE2
550 VP=VB:CP=CB
560 FOR ZL=1 TO 24:GOSUB 600:NEXT
570 X=20:Y=12:GOTO 10
600 FOR VC=1 TO 40
610 B$=MID$(B$(ZL),VC,1):IF B$="@" THEN
B$=" "
620 C$=MID$(C$(ZL),VC,1)
640 POKE CP,ASC(C$)-65
650 POKE VP,ASC(B$)
680 VP=VP+1:CP=CP+1
690 NEXT:RETURN
```

Line 610 restores the spaces we had to save as @ symbols to trick BASIC into storing our art intact.

## ADVANCED TOPICS

It would be an easy matter to get the other six colors on the Commodore's palette—by changing line 70 to accept more symbols. If you make a change like that, though, be sure to write down what keys yield the new colors—this is called "documenting your program," and is very important if anyone else is going to use it.

Another way to "save" a picture is by photographing it. You need to have a camera that can be set for long exposures, because the light from a TV is weak, and a short exposure may have bars across the screen.

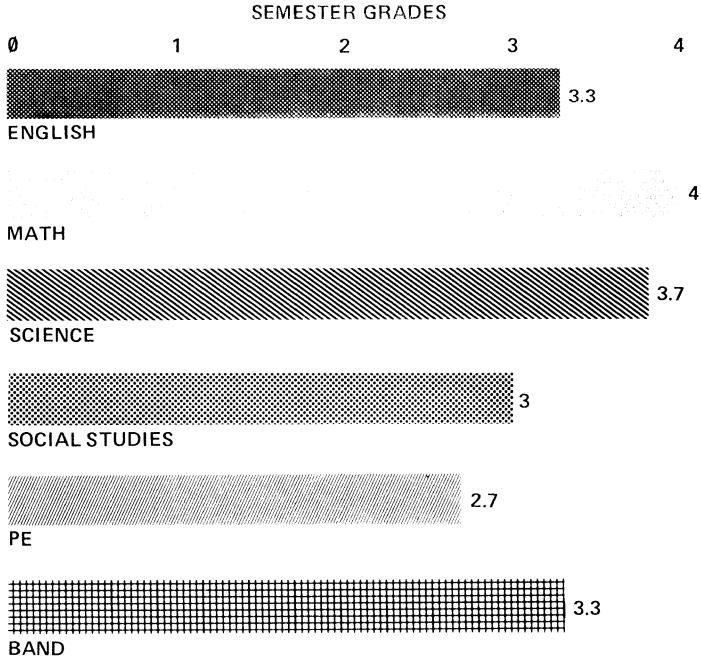
Before you push the shutter-release button, make sure of the following points:

- Eliminate screen glare. (If you can see a reflection of a window, or a wall, or yourself, you can be sure it will be in the photograph, too. Change the lighting. Taking your picture after dark in a darkened room may be the best way.)
- Fill the frame and make sure it's sharply focused. (If you can't get close enough, don't waste film. Borrow a camera with a longer lens.)
- Use a tripod or make a steady stand for your camera. (Humans move a lot in half a second. You may even need to ask people to be motionless while you make your picture, so the house doesn't shake.)
- Adjust your TV for its crispiest, juiciest image. (What looks good through the camera may be different than what's easiest to work with on the screen.)
- Bracket your exposures. Try several different exposure times, from as short as 1/60th of a second to as long as 2 seconds. Experience is the best teacher.

# BARS

## plot data with a bar graph

Computer graphics are exciting, as we can tell from prime-time TV, fantasy movies, and arcade games. Plotting graphs is a less-exciting cousin of the flashy displays we sometimes see, but it is an excellent use for the computer. It is often easier to see relationships and judge their meaning if can view them graphically:



Get the picture? For the programmer, this program is a breeze. It includes a new trick—using the printer—but it works fine without a printer.

BARS lets the user translate numerical data into bar graphs on the computer screen. The screen can handle up to seven bars, and a large range of possible numbers, and graphically represent the relationships between the different classes.

There are four possible different versions of BARS, like this:

| | *Input* | *Data* |
|---|---|---|
| screen | BARS | BARS2 |
| printer | BARS3 | BARS4 |

The screen version displays the graph on the video screen, while the printer version prints it on the Commodore Graphics Printer. The INPUT version asks the user to put the data in each time the program is run; in the DATA version the data is part of the program, so a graph is drawn automatically when the program is RUN.

All versions of the program require scaling information: what is the lowest number you will show on the graph? the highest number? How many bars will you show? and what is the overall title for your graph? The program works with data pairs—a title, and a number, like

```
USA,69
```

which represents the life expectancy of a male born in the United States. When you sit down at the computer to generate a bar graph, you will need to know all the data.

## WRITING THE PROGRAM

The second version is the easiest on the programmer, because you won't have to keep giving the computer data for the graph, so we start writing there:

```
1 REM                                    * BARS *
4 M$="                "
5 C$=CHR$(147):PRINT C$;M$;"BARS"
10 READ B:IF B=-1 THEN 30
```

```
20 B$(N)=CHR$(B):N=N+1:GOTO 10
30 INPUT"YOUR LOWEST NUMBER";LN
40 INPUT"     HIGHEST NUMBER";HN
50 INPUT"      HOW MANY BARS";NB
60 INPUT"        GRAPH TITLE";T$
70 FOR N=0 TO NB-1
80 PRINT "BAR #";N+1;TAB(13);:INPUT "TIT
LE";T$(N)
90 PRINT TAB(12);:INPUT "NUMBER";L(N):NE
XT
```

This section is a simple series of READs that get DATA from anywhere in the program. Lines 10 and 20 read the characters that get translated into bars on your video screen. Lines 30-90 read the data for your particular graph—Low Number, High Number, Number of Bars, Title string, and an individual Title string for each bar.

We can build a program in any order, so let's do the DATA next:

```
900 DATA 28,5,30,156,158,144,159,-1
910 DATA 0,75,6
920 DATA LIFE EXPECTANCY - MALES
930 DATA USA,69
940 DATA COLOMBIA,44
950 DATA NORWAY,71
960 DATA JAPAN,71
970 DATA INDIA,42
980 DATA AUSTRALIA,68
```

Line 900 contains the color control codes that govern the color of anything written by the cursor. The -1 at the end tells the READ statement in line 10 that it has read all the color controls and can go on to the next part.

```
 30 READ LN
 40 READ HN
 50 READ NB
 60 READ T$
 70 FOR N=0 TO NB-1
 80 READ T$(N)
 90 READ L(N):NEXT
100 PRINT C$:PRINT M$;T$:PRINT
120 I=(HN-LN)/5:IB=36/(HN-LN)
130 FOR N=0 TO 5:PRINT INT(LN+(N*I));"
  ";:NEXT:PRINT
140 PRINT:FOR N=0 TO NB-1
150 PRINT " ";B$(N);"[RU]";
160 FOR S=LN TO INT(L(N)*IB)-1:PRINT " "
;:NEXT
170 PRINT "[RU]";L(N)
180 PRINT "    ";T$(N)
190 PRINT:NEXT
200 GOTO 200
```

Lines 120-130 draw the reference line that tells what the bars represent. If these numbers turn out strangely, you can adjust the High Number and Low Number to make them more pleasing. The loop between line 140 and 190 turns on the reverse video in the right color (line 150—the graphics block is <CTRL><9>), makes a bar the appropriate number of spaces long (160), turns off reverse video, restores the cursor color to a nice legible light blue, prints the value for the bar (170—the graphics blocks are <CTRL><0> and <COMMODORE-KEY><7> respectively), and prints the bar's title (180).

By the way, what in the world does line 200 do? It's easy to find out: take it out, RUN the program, and see what happens.

The program is done. Test and debug it before we move on to three more versions. And be sure to save this version if you want to use it again.

It is easy to "hardcopy" your graph with a Commodore Graphics Printer. It just takes a print module:

```
30 INPUT"YOUR LOWEST NUMBER";LN
40 INPUT"    HIGHEST NUMBER";HN
50 INPUT"     HOW MANY BARS";NB
60 INPUT"        GRAPH TITLE";T$
70 FOR N=0 TO NB-1
80 PRINT "BAR #";N+1;TAB(13);:INPUT "TIT
LE";T$(N)
90 PRINT TAB(12);:INPUT "NUMBER";L(N):NE
XT

200 GOTO 200
```

Line 200 replaces the earlier line (that swallowed its own tail) with a query. Lines 210 and 220 read the codes to make graphic blocks on the printer. Line 990 contains those codes.

## *Commodore 64 notes*

Using a Printer in BASIC

The Commodore 64 uses "distributive intelligence." The devices outside the computer—disk or tape drive, printer—are microcomputers themselves, living in harmony with the Central Processing Unit (the CPU) in the Commodore keyboard. Tasks like saving and printing are delegated to these peripheral members of the computing team; once the CPU tells a peripheral to do something, it doesn't give it another thought. Their electronic conversation goes something like this:

```
OPEN3,4
```

means "I'm going to be putting things in buffer 3, printer (the printer is device number 4), and I expect you to deal with them." The printer doesn't say anything.

```
PRINT#3,stuff to be printed
```

places things in buffer 3; the printer has been looking for things to appear in buffer 3, and immediately moves anything in buffer 3 into its own

memory, or *print buffer*. Whether it prints or not depends on how the "stuff to be printed" ends: if it ends with a comma (,) or a semicolon (;) the printer waits for the rest of the line. If the "stuff" just ends, the printer prints the line, clears its own buffer, and goes back to looking for messages in buffer 3.

```
PRINT#3
```

all by itself prints whatever is in the print buffer. If there's nothing in the print buffer, it prints an empty line—exactly like PRINT for the video display.

```
CLOSE3
```

tells the printer it can relax: nothing new will be coming to buffer 3.

You can LIST programs on your printer by typing the following (no line numbers, and you can't be "inside" a program):

```
OPEN3,4
CMD3
```

The printer types READY to let you know it's paying attention.

```
LIST
```

and the program in memory is printed.

```
PRINT#3
```

clears the buffer (in case the last line ended with a <,> or <;>, and finally

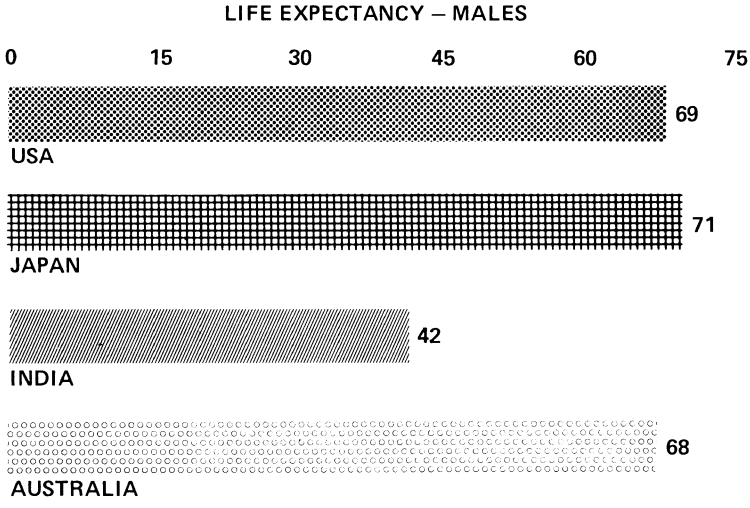
```
CLOSE3
```

so that the rest of your conversation with the printer happens on the screen, not the printer.

---

Now that you have three versions of the program, it's an easy matter to borrow the lines between 200 and 300—and don't forget line 990. While you're about it, you might play with some of the other codes your printer can make—you might think they look better on your graphs. (Check the table of the printer and screen codes in the Appendix)

You may have noticed the <"> symbol at the beginning of each bar in the print-outs. For a reason known only to the system designers, the printer won't print the interesting graphic block codes unless they are preceded by a quote symbol (as in a program listing.) In programming, especially in BASIC, we should not concern ourselves with what *should be,* but only with what *works.*


LIFE EXPECTANCY – MALES
0
15
30
45
60
75
69
USA
71
JAPAN
42
INDIA
68
AUSTRALIA

# SORT

## elementary data processing

Computers may not be good at noticing unexpected relationships, but they are superb at working with lists. A lot of paperwork consists of taking a list organized in one way and reshuffling it. In this chapter, you will write a program that accepts lists of random data and puts them into alphabetical order.

The programmer will teach the computer to remember, sort, display, save, retrieve, and print data, useful skills for any computer.

SORT is a simple program for alphabetizing a disorganized list. The SORT routine itself is short and easily understood, and can be used in other programs. In fact, it is used again in the hot-rod version of SORT at the end of this section, and in DATEBOOK and ORGANIZE in the final section of the book.

SORT is *menu-driven*—you are presented with a list (a menu) of options you can choose. You will usually start by choosing the first option, ENTER DATA, because the program doesn't do anything very interesting until it has some data to sort. You enter data one piece at a time—a single piece of data is called a *datum*—until you're done. When you've entered all the data you want to sort, you use the computer's escape code, ZZZ, and the menu reappears.

Once the computer has some data to play with, you can choose option 3 to DISPLAY your entries or option 2 to SORT them. If you choose to SORT, your data will flash by as it is manipulated by the computer. This program assumes you won't have hundreds of pieces of data to sort, and so it uses a simple, inefficient, sorting technique.

SuperSORT, the upgraded version of the sorting program, adds several more options: you can SAVE, LOAD, PRINT, and CLEAR your data.

## WRITING THE PROGRAM

If you have been working through this book a chapter at a time, there will be nothing very surprising in this program. It breaks up easily into the MENU/Control module, the SORT module, and the DISPLAY module.

The MENU module:

```
1 REM                                    * SORT *
2 C$=CHR$(147):M$="                      "
5 DIM A$(100)
6 N=1:PRINT C$
10 PRINT M$;"SORT"
11 PRINT "1: ENTER DATA"
12 PRINT "2: SORT"
13 PRINT "3: DISPLAY"
19 PRINT "9: QUIT"
20 INPUT "            OPTION";Q
30 ON Q GOTO 100,200,300,10,10,10,10,10,
900
```

This section reminds you of your choices, and sends program control off to the other modules when you have chosen.

The ENTER DATA section uses another version of the keyboard subroutine:

```
100 PRINT "ENTER DATA       ZZZ TO QUIT"
110 PRINT N;":";:GOSUB 1000:PRINT
120 IF T$="ZZZ" THEN NO=N-1:GOTO 10
130 A$(N)=T$:N=N+1:GOTO 110


900 PRINT "DONE":END
1000 T$="":PRINT TAB(5);";";
1010 GET I$:IF I$="" THEN 1010
1020 IF I$=CHR$(13) THEN RETURN
1090 PRINT "[CR]";I$;";";:T$=T$+I$:GOTO
1010
```

The graphics block in line 1000 lets the user know where the cursor is. Get it by pressing <SHIFT><F>. Line 1090 contains the <SHIFT>CURSOR-LEFT> character, then prints the most recent keystroke, and then the little cursor block again.

You can "test-drive" your program now, to see if it is getting data into the string array A$(n). Check by QUITting and asking the computer to show you your A$(n)s like this (remember, no line numbers):

```
FORK=1 TO N:PRINT A$(K):NEXT
```

The SORT module looks like this:

```
200 PRINT "SORTING"
210 FOR N=1 TO ND:P=N+1
220 PRINT A$(N),A$(P), :IF A$(N)<A$(P) TH
EN 240
230 T$=A$(N):A$(N)=A$(P):A$(P)=T$
235 PRINT "SWAPPING",
240 PRINT:P=P+1:IF P<=ND THEN 220
250 NEXT:A$(ND)=T$
280 PRINT "= = = = DONE = = = ="
```

The real work takes place in lines 220 and 230: Line 220 compares each datum with each of the data entered after it, and if it is *less than* a later entry, line 230 pops the *first* datum into a Temporary string (T$=A$(n)), pops the later datum into the earlier position (A$(p)=T$). All the PRINTing in this module is diagnostic—so you can see the work taking place inside the computer, and find out what's wrong during debugging. Programmers often insert PRINTs and STOPs to help find programming problems, then take them out when the program is running smoothly. If you take out the diagnostics—that would be line 235, and part of line 240—the routine will sort lists faster, because the computer won't have to work so hard.

The DISPLAY module is simplicity itself:

```
300 FOR N=1 TO ND:PRINT N;TAB(5);A$(N):N
EXT
380 INPUT "MENU";Q$
390 GOTO 10
```

With these lines programmed in, SORT should do everything advertised. But wouldn't it be nice if you could SAVE lists, and LOAD them, and PRINT them and MODIFY them? Read on.

## SUPERSORT—MAKING A TOY INTO A TOOL

SuperSORT is a junior version of the most important data-management tool known to Man or computer: the systematic list-reorganization utility. When you get down to it, much of the work computers do centers on taking a list of data in one order—alphabetical, perhaps, so that the subscription clerk at the local newspaper can make sure names, address, and most important, the expiration date, are all correct—into another order—a carrier-route sort for the postal folks, so that the newspaper can be distributed efficiently to the subscriber.

In addition to ENTERing, DISPLAYing, SORTing, and QUITting the way you can now also SAVE, LOAD, PRINT, MODIFY, and CLEAR your data. In other words, you can do all the things one generally needs to do with data. To do it well, you will need a disk drive.

## UPGRADING THE PROGRAM

The menu now looks like this:

```
1 REM                                   * SSORT *
2 C$=CHR$(147):M$="                     "
5 DIM A$(100)
6 N=1:PRINT C$
10 PRINT M$;"* SORT *"
11 PRINT "1: ENTER DATA"
12 PRINT "2: SORT          6: PRINT"
13 PRINT "3: DISPLAY       7: MODIFY DATA"
14 PRINT "4: SAVE          8: CLEAR"
15 PRINT "5: LOAD          9: QUIT"
20 PRINT M$;:INPUT "            OPTION";Q
30 ON Q GOTO 100,200,300,400,500,600,700
,800,900
```

The lines between 200 and 390—the ENTER, SORT, and DISPLAY modules are exactly as in SORT.

The SAVE and LOAD modules can sensibly be entered together:

```
400 INPUT "FILENAME FOR SAVE";F$
410 F$=F$+",SEQ,W":OPEN 2,8,4,F$
420 FOR N=1 TO ND
430 PRINT#2,A$(N)
440 NEXT:PRINT#2,"ZZZ":CLOSE2
490 GOTO 10
500 INPUT "FILENAME TO LOAD";F$
510 F$=F$+",SEQ,R":OPEN 2,8,4,F$
530 INPUT#2,A$(N):IF A$(N)="ZZZ" THEN ND
=N-1:CLOSE2:GOTO 10
540 N=N+1:GOTO 530
```

Lines 410 and 510 inform the intelligence within the disk drive what kind of file to expect. The program uses buffer number two, the disk a secondary address 4, and F$ tells it the name of the file, that it is a SEQuential file—a fancy word for a list—and that it is to be Written to or Read from respectively. With that information exchanged, the program can then assume that the storage device will save anything PRINTed to buffer 2, or will provide reliable data when asked to INPUT from buffer 2. (There is no magic to the number 2—use any reasonable number. If it's unreasonable, be assured that the computer's BASIC interpreter will complain.)

The SAVE module writes (line 440) a final word, ZZZ, after your last datum but before closing the file, to mark the end of the file. The READ module reads from the disk until it encounters the end-of-file marker ZZZ.

To convert this program for tape Input/Output, please consult your Commodore manuals.

PRINTing your list requires this code:

```
600 INPUT "PRINT LINE NUMBERS";Y$
610 INPUT "LOWER CASE";L$:IF L$="Y" THEN
 630
620 OPEN3,4:GOTO 650
630 OPEN3,4,7
650 FOR N=1 TO ND
```

```
660 IF Y$="N" THEN 680
670 PRINT#3,N;
680 PRINT#3,CHR$(16);"08";A$(N):NEXT
690 CLOSE3:GOTO 10
```

If you're not clear about what's happening here, take a look at the Commodore Notes in the section on BARS. You get two choices: you can print your list with or without line numbers, and in UPPER or lower case:

```
1          RANGE          calf
2          WORLD          does
3          SWAP           elf
4          PINT           extra
5          EXTRA          field
6          FIRED          fired
7          YOLK           guess
8          GUESS          half

with line numbers         without line numbers
UPPER case print              lower case print
```

Clearing data takes one simple line:

```
800 FOR N=1 TO ND:A$(N)="":N=1:ND=0:GOTO
 10
```

Modifying a datum is only slightly trickier:

```
700 INPUT "LINE TO CORRECT";X:IF X=0 THE
N 10
710 PRINT X;TAB(5);A$(X)
720 PRINT TAB(5);:GOSUB 1000:PRINT
730 IF T$="???" THEN 10
790 A$(X)=T$:GOTO 700
```

After specifying the line to correct (700), you type the correct line. Much like the ENTRY mode, you can keep correcting data until

you type ZZZ. Let's hope none of you ever *really needs* a datum ZZZ!

You should be testing each module after you add it. This is a valuable program, so don't forget to save it.

## ADVANCED TOPICS

Like all the programs in the book, superSORT is meant to be only a starting place as you move toward customizing your computer to do the work you need to do. If part of the program doesn't perform to your needs, fix it! For example, line 680 in the print module leaves a left margin by moving the printer head 8 spaces from the left margin before typing the data— accomplished by the CHR$(16);"08". It would be a simpler matter to change the "08" to a "25" and have the printer print the list closer to the center of the page.

Borrow from this program, and modify it without fear— especially if you have a working copy saved on disk!

If this program suggests useful work to you, check the two adaptations in the last section—DATEBOOK and ORGANIZE. They work with three kinds of data at a time, instead of just one.

# CALC

## a smart calculator

The way people think, and the way machines think, are sometimes far apart. The way we write

```
2 + 3 =
```

is a good example. A computer is much happier if we tell it all the players, then tell it what game to play, like this:

```
2 3 + . *
```

But computers are supposed to be our servants, not the other way 'round, so programmers have to teach them to think like people! This program teaches the computer to behave like a four-function calculator with a simple memory. It uses the BASIC symbols: * means *times* and / means *divided by.*

```
CALC:2+3-4 = 1
CALC:16*24 = 384
CALC:A/12=32 YES
CALC:2+2=5 NO
CALC:
```

*One computer language, FORTH, does exactly that. If BASIC feels clunky to you, you aren't alone: many scientists and computer specialists program exclusively in FORTH. Check into it.

CALC turns your computer into a very simple "expert system"—its expertness is just like a calculator: it's good with numbers.

You can type in complex problems using the number keys, + for plus, - for minus, * for times, and / for divided by. Like most calculators, the order of the calculations is critical:

```
3 * 5 + 2
```

is not the same thing as

```
2 + 5 * 3
```

You can use your last answer in a calculation, like this: If you calculate

```
2 + 5
```

the computer figures that out as 7. If you then ask for

```
4 * A
```

the computer substitutes your last answer, 7, for A, and informs you that the result is 28. This turns out to be a useful trait.

One more wrinkle: If you want to know if one of your calculations is right, but don't want to know the answer if you're wrong, you can type your questions like this:

```
4 * 7 = 28
```

the computer responds with a "YES" if you are right, and a "NO" if you aren't. That way you can check your homework, and learn the lessons you are supposed to, without cheating.

## WRITING THE PROGRAM

This is a hard program to test in chunks, because all of the parts are needed to make things work right. The program works by accepting a whole problem, then "parsing it"—looking at it a character at a time, and planning a solution.

The keyboard is so simple that we'll throw in the program title line, too:

```
1 REM                                   * CALC *
10 MO=0:INPUT "CALC";A$
```

The "Parser" looks at each consecutive character:

```
20 LA=LEN(A$):N=1:P=1
30 I$=MID$(A$,P,1)
40 IF I$="+" THEN O(N)=1:GOTO 180
50 IF I$="-" THEN O(N)=2:GOTO 180
60 IF I$="*" THEN O(N)=3:GOTO 180
70 IF I$="/" THEN O(N)=4:GOTO 180
80 IF I$>="0" AND I$<="9" THEN V$=V$+I$:
GOTO 190
90 IF I$="." THEN V$=V$+".":GOTO 190
100 IF I$="A" THEN V$=STR$(OA):GOTO 190
110 IF I$="=" THEN MO=1:QA=VAL(RIGHT$(A$
,LA-P)):GOTO 200
180 V(N)=VAL(V$):V$="":N=N+1
190 P=P+1:IF P<=LA THEN 30
```

When this segment is done, the problem has been broken down into Values—one for each value in a variable array called V(n)—and Operations—one for each relationship between two values, stored in O(n). There will be one less operation, we hope, than there are Values.

The next section processes the Values in the manner dictated by the Operations:

```
200 V(N)=VAL(V$):V$="":LN=N:N=1
210 IF LN=1 THEN 400
220 IF O(N)=3 THEN V(N)=V(N)*V(N+1):GOSU
B 300:N=1:GOTO 210
230 IF O(N)=4 THEN V(N)=V(N)/V(N+1):GOSU
B 300:N=1:GOTO 210
240 N=N+1:IF N<LN THEN 210
250 N=1:IF LN=1 THEN 400
260 IF O(N)=1 THEN V(N)=V(N)+V(N+1):GOSU
B 300:N=1:GOTO 250
270 IF O(N)=2 THEN V(N)=V(N)-V(N+1):GOSU
B 300:N=1:GOTO 250
```

```
280 N=N+1:IF N<LN THEN 250
290 N=1:GOTO 210
300 FOR M=N+1 TO LN-1
310 V(M)=V(M+1):O(M-1)=O(M)
320 NEXT:LN=LN-1:RETURN
```

One by one, the results are accumulated into the first Value, V(1), and the number of values to be processed is reduced—in computer talk we call it decremented—by one, until there's only one left: that's the answer.

The answer is displayed next:

```
400 FOR P=1 TO 32-LA:PRINT "[CR]";:NEXT
410 IF MO=1 THEN 500
420 OA=V(1):PRINT "=";OA:GOTO 10
500 IF QA=V(1) THEN PRINT "YES":GOTO 10
510 PRINT "NO":GOTO 10
```

The graphics loop in line 400 backs the cursor up to the right place for the answer. If we want to see an answer, line 420 prints it. If we just want to know if we are right, lines 500 and 510 tell us. Then the program returns for another problem.

## ADVANCED TOPICS

There is lots of room to add the <UP-ARROW> and exponentiation to our calculator. Two other useful abilities would be: processing ((paren)theses) and multiple memories. I leave this challenge to the advanced programmers.

If you start getting good using it, you can graduate the computer's own "immediate mode" calculator, wherein you type problems in like this:

```
? 2 + 3 <RETURN>
? ((32.15/48.65)/12.5+3.2)+(121.9*.03)
```

# ANYBASE

## a counting machine

When they taught most of us to count, they forgot to mention that the decimal system is only one of a whole galaxy of perfectly lovely counting systems. We count with ten fingers, but computers like the Commodore, for instance, count with eight hands, each containing only one finger! And in Base 12, the Baker's Base, 10 is evenly divisible by 2, 3, 4, and 6!

ANYBASE contains two identical and interlocking counting machines. You can tell one to run as if it has ten fingers— the decimal system we use—and the second that it has only one finger—like the binary system that computers use, and watch the counters work together.

NOTE: If you have no notion why anyone would want to do *that,* you are not required to read any more of this chapter. On the other hand, if you know *exactly* what I'm talking about, skip a page. On the *third* hand, if you are interested (or not quite sure) read on:

As I was saying: on the planet, ZamoGram, all trade is conducted in base 4. You count like this:

| *Earth* | *ZamoGram* |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |

So far, so good. But now we get into trouble.

| | |
|---|---|
| 4 | 10 |
| 5 | 11 |
| 6 | 12 |
| 7 | 13 |
| 8 | 20 |

Are you with me? The next number after 33 would be 100, right?

If your business requires frequent translations between the ZamoGrams and the Twist-tytes (who use base 35—you, for example, my friend, might weigh between 1J and 2Z pounds), you, I say, NEED THIS PROGRAM!

## WRITING THE PROGRAM

This program is exceptionally modular: two of its modules are nearly identical. But first, the "human interface":

```
1 REM                                  * ANYBASE *
10 INPUT "      NUMBER";Q$
20 INPUT "    ITS BASE";B2
30 INPUT "TARGET BASE";B1
70 PRINT "[CD][CD]";:GOSUB 300:GOSUB 200
90 IF C$=Q$ THEN PRINT "[CD]":RUN
100 GOTO 70
```

Line 10 gets your desired number as a string of characters, Q$, while lines 20 and 30 get the number-base of your desired number, and the base to translate into. The graphics blocks in line 70 are <SHIFT><CURSOR-UP>s, which position the cursor opposite the desired number's base to display the count as it progresses; then the line sends out for the two counters. If the first counter has reached the target number, the counter stops; otherwise, it loops back to line 70.

The counters look like this:

```
200 N=0
210 N(N)=N(N)+1:IF N(N)<B1 THEN 230
220 N(N)=0:N=N+1:GOTO 210
230 IF N>HN THEN HN=N
240 PRINT TAB(20);">";:FOR D=HN TO 0 STE
P -1
250 IF N(D)<10 THEN D$=STR$(N(D)):GOTO 2
70
260 D$=" "+CHR$(55+N(D))
270 PRINT D$;:NEXT
290 PRINT:RETURN
300 M=0
```

```
310 M(M)=M(M)+1:IF M(M)<B2 THEN 330
320 M(M)=0:M=M+1:GOTO 310
330 IF M>HM THEN HM=M
340 PRINT TAB(20);">";:C$="":FOR D=HM TO
 0 STEP -1
350 IF M(D)<10 THEN D$=STR$(M(D)):GOTO 3
70
360 D$=" "+CHR$(55+M(D))
370 PRINT D$;:C$=C$+RIGHT$(D$,1):NEXT
390 PRINT:RETURN
```

Line 200 (and 310—the two are parallel) starts—the fancy computer term is *initializes*—the place counter. Line 210 checks the current place to see if adding one to it will be less than the number-base. (The basic rule of number-bases is that there is no numeral Z in base Z.) If not, the program "falls through" to line 220, which sets the current digit to 0, shifts its attention one place higher, and takes the unexpected step of looping back on itself: recursion in a BASIC program.

If the incremented number will be less than the number-base—or when the recursive program finally resolves itself, finding or creating a place it can increment—the string of digits representing the number is adjusted and displayed by lines 240 through 270. The counter then returns to the command immediately following where it was called.

## *Commodore 64 notes*

Duplicating Program Lines Revisited

ANYBASE provides a made-to-order opportunity to perfect your line-duplicating skills. If you type in lines 200 to 290, you have done 90% of the work.

If you just finished typing lines 200 to 290, they are displayed on the screen before you, and entered in program memory, as well. If you doubt (or if you have a scrambled screen) LIST 200-290.

Move the cursor with the <SHIFT><CURSOR-UP> key until you are on line 200. Press the <3> key to change the line number, then <CURSOR-RIGHT> over until you are on top of the N and overstrike it with an <M>. Now press <RETURN>. With a few keystrokes, you have modified and borrowed the line, with efficiency and a much-reduced chance of

typing errors. This feature is called "full-screen editing," and it certainly makes the programmer's task easier. A tool worth mastering.

Repeat the process, making the necessary changes and accepting each corrected line with a <RETURN> as you go. Remember, if you accept a line, then notice it needs another change, you can always move your cursor back into the line, fix it, and reaccept it. The most recent acceptance always prevails.

---

## ADVANCED SUBJECTS

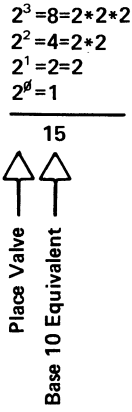
There are much more efficient interbase translation algorithms. The most elegant is based on the fact that exponents of the number bases come into play here. The decimal number 3179, we all agree, is made up of

```
   9 * 1 =     9(9 units)
  7 * 10 =    70(7 tens)
 1 * 100 =   100(1 hundred)
3 * 1000 =  3000(3 thousands)
          +
          ------
add up to  3179
```

What you may not know is that exponents are at play here:

$1000 = 10^3 = 10*10*10$
$100 = 10^2 = 10*10$
$10 = 10^1 = 10$
$1 = 10^0 = 1$ ← All Numbers to the Zero$^{th}$ Power = 1

The same thing happens in binary—base 2. The binary number 1111 can be evaluated like this



# UTILITIES

## for school and work

By now you know that computers can help with your work. This chapter helps you develop programs that can help you organize the events and data in your life. When you have finished these final pages, you can consider yourself an intermediate-grade BASIC-language computer programmer—with a fist full of modules to use on a variety of problems, and enough mastery over your machine to take on any problem it can solve.

DATEBOOK and ORGANIZE are simple data-base management systems. Both work with data that comes in three pieces—in DATEBOOK, these fields are the *date,* the *event,* and a *note* about it:

```
10/31   HALLOWEEN       PARTY @ CHAD'S
11/04   PAPER           ENGLISH-CHAUCER
11/10   TEST            SCIENCE
11/11   HOLIDAY         BAKERY
```

Each of these data pieces is called a *field,* and together form a *record.* These two programs let you reorganize data by any one, two, or three fields, so that you can view your data in the best possible order. For me, these two programs are the most useful in the book.

## DATEBOOK

The computer can be a willing and accurate helper with its powerful memory and easy-to-program study aids. The programs in this section are more complicated; more useful, too.

DATEBOOK helps you prepare for the inevitable: if you type in your homework assignments (and your parties)—date, event and description—this program helps you track them, so you need never be unprepared.

DATEBOOK is menu-driven: at any point your options are displayed for you. They are:

```
                1: ENTER DATA
2: SORT          6: PRINT
3: DISPLAY       7: MODIFY DATA
4: SAVE          8: CLEAR
5: LOAD          9: QUIT
```

If you select option 1, you are prompted (sample responses in italics):

DATE/TIME:*12/25*
EVENT:*Christmas*
NOTE:*special dinner in Santiago*

If you select option 2, SORT, you will be asked if you wish to sort by date, event, or note—you can get three different sortings. In much the same manner, if you ask for a display or a print-out of your calendar, you will be asked for the field number (DATE is field 1, NOTE is field 3) and the text you wish to match. If you had entered all your paper assignments through the end of the semester, for example, along with parties, birthdays, games, and other important dates, you could pop a list of due dates in order to plan your time.

## WRITING THE PROGRAM

Those who have already programmed SSORT recognized the Menu display, and you are right: you won't have to work very hard to get DATEBOOK running.

(For those of you who want a blow-by-blow explanation of what

the program does, the exciting part is back in the section on SORT.)

Initialization and Menu:

```
1 REM                              * DATEBOOK *
2 C$=CHR$(147):M$="                              "
5 DIM A$(2,100)
6 N=1:PRINT C$
10 PRINT M$;"* DATEBOOK *"
11 PRINT "      1: ENTER DATA"
12 PRINT "2: SORT          6: PRINT"
13 PRINT "3: DISPLAY       7: MODIFY DATA"
14 PRINT "4: SAVE          8: CLEAR"
15 PRINT "5: LOAD          9: QUIT"
20 PRINT M$;:INPUT "    OPTION";Q
30 ON Q GOTO 100,200,300,400,500,600,700
,890,900
```

Data entry:

```
100 PRINT "ENTER DATA       ZZZ TO QUIT"
110 PRINT "DATE/TIME:";:GOSUB 1000:PRINT
120 IF T$="ZZZ" THEN ND=N-1:GOTO 10
130 A$(0,N)=T$:PRINT "     EVENT:";:GOSUB
 1000:PRINT
140 A$(1,N)=T$:PRINT "      NOTE:";:GOSUB
 1000:PRINT
190 A$(2,N)=T$:N=N+1:GOTO 110
```

Sort module:

```
200 INPUT "SORT ON 1=DATE 2=EVENT 3=NOTE
";SK:PRINT "SORTING":FOR N=1 TO ND:P=N+1
220 IF A$(SK-1,N)<A$(SK-1,P) THEN 280
250 FOR SN=0 TO 2:T$(SN)=A$(SN,N):A$(SN,
N)=A$(SN,P):A$(SN,P)=T$(SN):NEXT
```

```
280 P=P+1:IFP<=ND THEN 220
290 NEXT:FOR SN=0 TO 2:A$(SN,ND)=T$(SN):
NEXT:PRINT "= = = = DONE = = = ="
```

Display:

```
300 INPUT "SELECT: FIELD NUMBER,KEY";SK$
,K$
310 IF SK$="" OR K$="" THEN SK=0:GOTO 33
0
320 SK=VAL(SK$):LT=LEN(K$)
330 FOR N=1 TO ND:IF SK=0 THEN 380
350 IF K$=LEFT$(A$(SK-1,N),LT) THEN 380
375 GOTO 390
380 PRINT A$(0,N);TAB(16);A$(1,N):PRINT
TAB(10);A$(2,N)
390 NEXT:INPUT "MENU";Q$:GOTO 10
```

Save and Load:

```
400 INPUT "FILENAME FOR SAVE";F$
410 F$=F$+",SEQ,W":OPEN 2,8,4,F$
420 FOR N=1 TO ND:IF A$(0,N)="" THEN 440
430 PRINT#2,A$(0,N);",";A$(1,N);",";A$(2
,N)
440 NEXT:PRINT#2,"ZZZ,,":CLOSE2
490 GOTO 10
500 INPUT "FILENAME TO LOAD";F$
510 F$=F$+",SEQ,R":OPEN 2,8,4,F$
530 INPUT#2,A$(0,N),A$(1,N),A$(2,N):IF A
$(0,N)="ZZZ" THEN ND=N-1:CLOSE2:GOTO 10
540 N=N+1:GOTO 530
```

**Print module:**

```
600 INPUT "PRINT SELECT: FIELD NUMBER,KE
Y";SK$,K$
610 IF SK$="" OR K$="" THEN SK=0
620 INPUT "LOWER CASE";L$:IF L$="Y" THEN
 635
630 OPEN3,4:GOTO 640
635 OPEN3,4,7
640 FOR N=1 TO ND:IF SK=0 THEN 680
650 IF K$<>LEFT$(A$(SK-1,N),LT) THEN 690
680 PRINT#3,CHR$(16);"08";A$(0,N);CHR$(1
6);"24";A$(1,N);CHR$(16);"40";A$(2,N)
690 NEXT:CLOSE3:GOTO 10

READY.
```

**Reschedule:**

```
700 SK$="":INPUT "RESCHED: FIELD NUMBER,
KEY";SK$,K$:IF SK$="" THEN 10
710 SK=VAL(SK$):LT=LEN(K$):N=1
730 IF K$=LEFT$(A$(SK-1,N),LT) THEN 780
740 GOTO 760
760 N=N+1:IF N<=ND THEN 730
770 PRINT K$;" NOT FOUND":GOTO 700
780 PRINT A$(0,N);TAB(16);A$(1,N):PRINT
TAB(10);A$(2,N)
790 SK$="":INPUT "RESCHED 1=DATE 2=EVENT
 3=NOTE";SK$:IF SK$="" THEN 760
800 SK=VAL(SK$):ON SK GOTO 810,820,830
810 PRINT "DATE/TIME:";:GOSUB 1000:PRINT
:A$(0,N)=T$:GOTO 790
820 PRINT "    EVENT:";:GOSUB 1000:PRINT
:A$(1,N)=T$:GOTO 790
830 PRINT "     NOTE:";:GOSUB 1000:PRINT
:A$(2,N)=T$:GOTO 790
```

Clear, End, and the Keyboard:

```
890 FOR N=1 TO ND:FOR SK=0 TO 2:A$(SK,N)
="":NEXT:N=1:ND=0:GOTO 10
900 PRINT "DONE":END
1000 T$="":PRINT TAB(5);";";
1010 GET I$:IF I$="" THEN 1010
1020 IF I$=CHR$(13) THEN RETURN
1030 IF I$=CHR$(20) THEN 1100
1090 PRINT "[CR]";I$;";";:T$=T$+I$:GOTO
1010
1100 LT=LEN(T$):IF LT=0 THEN 1010
1110 T$=LEFT$(T$,LT-1):PRINT "[CR] [CR][
CR];";:GOTO 1010
```

There is a new feature, added to the last module of this program: if you make a mistake, you can backspace and *fix* it—inaccurate typists like me will find that a thoughtful feature. If you like it, you may wish to retrofit SSORT with it, too. It happens in the change to line 1030 and the two new lines, 1100 and 1110.

## USING DATEBOOK

There are some tricks that may help you use DATEBOOK more effectively. You will notice that sorting sometimes give unexpected results. For example,

```
9/30
```

the usual abbreviation for September 30th, will sort *after*

```
10/13
```

because the computer thinks 10/ is less than 9/3—and it's right. (Start counting at the leftmost character: 1 is less than 9, isn't it?)

You can avoid that problem by using a leading 0, like

```
09/30
```

Because the sorting routine compares items from left to right, use the words you are most likely to sort by as the first characters in an entry.

Erase the record from the calendar by rescheduling the event's DATA to a blank—press <RETURN> only. You can reschedule an event by changing the DATE, then resorting the data.

You can load lists onto the ends of lists, and save them as one giant list simply by loading more than one file.

Always remember: on a Commodore, you can't save a file which already exists on disk. Keep track of the current calendar file name.

## ORGANIZE: TO DO FOR DATA WHAT YOU DID FOR DATES

ORGANIZE lets you work with other kinds of facts the way DATEBOOK keeps your calendar: it's a special-purpose data base manager to customize for any kind of need you can imagine.

## WRITING THE PROGRAM

Upgrading DATEBOOK takes a few easy changes.

Change the program name in line 1 and 10 (although this is optional, it will cut down on confusion later.)

```
1 REM                                * ORGANIZE *
2 C$=CHR$(147):M$="                              "
5 DIM A$(2,100)
6 N=1:PRINT C$
10 PRINT M$;"* ORGANIZE *"
```

Change the prompts in lines 110, 130, 140, 200, 790, 820, 830, and 840:

```
110 PRINT:PRINT "NAME:";:GOSUB 1000:PRIN
T
120 IF T$="ZZZ" THEN NO=N-1:GOTO 10
130 A$(0,N)=T$:PRINT "NOTE:";:GOSUB 1000
:PRINT
140 A$(1,N)=T$:PRINT "DATA:";:GOSUB 1000
:PRINT
```

```
200 INPUT "SORT ON 1=NAME 2=NOTE 3=DATA"
;SK:PRINT "SORTING":FOR N=1 TO ND:P=N+1

790 SK$="":INPUT "CHANGE  1=NAME 2=NOTE
3=DATA";SK$:IF SK$="" THEN 760
800 SK=VAL(SK$):ON SK GOTO 810,820,830
810 PRINT "NAME:";:GOSUB 1000:PRINT:A$(0
,N)=T$:GOTO 790
820 PRINT "NOTE:";:GOSUB 1000:PRINT:A$(1
,N)=T$:GOTO 790
830 PRINT "DATA:";:GOSUB 1000:PRINT:A$(2
,N)=T$:GOTO 790
```

You are finished: ORGANIZE is ready for testing and use.

I was curious about our presidents during the first 100 years of our country's history, so I typed them in chronologically:

```
Washington          1789-1797          Virginia
Adams (John)        1797-1801          Massachusetts
Jefferson           1801-1809          Virginia
Madison             1809-1817          Virginia
Monroe              1817-1825          Virginia
Adams (J.Q.)        1825-1829          Massachusetts
Jackson             1829-1837          South Carolina
Van Buren           1837-1841          New York
Harrison (Wm)       1841               Virginia
Tyler               1841-1845          Virginia
Polk                1845-1849          North Carolina
Taylor              1849-1850          Virginia
Fillmore            1850-1853          New York
Pierce              1853-1857          New Hampshire
Buchanan            1857-1861          Pennsylvania
Lincoln             1861-1865          Kentucky
Johnson (A.)        1865-1869          North Carolina
Grant               1869-1877          Ohio
Hayes               1877-1881          Ohio
Garfield            1881               Ohio
Arthur              1881-1885          Vermont
Cleveland           1885-1889          New Jersey
```

After putting all the presidents into the computer, it was easy to get a chronological list of the seven (!) presidents born in Virginia. Want an alphabetical listing? Sort by NAME first. Sorting by NAME, then by DATA yields a list sorted by state of birth that begins like this:

```
Lincoln          1861-1865      Kentucky
Adams (J.Q.)     1825-1829      Massachusetts
Adams (John)     1797-1801      Massachusetts
Pierce           1853-1857      New Hampshire
Cleveland        1885-1889      New Jersey
Fillmore         1850-1853      New York
Van Buren        1837-1841      New York
Johnson (A.)     1865-1869      North Carolina
Polk             1845-1849      North Carolina
```

## HEADER

You can plug HEADER in to a program that you are using for school or work to put a name, date, and title on every page, like this:

Michael Potts
October 14th
Heading Program

## WRITING THE PROGRAM

There isn't much to it (but it makes your work look good):

```
6000 open3,4,7:gosub 6010:goto 6020
6010 print#3,chr$(16);"55";:return
6020 print#3,"Michael Potts":gosub 6010
6030 input "today's date";d$:print#3,d$:
gosub 6010
6040 input "title";t$:print#3,t$
6090 print#3:print#3:close3:return
```

You insert it right at the beginning of a print routine. A bit of attention to your printer—always start typing at the same place on a clean sheet—and you have nearly professional output.

## CONCLUSION

Although this is the end of the book, it's only the beginning for a programmer. Not everyone is cut out to be a BASIC programmer—and BASIC is not the only language available. BASIC is a fine language for the kinds of programs in this book, and for business applications (like mailing labels and writing checks). Other languages have other strengths—PILOT is excellent for writing dialogues between human and computer; FORTH is perfect for controlling real world devices like thermocouples and motors; LOGO is specially designed for helping discover the structure and power of the computer. For those who feel particularly friendly with their computers after working though this book, machine language and assembly language offer the most direct possible access to the computer's power. And new languages appear with dizzying frequency.

Programmers must plan to become multilingual, and must learn to work with many different machines. The computer field is volatile almost beyond belief—new machines, languages, and capabilities are offered on a weekly basis, while older machines fall by the wayside. The Commodore is an excellent entry-level tool into the field. Patient urging from the programmer can turn it into a superlative computing tool—there is little the Commodore 64 *can't* do.

If you have enjoyed this book, and especially, if you have found unusual uses for some of the programs in this book, I would be glad to hear from you:

Michael Potts
The Caspar Institute
Post Office Box 88
Caspar, CA 95420

# appendices

# other tools

Programming is one activity that uses the computer usefully, but there are other interesting things to do.

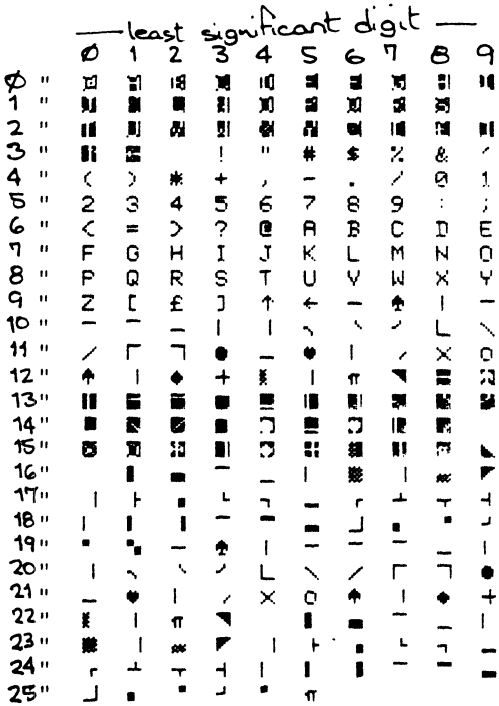
Word processing turns your computer into a very smart typewriter. It can help with organizing and writing papers, letters, and other written work. You could write or buy utilities that make your computer even smarter for writing: spelling checkers, table-of-contents generators, fancy-printing programs. The limitation is your time (or budget) and imagination. *Quick Brown Fox* is a word processor for the Commodore. CP/M programs are the most widely used word-processing packages in the world, with something to fit anyone who writes.

Spread-sheet programs turn your computer into an excellent accountant's helper—a sort of super calculating machine. With a spreadsheet program loaded into your computer, you can do elaborate simulations and "What-if" evaluations, or simply balance your personal finances. *Visi-Calc* is such a program.

You need keyboard skills to use a computer well—knowing which finger hits which key. Good computerized touch-typing instruction courses and games exist that will make your dialogue with the computer much more efficient and enjoyable.

The outside world is available to your computer, and with a modem and communications software you can take control of some of the most powerful computers in the world. For more information on this, an excellent resource is the book *The Complete Handbook of Personal Computer Communications* (Alfred Glossbrenner, St. Martin's Press, New York, 1983.) *The Source, CompuServe,* and *Dialog* are three of the most interesting computers for your computer to call.

## Printer




On Screen
NORMAL
GRAPHICS

reverse upper case

141 - 148
cursor control

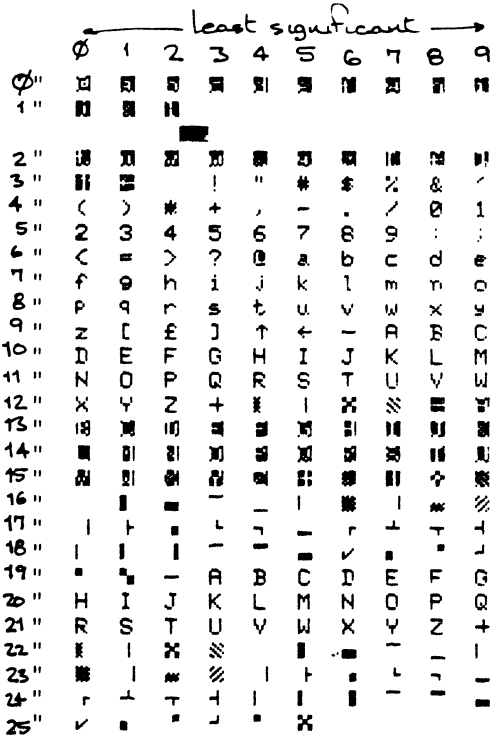
```
READY.

100 OPEN3,4
110 R=0:C=0
120 PRINT#3,CHR$(34);
130 IF R=25 AND C=6 THEN 300
140 IF R=1 AND C=3 THEN PRINT " HM";:GOTO 200
150 IF R=14 AND C=3 THEN PRINT " CH";:GOTO 200
180 PRINT#3,"  ";CHR$(10*R+C);
200 C=C+1:IF C<10 THEN 130
210 C=0:PRINT#3:R=R+1:GOTO 120
300 PRINT#3

READY.
```

# Printer

On Screen
(shift) C= mode




reverse lower case
13 homes cursor
20 missing

141 - 148
cursor control etc.

form generation
characters

```
ready.
```

```
100 open3,4,7
110 r=0:c=0
120 Print#3,chr$(34);
130 if r=25 and c=6 then 300
180 Print#3,"  ";chr$(10*r+c);
200 c=c+1:if c<10 then 130
210 c=0:Print#3:r=r+1:goto 120
300 Print#3:close3
```

```
ready.
```

| | | |
|---|---|---|
| POKE 53280,X | Border color | 0 through 15 |
| POKE 53281,X | Background color | 0 through 15 |
| POKE 646,X | Character color | 0 through 15 |

Also, [CTRL] and [CMDR] plus the number keys 1 through 8 may also be used to select colors.

| COLOR | CTRL | ASC | POKE | SYMBL | COLOR | CBM | ASC | POKE | SYMBL |
|---|---|---|---|---|---|---|---|---|---|
| BLACK | 1 | (144) | 0 | | ORANGE | 1 | (129) | 8 | |
| WHITE | 2 | (5) | 1 | | BROWN | 2 | (149) | 9 | |
| RED | 3 | (28) | 2 | | LIGHT RED | 3 | (150) | 10 | |
| CYAN | 4 | (159) | 3 | | GRAY 1 | 4 | (151) | 11 | |
| PURPLE | 5 | (156) | 4 | | GRAY 2 | 5 | (152) | 12 | |
| GREEN | 6 | (30) | 5 | | LIGHT GREEN | 6 | (153) | 13 | |
| BLUE | 7 | (31) | 6 | | LIGHT BLUE | 7 | (154) | 14 | |
| YELLOW | 8 | (158) | 7 | | GRAY 3 | 8 | (155) | 15 | |

| KEY | ASC | SYMBOL | KEY | ASC | SYMBOL |
|---|---|---|---|---|---|
| HOME | (19) | | FN1 | (133) | |
| CLR | (147) | | FN2 | (137) | |
| CRSR DOWN | (17) | CRSR ⇓ | FN3 | (134) | |
| CRSR UP | (145) | ⇑ CRSR | FN4 | (138) | |
| CRSR RIGHT | (29) | CRSR ⇒ | FN5 | (135) | |
| CRSR LEFT | (157) | ⇐ CRSR | FN6 | (139) | |
| RVS ON | (18) | RVS/ON | FN7 | (136) | |
| RVS OFF | (146) | RVS/OFF | FN8 | (140) | |

Each function key has an ASCII value and a unique graphic symbol that may be used in program control.

## Color, Editing, and Function Key Codes

# SUPERFINDME

```
1 REM                                * SFINDME *
2 C$=CHR$(147):VB=1024:LN=0:GOSUB 2000
10 PRINT C$;"      FIND ME!":PRINT " Y"
20 FOR R=8 TO 0 STEP -1:PRINT R;
30 FOR C=0 TO 9:PRINT "+  ";:NEXT C:PRIN
T:PRINT
40 NEXT R:PRINT "  ";:FOR C=0 TO 9:PRINT
 C;:NEXT:PRINT "X"
50 HN=9:GOSUB 1300:X=R:HN=8:GOSUB 1300:Y
=R
60 N1=X+1:N2=Y+1:GOSUB 2100
70 PRINT:PRINT "CAN YOU FIND ME (X,Y)";
80 INPUT QX,QY:IF QX=X AND QY=Y THEN 100
90 GOTO 200
100 PRINT "YOU FOUND ME!";
110 N1=X+1:N2=N1:GOSUB 2100:N1=Y+1:N2=N1
:GOSUB 2100
120 N1=3:N2=5:GOSUB 2100:GOSUB 2100
130 N1=7:N2=9:GOSUB 2100
170 QX=0:QY=0
180 FOR T=1 TO 500:NEXT:GOTO 10
200 PRINT "YOU MISSED ME";
210 N1=QX+1:N2=X+1:GOSUB 2100
220 N1=QY+1:N2=Y+1:GOSUB 2100
230 FOR N=1 TO 13:PRINT "[CR] [CR]";:NEX
T
240 PRINT "[CD][CD]";:GOTO 70
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

```
2000 S=54272:FOR L=S TO S+24:POKE L,0:NE
XT:K=1
2010 POKE S+5,9:POKE S+6,0:POKE S+24,15
2030 READ HF(K):READ LF(K):K=K+1:IF K<11
 THEN 2030:RETURN
2050 DATA 4,48,4,180,5,71,6,71,7,12
2060 DATA 8,97,9,104,10,143,12,143,14,24
2100 POKE S+1,HF(N1):POKE S,LF(N1):POKE
S+4,33
2110 FOR T=1 TO 125:NEXT
2120 POKE S+4,32:FOR T=1 TO 32:NEXT
2130 POKE S+1,HF(N2):POKE S,LF(N2):POKE
S+4,33
2140 FOR T=1 TO 250:NEXT
2150 POKE S+4,32:FOR T=1 TO 32:NEXT
2160 FOR L=S TO S+4:POKE L,0:NEXT
2190 RETURN
```

# ONECLAP

```
1 REM                                   * ONECLAP *
2 GOSUB 900:M$="                        ":K$=M$+"
  "
5 DIM A$(4,100):MO$="ANY LETTER":MO=0
10 PRINT:PRINT "HELLO":PRINT "MY NAME IS
 ONECLAP.":PRINT
20 INPUT "WHAT IS YOUR NAME";N$:PRINT:N=
0:GOTO 3000
30 GOSUB 300:IF LEN(T$)>1 THEN 60
40 GOSUB 400:K=K+1:IF K<10 THEN 30
50 MO$="ONECLAP":MO=1:GOSUB 300
60 F$=MID$(T$,LEN(T$)-1,1):GOSUB 500:W=0
:IF F=-1 THEN 250
70 IF A$(F,W)=T$ THEN 100
80 W=W+1:IF W<W(F)+1 THEN 70
90 GOTO 250
100 PRINT "[CD][CD][CD]":PRINT K$;T$;" I
S ON MY LIST.":GOSUB 800
110 NR=NR+1:PRINT:PRINT:IF T$="QUIT" THE
N 700
120 PRINT
130 PRINT M$;"[RV][RED] [GRN] [RED] [GRN
] [RED] [GRN] [RED] [GRN] [RED] [GRN] [R
ED] [GRN] [RED] "
140 PRINT M$;"[RV][GRN] [RV]            [
RV] "
150 PRINT M$;"[RV][RED] [RV]  Y E S !  [
RV][RED] "
```

```
160 PRINT M$;"[RV][GRN] [RV]              [
RV] "
170 PRINT M$;"[RV][RED] [GRN] [RED] [GRN
] [RED] [GRN] [RED] [GRN] [RED] [GRN] [R
ED] [GRN] [RED] "
180 PRINT:PRINT M$,"              , MORE
?";
190 GET Q$:IF Q$="" THEN 190
200 IF Q$="N" THEN 700
210 GOTO 50
220 B=B+1:IF B<11 THEN 290
230 GOTO 700
250 PRINT "[CD][CD][CD]":PRINT K$;T$;" I
S NOT ON MY LIST.":GOSUB 800
260 B$(B)=B$(B)+T$+" ":IF LEN(B$(B))<36
THEN 290
270 B=B+1:IF B<11 THEN 290
280 GOTO 700
290 GOTO 50
300 GOSUB 900:PRINT "[RV][RED]        "
310 PRINT K$;"[RV] [RV]      [RV] "
320 PRINT K$;"[RV] [RV]      [RV][RED] "
330 PRINT K$;"[RV] [RV]      [RV] "
340 PRINT K$;"[RV]        "
350 PRINT "[CR][CR][CR][CR][CR][CR][CR][
CR][CR][CR][CR][CR][CR][CR][CR][CR][CR][
CR][CR][CR][CR][CR][CD][CD][RV]";
360 GOSUB 1000
390 RETURN
400 F$=T$:GOSUB 500:IF F<0 THEN 420
410 HN=W(F):GOSUB 1300:W=R:R=F:GOTO 480
420 HN=4:GOSUB 1300:W=0
430 IF LEFT$(A$(R,W),1)=T$ THEN 480
440 IF RIGHT$(A$(R,W),1)=T$ THEN 480
450 W=W+1:IF W<W(R)+1 THEN 430
460 R=R+1:W=0:IF R<5 THEN 430
```

```
470 R=0:GOTO 430
480 T$=A$(R,W):PRINT "[CR] [CR]";T$
490 GOSUB 800:RETURN
500 F=-1:IF F$="A" THEN F=0
510 IF F$="E" THEN F=1
520 IF F$="I" THEN F=2
530 IF F$="O" THEN F=3
540 IF F$="U" THEN F=4
590 RETURN
700 PRINT C$:PRINT K$;"ONE CLAP":PRINT
710 PRINT N$;" FOUND";NR;"GOOD WORDS."
720 IF B$(0)="" THEN 790
730 PRINT:PRINT "WORDS NOT ON MY LIST"
740 FOR N=0 TO B-1:PRINT B$(N):NEXT
790 END
800 FOR T=0 TO 999:NEXT:RETURN
900 PRINT CHR$(147);"[RV]";N$;" "
910 FOR N=1 TO 3:PRINT:NEXT
920 PRINT K$;MO$:PRINT:PRINT K$;
990 RETURN
1000 T=0:T$="":REM                    * KEYB *
1010 GET I$:T=T+1:IF T<250 THEN 1060
1020 IF MO=0 AND LEN(T$)=1 THEN 1050
1030 IF MO=1 AND LEN(T$)=3 THEN 1050
1040 GOTO 1060
1050 PRINT "[CR][CR][CR][CR]PRESS <RETUR
N>[CR][CR][CR][CR][CR][CR][CR][CR][CR][C
R][CR][CR][CR][CR][CR][CR][CR][CR]";:T=0
1060 IF I$="" THEN 1010
1070 IF MO=1 AND LEN(T$)<3 THEN 1090
1080 IF I$=CHR$(13) THEN RETURN
1090 IF I$<>CHR$(13) THEN PRINT I$;:T$=T
$+I$
1100 T=0:GOTO 1010
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
3000 PRINT "EXCUSE ME, ";N$
```

```
3010 PRINT "I'M READING MY LIST NOW."
3020 PRINT:W=0
3030 GOSUB 3200:IF MID$(A$,2,1)="E" THEN
 3050
3040 A$(0,W)=A$:W=W+1:GOTO 3030
3050 W(0)=W-1:W=1::A$(1,0)=A$
3060 GOSUB 3200:IF MID$(A$,2,1)="I" THEN
 3080
3070 A$(1,W)=A$:W=W+1:GOTO 3060
3080 W(1)=W-1:W=1::A$(2,0)=A$
3090 GOSUB 3200:IF MID$(A$,2,1)="O" THEN
 3110
3100 A$(2,W)=A$:W=W+1:GOTO 3090
3110 W(2)=W-1:W=1::A$(3,0)=A$
3120 GOSUB 3200:IF MID$(A$,2,1)="U" THEN
 3140
3130 A$(3,W)=A$:W=W+1:GOTO 3120
3140 W(3)=W-1:W=1::A$(4,0)=A$
3150 GOSUB 3200:IF A$="ZZZ" THEN 3170
3160 A$(4,W)=A$:W=W+1:GOTO 3150
3170 W(4)=W-1
3190 GOTO 30
3200 READ A$:PRINT M$;A$:RETURN

9000 DATA BAT,CAT,EAT,FAT,HAT,MAT,PAT,RA
T,SAT,TAT,VAT,WAX
9010 DATA BET,GET,JET,LET,MET,NET,PET,SE
T,VET,YET
9020 DATA BIT,FIT,HIT,KIT,PIT,SIT,TIT,WI
T,ZIP,QUIT
9030 DATA COT,DOT,GOT,HOT,JOT,LOT,NOT,PO
T,ROT,SOT,TOT
9040 DATA BUT,CUT,GUT,HUT,JUT,NUT,PUT,RU
T
9990 DATA ZZZ
```

# PATTERN

```
1 REM                              * PATTERN *
5 LN=1
10 PRINT "[RED]QQQ   Q QQQQQQQQQQQQQQ Q
 Q  QQQ"
20 PRINT "[GRN]Q  Q Q Q  Q  Q Q   Q  QQQ
 Q Q"
30 PRINT "[RED]QQQ QQQQQ Q  Q QQQ QQQ Q
QQ  QQQ"
40 PRINT "[GRN]Q   Q   Q Q  Q Q   Q Q Q
 Q     Q"
50 PRINT "[RED]Q   Q   Q Q  Q QQQQQ  QQ
 Q QQQQ"
60 PRINT:INPUT "EASY (0) OR HARD (9)";DF
70 PRINT:HN=INT(DF/2)+1:IF HN>5 THEN HN=
5
80 GOSUB 1300:ON R GOTO 100,200,300,400,
500
100 HN=4*(DF+1):GOSUB 1300:S=R
120 HN=2*(DF+1):GOSUB 1300:I=R
130 HN=2:GOSUB 1300:L=3+R
140 HN=L:GOSUB 1300:M=R
150 FOR N=1 TO L
160 IF N=M THEN PRINT " ? ";:GOTO 180
170 PRINT S+I*(N-1);
180 NEXT:A=S+I*(M-1):GOSUB 1000:IF Q=A T
HEN 1100
190 GOTO 140
```

```
200 HN=4*SQR(DF+1):GOSUB 1300:S=R
220 HN=2*(DF+1):GOSUB 1300:I=R
225 HN=INT(DF/2):GOSUB 1300:I2=R
230 HN=2:GOSUB 1300:L=3+R
240 HN=L:GOSUB 1300:M=R
250 FOR N=1 TO L
260 IF N=M THEN PRINT " ? ";:GOTO 280
270 PRINT S*I*N;
280 NEXT:A=S*I*M:GOSUB 1000:IF Q=A THEN
1100
290 GOTO 240
300 HN=DF:GOSUB 1300:S=R
320 HN=DF+1:GOSUB 1300:I=R
325 HN=INT(DF/2):GOSUB 1300:I2=R
326 IF DF/2=INT(DF/2) THEN 330
327 I2=I2-HN:IF I2=0 THEN I2=1
330 HN=2:GOSUB 1300:L=3+R
340 HN=L:GOSUB 1300:M=R
350 I1=I:FOR N=1 TO L
360 IF N=M THEN PRINT " ? ";:A=S+I1*(N-1
):I1=I1+I2:GOTO 380
370 PRINT S+I1*(N-1);:I1=I1+I2
380 NEXT:GOSUB 1000:IF Q=A THEN 1100
390 GOTO 340
400 IF FR=0 THEN 1600
410 HN=FR:GOSUB 1300:A$=A$(R-1):P=1
420 L=LEN(A$):HN=L:GOSUB 1300:M=R
430 HN=2:GOSUB 1300:I=R:IF I=1 THEN 450
440 I=-1:P=L
450 IF P=M THEN PRINT " ? ";:Z$=MID$(A$,
P,1):GOTO 470
460 PRINT " ";MID$(A$,P,1);" ";
470 P=P+I:IF P=0 OR P>L THEN 490
480 GOTO 450
490 GOSUB 1000:IF Q$=Z$ THEN 1100
```

```
495 IF I<0 THEN 440
496 P=1:GOTO 450
500 HN=SQR(DF):GOSUB 1300:S=R
520 HN=2:GOSUB 1300:I=R+1
530 HN=2:GOSUB 1300:L=3+R
540 HN=L:GOSUB 1300:M=R
550 FOR N=1 TO L
560 IF N=M THEN PRINT " ? ";:A=S*M^I:GOT
O 580
570 PRINT S*N^I;
580 NEXT:GOSUB 1000:IF Q=A THEN 1100
590 GOTO 540
1000 PRINT
1010 INPUT "WHAT IS MISSING";Q$
1020 IF Q$="" THEN 1090
1030 IF ASC(Q$)>64 THEN 1090
1040 Q=VAL(Q$)
1090 RETURN
1100 PRINT "YOU ARE RIGHT."
1190 GOTO 70
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
1600 DIM A$(100):N=0
1610 READ A$(N):IF A$(N)="END" THEN 1690
1620 N=N+1:GOTO 1610
1690 FR=N:GOTO 410
9000 DATA BIRTHDAY,WASHINGTON,FISH
9010 DATA COMPUTER,HORSE,BOAT
9990 DATA "END"
```

# REMEMBER

```
1 REM                              * REMEMBER *
5 C$=CHR$(147):PRINT C$;"REMEMBER..."
6 M$="[CD][CD][CD][CD][CD][CD][CD][CD]
                "
9 TK=.01:SK=33
10 LN=0:PRINT:PRINT:PRINT
20 INPUT "0=EASY 9=HARD  ...HOW TOUGH";D
F:TD=200*(11-DF)
30 L=3+(DF/3):HN=9:IF DF>5 THEN HN=35
40 T$="":FOR N=1 TO L:GOSUB 1300
50 IF R>9 THEN 70
60 T$=T$+STR$(R):GOTO 80
70 T$=T$+" "+CHR$(54+R)
80 NEXT
100 PRINT C$;DF;M$;T$
110 FOR T=0 TO TD:NEXT
120 PRINT C$;M$;"?    ";:Q$="":T=0:GET I$
130 GET I$:T=T+1
150 IF I$="" THEN 130
170 IF I$=CHR$(13) THEN PRINT:GOTO 210
180 Q$=Q$+" "+I$:PRINT I$;" ";
190 IF Q$<>T$ THEN 130
200 PRINT:PRINT:PRINT"EXACTLY!"
210 IF Q$="" THEN 300
220 LT=LEN(T$):LQ=LEN(Q$):SC=0:FOR N=2 T
O LT STEP 2
230 IF N>LQ THEN 260
240 IF MID$(T$,N,1)=MID$(Q$,N,1) THEN SC
=SC+1
250 NEXT:PRINT "SYMBOLS RIGHT:";SC
```

```
260 SC=INT((SK*L)/((T*TK)*(SC/LT)))+1
280 PRINT "TIME:";T*TK;"SECONDS":PRINT "
SCORE";SC:TS=TS+SC:NQ=NQ+1:TT=TT+T*TK
290 FOR T=1 TO TD:NEXT:GOTO 40
300 PRINT C$;"REMEMBER...":PRINT
310 PRINT "TOTAL SCORE        ";TS
320 PRINT "STRINGS ATTEMPTED ";NQ
330 PRINT "AVERAGE SCORE/$    ";TS/NQ
340 PRINT "TOTAL SECONDS      ";TT
350 PRINT "AVERAGE  TIME/$    ";TT/NQ
390 FOR T=1 TO 3*TD:NEXT:GOTO 40
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

# SPELL

```
1 REM                                   * SPELL *
2 DIM A$(100),B$(100),C$(100)
5 C$=CHR$(147)+"[CD][CD][CD][CD][CD][CD]
[CD]":PRINT C$:M$="                    ":PRINT
 M$;"SPELL":PRINT
8 R$(1)="RIGHT!![CR][CR]":R$(2)="CORRECT
  ":R$(3)="YOU GOT IT!"
9 TD=1000:LN=1:LD=9
10 GOTO 500
20 FOR N=1 TO ND:A$=A$(N)
30 PRINT A$:B$(N)=LEFT$(A$,LD):LT=LEN(A$
)-LD:C$(N)=RIGHT$(A$,LT)
40 IF RIGHT$(B$(N),1)=" " THEN B$(N)=LEF
T$(B$(N),LEN(B$(N))-1):GOTO 40
50 NEXT:FOR T=0 TO TD:NEXT
60 HN=ND:GOSUB 1300:PRINT C$;M$;"  ";B$(
R)
70 FOR T=0 TO TD:NEXT:PRINT C$;M$;:INPUT
 Q$
80 IF Q$=B$(R) THEN 100
90 PRINT:PRINT M$;"SORRY, IT'S ";B$(R):W
R=WR+1:GOTO 130
100 HN=3:GOSUB 1300:PRINT:PRINT M$;R$(R)
110 RI=RI+1
130 FOR N=0 TO TD:NEXT:IF RI<10 THEN 60
140 IF RI>30 THEN 200
150 HN=ND:GOSUB 1300:PRINT C$;"   ";C$(R
)
160 PRINT:PRINT M$;:INPUT Q$:GOTO 80
```

```
200 WG=0:RG=0:HN=ND:GOSUB 1300:PRINTC$;M
$;
210 T$=B$(R):LT=LEN(T$):FOR Z=1 TO LT
220 PRINT "- ";:D$(Z)=MID$(B$(R),Z,1)
230 NEXT:PRINT:PRINT M$;"GUESS  ?";
240 GET I$:IF I$="" THEN 240
250 PRINT I$;:Z=1
260 IF I$=D$(Z) THEN 310
270 Z=Z+1:IF Z<=LT THEN 260
280 PRINT "[CR] [CR]";:IF RF=1 THEN RF=0
:GOTO 240
290 WG=WG+1:IF WG<2*LT THEN 240
300 PRINT:PRINT "YOU LOSE":GOTO 130
310 PRINT:PRINT "[CD][CD]";M$;:IF Z=1 TH
EN 330
320 FOR P=1 TO Z-1:PRINT "[CR][CR]";:NEX
T
330 PRINT D$(Z);:D$(Z)="":RG=RG+1:RF=1
340 IF RG=LT THEN PRINT:PRINT:GOTO 100
350 PRINT:PRINT M$;"[CR][CR][CR][CR][CR]
[CR][CR][CR][CR]";:GOTO 270
360 GOTO 240
499 END
500 INPUT "FILENAME TO LOAD";F$
510 F$=F$+",SEQ,R":OPEN 2,8,4,F$:N=1
530 INPUT#2,A$(N):IF A$(N)="ZZZ" THEN ND
=N-1:CLOSE2:GOTO 20
540 N=N+1:GOTO 530
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

# MATHFAX

```
1 REM                                  * MATHFAX *
5 C$=CHR$(147)+"[CD][CD][CD][CD][CD][CD]
[CD][CD]":M$="                    "
10 PRINT C$;M$;"* MATH FACTS *":LN=0:PRI
NT
20 INPUT "1=ADD 2=SUBTRACT 3=MULTIPLY 4=
DIVIDE";HO:HO=HO-1
30 INPUT "HIGHEST NUMBER FOR ADDING";HA:
IF HO<2 THEN 50
40 INPUT "            FOR MULTIPLYING";HM
50 HN=HO:GOSUB 1300:OP=R+1:PRINT C$;M$;:
ON OP GOTO 60,60,90,90
60 HN=HA:GOSUB 1300:N1=R:GOSUB 1300:N2=R
:ON OP GOTO 70,80
70 PRINT N1;"+";N2;"= ";:A=N1+N2:GOTO 13
0
80 PRINT N1+N2;"-";N1;"= ";:A=N2:GOTO 13
0
90 HN=HM:GOSUB 1300:N1=R:GOSUB 1300:N2=R
:ON OP GOTO 0,0,100,110
100 PRINT N1;"*";N2;"= ";:A=N1*N2:GOTO 1
30
110 IF N2=0 OR N2=0 THEN 90
120 PRINT N1*N2;"/";N1;"= ";:A=N2:GOTO 1
30
130 INPUT Q:IF Q<>A THEN 160
140 PRINT:PRINT M$;"         YES"
150 FOR T=0 TO 999:NEXT:GOTO 50
160 PRINT:PRINT M$;"         NO":ON OP GO
TO 170,165,180,180
```

```
165 NT=N1+N2:GOTO 175
170 NT=N2
175 PRINT M$;:GOSUB 200:PRINT M$;:NT=N1:
GOSUB 200:GOTO 150
180 N2=0
190 IF N2<N1 THEN PRINT M$;:NT=N2:GOSUB
200:N2=N2+1:GOTO 190
195 GOTO 150
200 NP=0
210 IF NP<NT THEN PRINT "Q";:NP=NP+1:GOT
O 210
220 PRINT:RETURN
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

# TIMELINE

```
1 dim n(100),n$(100):m$="                "
2 c$=chr$(147)+"           * TIMELINE *"
chr$(13):print c$:n=1
10 read n(n),n$(n):if n$(n)="end" then 3
0
20 n=n+1:goto 10
30 ln=n-1:for n=1 to ln:print m$;n(n):pr
int n$(n):print:next:y=3
40 print c$:for n=y-2 to y+2:print m$;"-
";n(n);"-":print n$(n):print:next
50 print "-  -  -  -  -  -  -  -  -  -"
60 print "  - earlier  + later   What ne
xt?";:gosub 1000
65 if q$="+" then y=y+4:goto 80
70 if q$="-" then 90
75 y=y+1
80 if y>ln-2 then y=ln-2
85 goto 40
90 y=y-4:if y<3 then y=3
95 goto 40
99 stop
100 data 1000,"Leif Ericson discovers Am
erica"
110 data 1522,"first circumnavigation of
 the Earth"
120 data 1607,"Jamestown settled in Virg
inia"
130 data 1620,"Pilgrims arrive at Plymou
th Rock"
```

```
140 data 1756,"French and Indian War beg
ins"
150 data 1775,"James Watt invents steam
engine"
160 data 1776,"Declaration of Independen
ce"
170 data 1778,"Captain Cook discovers Ha
waii"
180 data 1787,"U.S. Constitution signed"
190 data 1791,"Bill or Rights ratified"
200 data 1803,"Louisiana Purchase"
210 data 1807,"Robert Fulton's steamboat
 up the Hudson"
220 data 1834,"Charles Babbage's Analyti
cal Engine"
230 data 1846,"Potato Famine - U.S.-Mexi
co War"
240 data 1849,"California Gold Rush"
250 data 1858,"First Trans-Atlantic cabl
e"
260 data 1861,"Civil War begins"
270 data 1865,"Abraham Lincoln shot"
280 data 1869,"Golden spike: railroad ac
ross the U.S."
290 data 1888,"George Eastman invents th
e Kodak camera"
999 data 999,end
1000 get q$:if q$="" then 1000
1010 return
```

# MINIWOMBATS

```
1 rem"                          * miniWOMBATS *
5 c$=chr$(147)+"{CD}{CD}{CD}":m$="

8 poke 53272,23:rem: sets lower case on
9 sx=1:p$="Sienna":j$="wombat"
10 print c$;m$;"* math facts *":in=0:pri
nt
20 input "1=add 2=subtract 3=multiply 4=
divide";ho:ho=ho-1
30 input "highest number for adding";ha:
if ho<2 then 50
40 input "           for multiplying";hm
50 print c$
55 input "Please tell me your name";n$:i
f asc(n$)<192 then gosub 1120:goto 55
60 hn=3:gosub 1300:if r=3 then gosub 110
0
70 gosub 1300:if r=3 then gosub 1200
80 hn=ho:gosub 1300:op=r+1:if op>2 then
hn=hm:goto 90
85 hn=ha
90 gosub 1300:n1=r:gosub 1300:n2=r:on op
 goto 100,300,500,700
100 hn=2:gosub 1300:on r+1 goto 110,140,
170
110 q$=p$+" found a bag containing"+str$
(n1)+" "+j$+"s. ":gosub 1400
120 q$=q$+"already has"+str$(n2)+" "+j$+
"s at home. How many does ":gosub 1420
130 q$=q$+"have now":goto 290
```

```
140 q$=p$+" had"+str$(n1)+" "+j$+"s in "
:gosub 1480:q$=q$+"pocket, "
150 q$=q$+" and later won"+str$(n2)+" in
 a bet with "+n$+". How many did "
160 gosub 1420:q$=q$+"have then":goto 29
0
170 q$=p$+" receives 2 envelopes. One co
ntains"+str$(n1)+" "+j$+"s; the other"
180 q$=q$+" contains"+str$(n2)+". How ma
ny does ":gosub 1420
190 q$=q$+"have now":goto 290
290 a=n1+n2:goto 900
300 n2=n2+n1:hn=2:gosub 1300:on r+1 goto
 310,340,370
310 q$=p$+" hides"+str$(n2)+" "+j$+"s, a
nd you find"+str$(n1)+". "
320 q$=q$+"How many are still hidden":go
to 490
340 q$=p$+" had too many "+j$+"s and gav
e you"+str$(n2)+". Later, ":gosub 1420
350 q$=q$+"lost all of ":gosub 1440:q$=q
$+"and you gave"+str$(n1)
360 q$=q$+" back. How many do you still
have":goto 490
370 q$="Yesterday, "+p$+" bought"+str$(n
2)+" "+j$+"s, but this morning "
380 gosub 1420:q$=q$+"could only find"+s
tr$(n1)
390 q$=q$+". How many were missing":goto
 490
490 a=n2-n1:goto 900
500 hn=2:gosub 1300:on r+1 goto 510,540,
570
510 q$=p$+" won"+str$(n1)+" coupons at t
he fair. ":gosub 1400
```

```
520 q$=q$+"exchanged each coupon for"+st
r$(n2)+" "+j$+"s. How many does "
530 gosub 1420:q$=q$+"have now":goto 690
540 q$=p$+" built a machine to make "+j$
+"s. It has made"+str$(n1)
550 q$=q$+" each day for"+str$(n2)+" day
s. How many has it made":goto 690
560 print r$;:input q:if q=a then 600
570 q$="A flying saucer deposits"+str$(n
1)+" silvery spheres in "+p$
580 q$=q$+"'s back yard."+str$(n2)+" "+j
$+"s jumped out of each one. How many "
590 q$=q$+j$+"s are rampaging around "+p
$+"'s house":goto 690
690 a=n1*n2:goto 900
700 if n1=0 then hn=hm:gosub 1300:n1=r:g
oto 700
710 n2=n1*n2:hn=2:gosub 1300:on r+1 goto
 720,750,780
720 q$=p$+" dug"+str$(n1)+" "+j$+" traps
 in the forest, and caught a total of"
730 q$=q$+str$(n2)+" "+j$+"s. On the ave
rage, how many did ":gosub 1420
740 q$=q$+"find in each trap":goto 890
750 q$=p$+" buys a packet with"+str$(n2)
+" "+j$+" seeds in it. The directions"
760 q$=q$+" tell ":gosub 1460:q$=q$+"to
plant"+str$(n1)+" in each hole.
770 q$=q$+"How many holes should ":gosub
 1420:q$=q$+"dig":goto 890
780 q$=p$+" shared"+str$(n2)+" "+j$+"s w
ith"+str$(n1-1)+" of ":gosub 1480
790 q$=q$+"friends. How many did each ge
t":goto 890
890 a=n2/n1
```

```
900 t$=q$:print:print " = # = # = # = #
= # =":print
910 lt=len(t$):r$=left$(t$,39)
920 lr=len(r$):if lr=lt then 960
930 if right$(r$,1)=" " then 950
940 r$=left$(r$,lr-1):lr=lr-1:goto 930
950 t$=right$(t$,lt-lr):print r$:goto 91
0
960 print r$;"[CD][CD][CU][CU]";:input q
:if q=a then 1000
970 print:print "Woops! That's not right
."
980 print "The right answer is";a;"."
990 goto 60
1000 hn=3:gosub 1300:on r goto 1010,1020
,1030
1010 print "You got it!":goto 1090
1020 print "Well done.":goto 1090
1030 print "That's right!"
1090 goto 60
1100 print:print " -- -- -- -- -- -- --
-- --":print
1110 input "Name a person";p$:if asc(p$)
>192 then 1150
1115 gosub 1120:print:goto 1110
1120 print "Person's names begin with a
CAPITAL."
1130 print "Remember the SHIFT key, and"
:print "please try again, ";n$;"."
1140 return
1150 print "Is ";p$;" a boy or girl";
1160 sx=0:input sx$:if sx$="boy" then sx
=2
1170 if sx$="girl" then sx=1
1180 if sx>0 then return
```

```
1190 print "I'm sorry. I don't know any
questions":print "about ";sx$;"s."
1195 print "Please try again.":goto 1100
1200 print:print " ++ ++ ++ ++ ++ ++ ++
++ ++":print
1210 input "Name an object";j$
1290 return
1300 r=int(ln+(hn-ln+1)*rnd(1)):return
1400 if sx=1 then q$=q$+"She ":return
1410 q$=q$+"He ":return
1420 if sx=1 then q$=q$+"she ":return
1430 q$=q$+"he ":return
1440 if sx=1 then q$=q$+"hers ":return
1450 q$=q$+"his ":return
1460 if sx=1 then q$=q$+"her ":return
1470 q$=q$+"him ":return
1480 if sx=1 then q$=q$+"her ":return
1490 q$=q$+"his ":return
9990 open3,4,7:cmd3:list:print#3:close3
```

# PAINT

```
1 REM                                   * PAINT *
5 C$=CHR$(147):PRINT C$;:X=20:Y=12:A=83
6 VB=1064:CB=55336:CF=1
7 DIM B$(24),C$(24)
10 GOSUB 1100:GOSUB 1000
20 IF I=17 THEN Y=Y+1:IF Y>24 THEN Y=24
30 IF I=145 THEN Y=Y-1:IF Y<1 THEN Y=1
40 IF I=29 THEN X=X+1:IF X>39 THEN X=39
50 IF I=157 THEN X=X-1:IF X<0 THEN X=0
60 IF I=32 AND PF=1 THEN PF=0:GOTO 70
65 IF I=32 THEN PF=1
70 IF I>47 AND I<58 THEN PC=VAL(I$)-1:CF
=PC:GOSUB 1100
80 IF I$="Q" THEN END
90 IF I$="Z" THEN 800
100 IF I$="S" THEN 300
110 IF I$="L" THEN 500
120 IF I$="T" THEN 200
190 GOTO 10
200 GET I$:IF I$="" THEN 200
210 IF I$=CHR$(13) THEN A=83:GOTO 10
220 I=ASC(I$):IF I<64 THEN A=I:GOTO 240
230 A=ASC(I$)-64
240 GOSUB 1100:X=X+1:IFX>39 THEN X=39
290 GOTO 200
300 VP=VB:CP=CB:PRINT "CRUNCHING.  ":FO
R ZL=1 TO 24
310 B$(ZL)="":C$(ZL)="":FOR ZC=0 TO 39
320 B$=CHR$(PEEK(VP)):IF B$=" " THEN B$=
"@"
```

```
330 B$(ZL)=B$(ZL)+B$:VP=VP+1
340 CZ=PEEK(CP)
350 IF CZ>15 THEN CZ=CZ-16:GOTO 350
360 C$(ZL)=C$(ZL)+CHR$(CZ+65):CP=CP+1
370 NEXT
380 PRINT TAB(15);ZL;"[CR][CR][CR][CR][C
R]";
390 NEXT:PRINT:PRINT"[CD]";
400 INPUT "FILENAME FOR SAVE";F$
410 F$=F$+",SEQ,W":OPEN 2,8,4,F$
420 FOR ZL=1 TO 24
430 PRINT#2,B$(ZL):PRINT#2,C$(ZL)
440 NEXT:CLOSE2
450 PRINT "[CR]";:FOR N=1 TO 40:PRINT CH
R$(20);:NEXT
490 GOTO 10
500 INPUT "FILENAME TO LOAD";F$
510 F$=F$+",SEQ,R":OPEN 2,8,4,F$
520 PRINT CHR$(147);:FOR ZL=1 TO 24
530 INPUT#2,B$(ZL):INPUT#2,C$(ZL)
540 NEXT:CLOSE2
550 VP=VB:CP=CB
560 FOR ZL=1 TO 24:GOSUB 600:NEXT
570 X=20:Y=12:GOTO 10
600 FOR VC=1 TO 40
610 B$=MID$(B$(ZL),VC,1):IF B$="@" THEN
B$=" "
620 C$=MID$(C$(ZL),VC,1)
640 POKE CP,ASC(C$)-65
650 POKE VP,ASC(B$)
680 VP=VP+1:CP=CP+1
690 NEXT:RETURN
800 INPUT "ZAP";Q$:IF Q$="Y" THEN 820
810 PRINT "[CD]      ":PRINT "[CD]";:GOT
O 10
```

```
820 RUN
1000 REM +KEYB
1010 GET I$:IF I$="" THEN 1010
1020 I=ASC(I$)
1030 IF PF=0 THEN A=OS:CF=OC
1040 IF PF=1 THEN A=160:CF=PC
1050 GOSUB 1100:A=83:CF=PC
1090 RETURN
1100 P=VB+40*(Y-1)+X:OS=PEEK(P)
1110 POKE P,A
1120 C=CB+40*(Y-1)+X:OC=PEEK(C)
1130 IF CF>-1 THEN POKE C,CF
1190 RETURN
```

# BARS

```
1 REM                                    * BARS *
4 M$="                "
5 C$=CHR$(147):PRINT C$;M$;"BARS"
10 READ B:IF B=-1 THEN 30
20 B$(N)=CHR$(B):N=N+1:GOTO 10
30 INPUT"YOUR LOWEST NUMBER";LN
40 INPUT"     HIGHEST NUMBER";HN
50 INPUT"       HOW MANY BARS";NB
60 INPUT"          GRAPH TITLE";T$
70 FOR N=0 TO NB-1
80 PRINT "BAR #";N+1;TAB(13);:INPUT "TIT
LE";T$(N)
90 PRINT TAB(12);:INPUT "NUMBER";L(N):NE
XT
100 PRINT C$:PRINT M$;T$:PRINT
120 I=(HN-LN)/5:IB=36/(HN-LN)
130 FOR N=0 TO 5:PRINT INT(LN+(N*I));"
 ";:NEXT:PRINT
140 PRINT:FOR N=0 TO NB-1
150 PRINT " ";B$(N);"[RV]";
160 FOR S=LN TO INT(L(N)*IB)-1:PRINT " "
;:NEXT
170 PRINT "[RV]";L(N)
180 PRINT "    ";T$(N)
190 PRINT:NEXT
200 GOTO 200
900 DATA 28,5,30,156,158,144,159,-1
```

```
1 REM                                    * BARS *
4 M$="          "
5 C$=CHR$(147)
10 READ B:IF B=-1 THEN 30
20 B$(N)=CHR$(B):N=N+1:GOTO 10
30 READ LN
40 READ HN
50 READ NB
60 READ T$
70 FOR N=0 TO NB-1
80 READ T$(N)
90 READ L(N):NEXT
100 PRINT C$:PRINT M$;T$:PRINT
120 I=(HN-LN)/5:IB=36/(HN-LN)
130 FOR N=0 TO 5:PRINT INT(LN+(N*I));"
  ";:NEXT:PRINT
140 PRINT:FOR N=0 TO NB-1
150 PRINT " ";B$(N);"[RV]";
160 FOR S=LN TO INT(L(N)*IB)-1:PRINT " "
;:NEXT
170 PRINT "[RV]";L(N)
180 PRINT "    ";T$(N)
190 PRINT:NEXT
200 GOTO 200
900 DATA 28,5,30,156,158,144,159,-1
910 DATA 0,75,6
920 DATA LIFE EXPECTANCY - MALES
930 DATA USA,69
940 DATA COLOMBIA,44
950 DATA NORWAY,71
960 DATA JAPAN,71
970 DATA INDIA,42
980 DATA AUSTRALIA,68
```

# SORT

```
1 REM                                    * SORT *
2 C$=CHR$(147):M$="                      "
5 DIM A$(100)
6 N=1:PRINT C$
10 PRINT M$;"SORT"
11 PRINT "1: ENTER DATA"
12 PRINT "2: SORT"
13 PRINT "3: DISPLAY"
19 PRINT "9: QUIT"
20 INPUT "           OPTION";Q
30 ON Q GOTO 100,200,300,10,10,10,10,10,
900
100 PRINT "ENTER DATA      ZZZ TO QUIT"
110 PRINT N;":";:GOSUB 1000:PRINT
120 IF T$="ZZZ" THEN ND=N-1:GOTO 10
130 A$(N)=T$:N=N+1:GOTO 110
200 PRINT "SORTING"
210 FOR N=1 TO ND:P=N+1
220 PRINT A$(N),A$(P),:IF A$(N)<A$(P) TH
EN 240
230 T$=A$(N):A$(N)=A$(P):A$(P)=T$
235 PRINT "SWAPPING",
240 PRINT:P=P+1:IF P<=ND THEN 220
250 NEXT:A$(ND)=T$
280 PRINT "= = = = DONE = = = ="
300 FOR N=1 TO ND:PRINT N;TAB(5);A$(N):N
EXT
380 INPUT "MENU";Q$
390 GOTO 10
900 PRINT "DONE":END
```

```
1000 T$="":PRINT TAB(5);";";
1010 GET I$:IF I$="" THEN 1010
1020 IF I$=CHR$(13) THEN RETURN
1090 PRINT "[CR]";I$;";";:T$=T$+I$:GOTO
1010
```

# SUPERSORT

```
1 REM                                   * SSORT *
2 C$=CHR$(147):M$="                     "
5 DIM A$(100)
6 N=1:PRINT C$
10 PRINT M$;"* SORT *"
11 PRINT "1: ENTER DATA"
12 PRINT "2: SORT          6: PRINT"
13 PRINT "3: DISPLAY       7: MODIFY DATA"
14 PRINT "4: SAVE          8: CLEAR"
15 PRINT "5: LOAD          9: QUIT"
20 PRINT M$;:INPUT "            OPTION";Q
30 ON Q GOTO 100,200,300,400,500,600,700
,800,900
100 PRINT "ENTER DATA       ZZZ TO QUIT"
110 PRINT N;":";:GOSUB 1000:PRINT
120 IF T$="ZZZ" THEN ND=N-1:GOTO 10
130 A$(N)=T$:N=N+1:GOTO 110
200 PRINT "SORTING"
210 FOR N=1 TO ND:P=N+1
220 PRINT A$(N),A$(P),:IF A$(N)<A$(P) TH
EN 240
230 T$=A$(N):A$(N)=A$(P):A$(P)=T$
235 PRINT "SWAPPING",
240 PRINT:P=P+1:IF P<=ND THEN 220
250 NEXT:A$(ND)=T$
280 PRINT "= = = = DONE = = = ="
300 FOR N=1 TO ND:PRINT N;TAB(5);A$(N):N
EXT
380 INPUT "MENU";Q$
390 GOTO 10
```

```
400 INPUT "FILENAME FOR SAVE";F$
410 F$=F$+",SEQ,W":OPEN 2,8,4,F$
420 FOR N=1 TO ND
430 PRINT#2,A$(N)
440 NEXT:PRINT#2,"ZZZ":CLOSE2
490 GOTO 10
500 INPUT "FILENAME TO LOAD";F$
510 F$=F$+",SEQ,R":OPEN 2,8,4,F$
530 INPUT#2,A$(N):IF A$(N)="ZZZ" THEN ND
=N-1:CLOSE2:GOTO 10
540 N=N+1:GOTO 530
600 INPUT "PRINT LINE NUMBERS";Y$
610 INPUT "LOWER CASE";L$:IF L$="Y" THEN
 630
620 OPEN3,4:GOTO 650
630 OPEN3,4,7
650 FOR N=1 TO ND
660 IF Y$="N" THEN 680
670 PRINT#3,N;
680 PRINT#3,CHR$(16);"08";A$(N):NEXT
690 CLOSE3:GOTO 10
700 INPUT "LINE TO CORRECT";X:IF X=0 THE
N 10
710 PRINT X;TAB(5);A$(X)
720 PRINT TAB(5);:GOSUB 1000:PRINT
730 IF T$="ZZZ" THEN 10
790 A$(X)=T$:GOTO 700
800 FOR N=1 TO ND:A$(N)="":N=1:ND=0:GOTO
 10
900 PRINT "DONE":END
1000 T$="":PRINT TAB(5);";";
1010 GET I$:IF I$="" THEN 1010
1020 IF I$=CHR$(13) THEN RETURN
1090 PRINT "[CR]";I$;";";:T$=T$+I$:GOTO
1010
```

# CALC

```
1 REM                                    * CALC *
10 MO=0:INPUT "CALC";A$
20 LA=LEN(A$):N=1:P=1
30 I$=MID$(A$,P,1)
40 IF I$="+" THEN O(N)=1:GOTO 180
50 IF I$="-" THEN O(N)=2:GOTO 180
60 IF I$="*" THEN O(N)=3:GOTO 180
70 IF I$="/" THEN O(N)=4:GOTO 180
80 IF I$>="0" AND I$<="9" THEN V$=V$+I$:
GOTO 190
90 IF I$="." THEN V$=V$+".":GOTO 190
100 IF I$="A" THEN V$=STR$(OA):GOTO 190
110 IF I$="=" THEN MO=1:QA=VAL(RIGHT$(A$
,LA-P)):GOTO 200
180 V(N)=VAL(V$):V$="":N=N+1
190 P=P+1:IF P<=LA THEN 30
200 V(N)=VAL(V$):V$="":LN=N:N=1
210 IF LN=1 THEN 400
220 IF O(N)=3 THEN V(N)=V(N)*V(N+1):GOSU
B 300:N=1:GOTO 210
230 IF O(N)=4 THEN V(N)=V(N)/V(N+1):GOSU
B 300:N=1:GOTO 210
240 N=N+1:IF N<LN THEN 210
250 N=1:IF LN=1 THEN 400
260 IF O(N)=1 THEN V(N)=V(N)+V(N+1):GOSU
B 300:N=1:GOTO 250
270 IF O(N)=2 THEN V(N)=V(N)-V(N+1):GOSU
B 300:N=1:GOTO 250
280 N=N+1:IF N<LN THEN 250
290 N=1:GOTO 210
```

```
300 FOR M=N+1 TO LN-1
310 V(M)=V(M+1):O(M-1)=O(M)
320 NEXT:LN=LN-1:RETURN
400 FOR P=1 TO 32-LA:PRINT "[CR]";:NEXT
410 IF MO=1 THEN 500
420 OA=V(1):PRINT "=";OA:GOTO 10
500 IF QA=V(1) THEN PRINT "YES":GOTO 10
510 PRINT "NO":GOTO 10
```

# ANYBASE

```
1 REM                                * ANYBASE *
10 INPUT "      NUMBER";Q$
20 INPUT "    ITS BASE";B2
30 INPUT "TARGET BASE";B1
70 PRINT "[CD][CD]";:GOSUB 300:GOSUB 200
90 IF C$=Q$ THEN PRINT "[CD]":RUN
100 GOTO 70
200 N=0
210 N(N)=N(N)+1:IF N(N)<B1 THEN 230
220 N(N)=0:N=N+1:GOTO 210
230 IF N>HN THEN HN=N
240 PRINT TAB(20);">";:FOR D=HN TO 0 STE
P -1
250 IF N(D)<10 THEN D$=STR$(N(D)):GOTO 2
70
260 D$=" "+CHR$(55+N(D))
270 PRINT D$;:NEXT
290 PRINT:RETURN
300 M=0
310 M(M)=M(M)+1:IF M(M)<B2 THEN 330
320 M(M)=0:M=M+1:GOTO 310
330 IF M>HM THEN HM=M
340 PRINT TAB(20);">";:C$="":FOR D=HM TO
 0 STEP -1
350 IF M(D)<10 THEN D$=STR$(M(D)):GOTO 3
70
360 D$=" "+CHR$(55+M(D))
370 PRINT D$;:C$=C$+RIGHT$(D$,1):NEXT
390 PRINT:RETURN
```

# DATEBOOK

```
1 REM                                * DATEBOOK *
2 C$=CHR$(147):M$="                                "
5 DIM A$(2,100)
6 N=1:PRINT C$
10 PRINT M$;"* DATEBOOK *"
11 PRINT "      1: ENTER DATA"
12 PRINT "2: SORT          6: PRINT"
13 PRINT "3: DISPLAY       7: MODIFY DATA"
14 PRINT "4: SAVE          8: CLEAR"
15 PRINT "5: LOAD          9: QUIT"
20 PRINT M$;:INPUT "    OPTION";Q
30 ON Q GOTO 100,200,300,400,500,600,700
,890,900
100 PRINT "ENTER DATA       ZZZ TO QUIT"
110 PRINT "DATE/TIME:";:GOSUB 1000:PRINT
120 IF T$="ZZZ" THEN ND=N-1:GOTO 10
130 A$(0,N)=T$:PRINT "     EVENT:";:GOSUB
 1000:PRINT
140 A$(1,N)=T$:PRINT "      NOTE:";:GOSUB
 1000:PRINT
190 A$(2,N)=T$:N=N+1:GOTO 110
200 INPUT "SORT ON 1=DATE 2=EVENT 3=NOTE
";SK:PRINT "SORTING":FOR N=1 TO ND:P=N+1
220 IF A$(SK-1,N)<A$(SK-1,P) THEN 280
250 FOR SN=0 TO 2:T$(SN)=A$(SN,N):A$(SN,
N)=A$(SN,P):A$(SN,P)=T$(SN):NEXT
280 P=P+1:IFP<=ND THEN 220
290 NEXT:FOR SN=0 TO 2:A$(SN,ND)=T$(SN):
NEXT:PRINT "= = = = DONE = = = ="
```

```
300 INPUT "SELECT: FIELD NUMBER,KEY";SK$
,K$
310 IF SK$="" OR K$="" THEN SK=0:GOTO 33
0
320 SK=VAL(SK$):LT=LEN(K$)
330 FOR N=1 TO ND:IF SK=0 THEN 380
350 IF K$=LEFT$(A$(SK-1,N),LT) THEN 380
375 GOTO 390
380 PRINT A$(0,N);TAB(16);A$(1,N):PRINT
TAB(10);A$(2,N)
390 NEXT:INPUT "MENU";Q$:GOTO 10
400 INPUT "FILENAME FOR SAVE";F$
410 F$=F$+",SEQ,W":OPEN 2,8,4,F$
420 FOR N=1 TO ND:IF A$(0,N)="" THEN 440
430 PRINT#2,A$(0,N);",";A$(1,N);",";A$(2
,N)
440 NEXT:PRINT#2,"ZZZ,,":CLOSE2
490 GOTO 10
500 INPUT "FILENAME TO LOAD";F$
510 F$=F$+",SEQ,R":OPEN 2,8,4,F$
530 INPUT#2,A$(0,N),A$(1,N),A$(2,N):IF A
$(0,N)="ZZZ" THEN ND=N-1:CLOSE2:GOTO 10
540 N=N+1:GOTO 530
600 INPUT "PRINT SELECT: FIELD NUMBER,KE
Y";SK$,K$
610 IF SK$="" OR K$="" THEN SK=0
620 INPUT "LOWER CASE";L$:IF L$="Y" THEN
 635
630 OPEN3,4:GOTO 640
635 OPEN3,4,7
640 FOR N=1 TO ND:IF SK=0 THEN 680
650 IF K$<>LEFT$(A$(SK-1,N),LT) THEN 690
680 PRINT#3,CHR$(16);"08";A$(0,N);CHR$(1
6);"24";A$(1,N);CHR$(16);"40";A$(2,N)
690 NEXT:CLOSE3:GOTO 10
```

```
700 SK$="":INPUT "RESCHED: FIELD NUMBER,
KEY";SK$,K$:IF SK$="" THEN 10
710 SK=VAL(SK$):LT=LEN(K$):N=1
730 IF K$=LEFT$(A$(SK-1,N),LT) THEN 780
740 GOTO 760
760 N=N+1:IF N<=ND THEN 730
770 PRINT K$;" NOT FOUND":GOTO 700
780 PRINT A$(0,N);TAB(16);A$(1,N):PRINT
TAB(10);A$(2,N)
790 SK$="":INPUT "RESCHED 1=DATE 2=EVENT
 3=NOTE";SK$:IF SK$="" THEN 760
800 SK=VAL(SK$):ON SK GOTO 810,820,830
810 PRINT "DATE/TIME:";:GOSUB 1000:PRINT
:A$(0,N)=T$:GOTO 790
820 PRINT "    EVENT:";:GOSUB 1000:PRINT
:A$(1,N)=T$:GOTO 790
830 PRINT "     NOTE:";:GOSUB 1000:PRINT
:A$(2,N)=T$:GOTO 790
880 GOTO 10
890 FOR N=1 TO ND:FOR SK=0 TO 2:A$(SK,N)
="":NEXT:N=1:ND=0:GOTO 10
900 PRINT "DONE":END
1000 T$="":PRINT TAB(5);";";
1010 GET I$:IF I$="" THEN 1010
1020 IF I$=CHR$(13) THEN RETURN
1030 IF I$=CHR$(20) THEN 1100
1090 PRINT "[CR]";I$;";";:T$=T$+I$:GOTO
1010
1100 LT=LEN(T$):IF LT=0 THEN 1010
1110 T$=LEFT$(T$,LT-1):PRINT "[CR] [CR][
CR];";:GOTO 1010
```

# ORGANIZE

```
1 REM                                * ORGANIZE *
2 C$=CHR$(147):M$="                              "
5 DIM A$(2,100)
6 N=1:PRINT C$
10 PRINT M$;"* ORGANIZE *"
11 PRINT "      1: ENTER DATA"
12 PRINT "2: SORT          6: PRINT"
13 PRINT "3: DISPLAY       7: MODIFY DATA"
14 PRINT "4: SAVE          8: CLEAR"
15 PRINT "5: LOAD          9: QUIT"
20 PRINT M$;:INPUT "   OPTION";Q
30 ON Q GOTO 100,200,300,400,500,600,700
,890,900
100 PRINT "ENTER DATA       ZZZ TO QUIT"
110 PRINT:PRINT "NAME:";:GOSUB 1000:PRIN
T
120 IF T$="ZZZ" THEN ND=N-1:GOTO 10
130 A$(0,N)=T$:PRINT "NOTE:";:GOSUB 1000
:PRINT
140 A$(1,N)=T$:PRINT "DATA:";:GOSUB 1000
:PRINT
190 A$(2,N)=T$:N=N+1:GOTO 110
200 INPUT "SORT ON 1=NAME 2=NOTE 3=DATA"
;SK:PRINT "SORTING":FOR N=1 TO ND:P=N+1
220 IF A$(SK-1,N)<A$(SK-1,P) THEN 280
250 FOR SN=0 TO 2:T$(SN)=A$(SN,N):A$(SN,
N)=A$(SN,P):A$(SN,P)=T$(SN):NEXT
280 P=P+1:IFP<=ND THEN 220
290 NEXT:FOR SN=0 TO 2:A$(SN,ND)=T$(SN):
NEXT:PRINT "= = = = DONE = = = ="
```

```
300 INPUT "SELECT: FIELD NUMBER,KEY";SK$
,K$
310 IF SK$="" OR K$="" THEN SK=0:GOTO 33
0
320 SK=VAL(SK$):LT=LEN(K$)
330 FOR N=1 TO ND:IF SK=0 THEN 380
350 IF K$=LEFT$(A$(SK-1,N),LT) THEN 380
375 GOTO 390
380 PRINT A$(0,N);TAB(16);A$(1,N):PRINT
TAB(10);A$(2,N)
390 NEXT:INPUT "MENU";Q$:GOTO 10
400 INPUT "FILENAME FOR SAVE";F$
410 F$=F$+",SEQ,W":OPEN 2,8,4,F$
420 FOR N=1 TO ND:IF A$(0,N)="" THEN 440
430 PRINT#2,A$(0,N);",";A$(1,N);",";A$(2
,N)
440 NEXT:PRINT#2,"ZZZ,,":CLOSE2
490 GOTO 10
500 INPUT "FILENAME TO LOAD";F$
510 F$=F$+",SEQ,R":OPEN 2,8,4,F$
530 INPUT#2,A$(0,N),A$(1,N),A$(2,N):IF A
$(0,N)="ZZZ" THEN ND=N-1:CLOSE2:GOTO 10
540 N=N+1:GOTO 530
600 INPUT "PRINT SELECT: FIELD NUMBER,KE
Y";SK$,K$:SK=VAL(SK$):LT=LEN(K$)
610 IF SK$="" OR K$="" THEN SK=0
620 INPUT "LOWER CASE";L$:IF L$="Y" THEN
 635
630 OPEN3,4:GOTO 640
635 OPEN3,4,7
640 FOR N=1 TO ND:IF SK=0 THEN 680
650 IF K$<>LEFT$(A$(SK-1,N),LT) THEN 690
680 PRINT#3,CHR$(16);"08";A$(0,N);CHR$(1
6);"24";A$(1,N);CHR$(16);"40";A$(2,N)
690 NEXT:CLOSE3:GOTO 10
```

```
700 SK$="":INPUT "RESCHED: FIELD NUMBER,
KEY";SK$,K$:IF SK$="" THEN 10
710 SK=VAL(SK$):LT=LEN(K$):N=1
730 IF K$=LEFT$(A$(SK-1,N),LT) THEN 780
740 GOTO 760
760 N=N+1:IF N<=ND THEN 730
770 PRINT K$;" NOT FOUND":GOTO 700
780 PRINT A$(0,N);TAB(16);A$(1,N):PRINT
TAB(10);A$(2,N)
790 SK$="":INPUT "CHANGE  1=NAME 2=NOTE
3=DATA";SK$:IF SK$="" THEN 760
800 SK=VAL(SK$):ON SK GOTO 810,820,830
810 PRINT "NAME:";:GOSUB 1000:PRINT:A$(0
,N)=T$:GOTO 790
820 PRINT "NOTE:";:GOSUB 1000:PRINT:A$(1
,N)=T$:GOTO 790
830 PRINT "DATA:";:GOSUB 1000:PRINT:A$(2
,N)=T$:GOTO 790
880 GOTO 10
890 FOR N=1 TO ND:FOR SK=0 TO 2:A$(SK,N)
="":NEXT:N=1:ND=0:GOTO 10
900 PRINT "DONE":END
1000 T$="":PRINT TAB(5);";";
1010 GET I$:IF I$="" THEN 1010
1020 IF I$=CHR$(13) THEN RETURN
1030 IF I$=CHR$(20) THEN 1100
1090 PRINT "[CR]";I$;";";:T$=T$+I$:GOTO
1010
1100 LT=LEN(T$):IF LT=0 THEN 1010
1110 T$=LEFT$(T$,LT-1):PRINT "[CR] [CR][
CR];";:GOTO 1010
```