

Bert Kersey & Bill Sanders



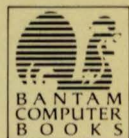
THE

BIG

TIP

BOOK

for the Apple® II Series



**THE
BIG TIP BOOK
FOR THE
APPLE II SERIES**

Bantam Computer Books
Ask your bookseller for the books you have missed

AMIGA DOS USER'S MANUAL

by Commodore-Amiga, Inc.

THE APPLE II/c BOOK

by Bill O'Brien

THE COMMODORE 64 SURVIVAL MANUAL

by Winn L. Rosch

COMMODORE 128 PROGRAMMER'S REFERENCE GUIDE

by Commodore Business Machines, Inc.

EXPLORING ARTIFICIAL INTELLIGENCE ON YOUR APPLE II

by Tim Hartnell

EXPLORING ARTIFICIAL INTELLIGENCE ON YOUR COMMODORE 64

by Tim Hartnell

EXPLORING ARTIFICIAL INTELLIGENCE ON YOUR IBM PC

by Tim Hartnell

EXPLORING THE UNIX ENVIRONMENT

by The Waite Group/Irene Pasternack

FRAMEWORK FROM THE GROUND UP

by The Waite Group/Cynthia Spoor and Robert Warren

HOW TO GET THE MOST OUT OF COMPUSERVE, 2d ed.

by Charles Bowen and David Peyton

HOW TO GET THE MOST OUT OF THE SOURCE

by Charles Bowen and David Peyton

MACINTOSH C PRIMER PLUS

by The Waite Group/Stephen W. Prata

THE MACINTOSH

by Bill O'Brien

THE NEW jr: A GUIDE TO IBM'S PCjr

by Winn L. Rosch

ORCHESTRATING SYMPHONY

by The Waite Group/Dan Schafer

PC-DOS/MS-DOS

User's Guide to the Most Popular Operating System for Personal Computers

by Alan M. Boyd

POWER PAINTING: COMPUTER GRAPHICS ON THE MACINTOSH

by Verne Bauman and Ronald Kidd/illustrated by Gasper Vaccaro

SMARTER TELECOMMUNICATIONS

Hands-On Guide to On-Line Computer Services

by Charles Bowen and Stewart Schneider

SWING WITH JAZZ: LOTUS JAZZ ON THE MACINTOSH

by Datatech Publications Corp./Michael McCarty

TEACH YOUR BABY TO USE A COMPUTER

Birth Through Preschool

by Victoria Williams, Ph.D. and Frederick Williams, Ph.D.

UNDERSTANDING EXPERT SYSTEMS

by The Waite Group/Mike Van Horn

USER'S GUIDE TO THE AT&T PC 6300 PERSONAL COMPUTER

by David B. Peatroy, Ricardo A. Anzaldua, H. A. Wohlwend, and Datatech Publications Corp.

**THE
BIG TIP BOOK
FOR THE
APPLE II SERIES**

Bert Kersey and Bill Sanders

Beagle Bros Micro Software Inc.



BANTAM BOOKS

TORONTO • NEW YORK • LONDON • SYDNEY • AUCKLAND

*This book is dedicated to Sophie,
the Beagle Bros beagle,
and Jingle, the Microcomscribe mutt.*

THE BIG TIP BOOK FOR THE APPLE II SERIES

A Bantam Book / April 1986

*Apple II, Apple II+, Apple IIc, and Apple IIe are registered
trademarks of Apple Computer, Inc.*

Cover design by J. Caroff Associates.

*Throughout the book, the trade names and trademarks of some
companies and products have been used, and no such uses
are intended to convey endorsement of or other affiliations
with the book.*

All rights reserved.

Copyright © 1986 by Bert Kersey and William B. Sanders.

Book design by Nicola Mazzella.

*This book may not be reproduced in whole or in part, by
mimeograph or any other means, without permission.*

For information address: Bantam Books, Inc.

ISBN 0-553-34280-0

Published simultaneously in the United States and Canada

*Bantam Books are published by Bantam Books, Inc. Its trade-
mark, consisting of the words "Bantam Books" and the por-
trayal of a rooster, is Registered in U.S. Patent and Trademark
Office and in other countries. Marca Registrada. Bantam
Books, Inc., 666 Fifth Avenue, New York, New York 10103.*

PRINTED IN THE UNITED STATES OF AMERICA

HL 0 9 8 7 6 5 4 3 2 1

CONTENTS

Preface	ix
1. Environment	1
2. Keyboard	15
3. Memory Structure	21
4. Applesoft Tips	35
5. Memory and Speed	68
6. List Tips	73
7. Text Tips	83
8. A Few Bugs	94
9. Protection	97
10. Sound	106
11. Graphics	109
12. Integer (the Other BASIC)	127
13. Machine Language	133
14. Disks and DOS	144
15. Catalog	175
16. Apple II, II + , //e, and //c	188
17. Maintenance	193
18. Insignifica	198
19. Two Liners	216
Apple Commands	232
Peeks, Pokes and Pointers	234
Apple Colors and ASCII Values	236
INDEX	237

ACKNOWLEDGMENTS

As the old Apple II evolved into the II+, //e, //c and beyond, far more people have lent us a hand than we can name, but we'll name a few anyway: Jack Cassidy, Eric Goetz, Mark Simonsen, Alan Bird, Rob Renstrom, Randy Brandt, Roger Wagner, Tom Weishaar and various members of the Apple Corps of San Diego. Thanks everybody!

PREFACE

If you hang around people with Apple computers, you pick up all kinds of nifty tips. Both of us hang around a lot and have amassed an amazing amount of tips from fellow users and our own experiments. Many of the tips in this book have been published separately in the Tip Books that accompany Beagle Bros software. Users from all over the country have used these little books to look up little tips (or a lot of tips in little places, or big tips . . . never mind). What's more, they've written in (mostly on word processors) with their discoveries in the form of tips. For the most part these tips are obviously useful, ingenious, and valuable. However, a lot of them are exotic, weird, and apparently useless. The obviously useful tips can be put to immediate use, and you'll be better off in a way you can appreciate right off the bat. The other ones, however, can actually be more useful in the long run. For example, one of my favorite CALLs is -1401. Now, as just about nobody knows (or cares), you can CALL -1401 to boot your system instead of the boring PR#6 or even duller method of turning on your computer with a DOS disk in drive #1. The question is *why*? What actually happens when you CALL -1401? As anyone who reads the *Apple Reference Manual* can tell you, the boot location is -1370 and not -1401. Nevertheless, you get a really clean boot with a CALL -1401. This kind of mystery can lead you by the nose into your Apple to learn more about why it does what it does (Don't tell me why if you know. I'd rather

find out myself.) So you see, the not-so-obvious tips can really set your sail to learning more about your Apple.

Thus, we have collected exactly two kinds of tips. Those that you can put to work and those that make you wonder. With our wheelbarrow full of tips, we decided that it would be a good idea to *get organized* (or sorted out, at least) and to arrange everything into the nineteen chapters and umpteen appendixes that make up this book. In that way, when you want a tip on DOS, you don't have to wade through all the tips on text, graphics, and all the other tips that have absolutely no bearing on DOS. (Besides, the wheelbarrow gets in the way of the monitor.)

In addition to the exactly two kinds of tips, there are a lot of program examples to show you how to do things with the tips. These illustrate a point made in the tip, show you how to do something really neat, provide a handy algorithm, or make you wonder.

There's more, though. A few years ago, Beagle Bros started a two-liner contest to see who could do the most in only two lines of a BASIC program. The creations were astounding. Not only were the two liners good little programs, they did it in two lines! These provide a wealth of tight algorithms and creative talent. Since fewer lines will speed up a program, these two liners can really teach you a lot about BASIC programming. (Besides, with only two lines, you don't have to spend your vacations keying them in.)

Finally, anyone who has done much programming finds a good chart worth a thousand SYNTAX ERRORS. Since Beagle Bros has some of the best charts in the business (this side of *The Apple Almanac*, at least), we thought it'd be a good idea to put them in the appendixes. You can tape the book to your wall and have instant access to your favorite chart. Of course, this is our get-rich-quick scheme, since you'll have to buy a lot of books to see all your charts at once. (I told you it wouldn't work, Bert.)

But this isn't just a rehash of the old Tip Books and charts. Bill was the editor of the San Diego Apple Corps' newsletter, *The Sandy Apple Press*, and picked up plenty of tips from Apple users while trying to figure out how to fill up the monthly issue. (The "letter" grew to fifty pages before Bill had enough sense to let someone else take over the editorship.) In addition, he wrote *The Elementary Apple* and coauthored with Eric Goetz, *The Apple Almanac*. These two books led to various discoveries that didn't quite fit into either work and have been sitting on file awaiting a suitable home. These tips have been added to Bert's collection.

As the Apple changed from the Apple II to the II+ to the //e and the //c, earlier tips had to be reconsidered. Not only did the machines change, but DOS did as well, evolving from 3.1 to ProDOS. Sometimes a tip for one machine may not be relevant to another. For example, if you dribble in your keyboard on your II, II+, or //e, you may get to see your chips fry as the liquid mingles with the electric circuits. This just isn't true on the //c since Apple, Inc., has made a dribble-resistant keyboard. Of course, Bert warned users of this disaster long ago and provided a surefire cure. (See the later section "Don't Drink and Compute.") After studying the design suggested by Bert, Apple decided to follow the tip and made the //c keyboard so that the liquid was rerouted to your lap instead of the circuits. (Well, maybe Apple, Inc. didn't get the idea from Bert, but the tip *did* come before the //c.) In order to make *The Big Tip Book* useful for all Apple IIw users (the w stands for "whatever") and not just the old or new ones, we have fixed up and rewritten the tips to keep up with modern times.

To get you going, the first tip is in using *The Big Tip Book*. After taking all of our combined tips out of the wheelbarrow, we put them in a loose-leaf binder arranged by chapter. Now it was a lot faster finding things in the binder than in the wheelbarrow, but since there were so many tips, often-used ones (especially those involving PEEKs, POKEs, and CALLs *nobody* remembers exactly), a quick lookup became a long lookup. So for the often-used but often-forgotten tip, we have the following tip. The 3M Company makes these little paper things called "Post-its." They have sticky stuff on them so that you can plaster a little note on paper, and it stays put until you pull it off. The stickum is nonbinding, so your tip book will not get ripped if you remove the Post-it. When you want to mark something, instead of dog-earing the page, put a little Post-it on the page with a note to yourself. For example, who can remember that you should use POKE -16368,0 to clear the keyboard buffer after using a WAIT -16384,128? We can't. So instead of wading through all the tips in the keyboard section (or was it the text section?), just mark the Post-it with something like "WAIT HERE" and stick it on the appropriate page so you can see at a glance where your tip is located. Then you can look it up quickly.

CHAPTER 1

ENVIRONMENT

SLOT AND DRIVE

If you have more than two disk drives, label them with black Dymo-type labels ("S6,D1", "S6,D2", "S5,D1", etc.) so you can tell at a glance which drive goes with which slot.

WHICH PADDLE?

Also, identify your paddles with "-0-" and "-1-" Dymo labels. You can tell which paddle is which with the following program. RUN it and mess with your paddles. . . .

```
10 HOME :BUTUN = - 16287
20 FOR X = 0 TO 1: NORMAL
30 IF PEEK (X + BUTUN) > 127 THEN INVERSE
40 VTAB 10: HTAB 1 + 20 * X
50 PRINT "PADDLE ";X;":":": NORMAL
```

```
60 PRINT " "; PDL (X); SPC( 2)
70 NEXT X: GOTO 20
```

The word "PADDLE" should light up when the appropriate button is pressed, and you should get values from 0 to 255 when you turn the knobs. If you don't, you can clean the contacts inside (carefully) with some electrical contact cleaner. The paddles are undoubtedly the weakest Apple hardware link. Good old Rob and Brent at the Apple store replaced my original thumb-killer paddle buttons with nice fat red ones they bought from Radio Shack (what?). I highly recommend the conversion to all serious paddlers.

SPACE ERASERS

Notice the SPC(2) after the PDL value in line 60. This erases trailing digits when the values drop from three to two or from two to one digit. Try the program with and without the SPC(2) and you'll see.

A MOUSE AND NO PADDLES?

Use the Dymo labeler to name your mouse "Ralph."

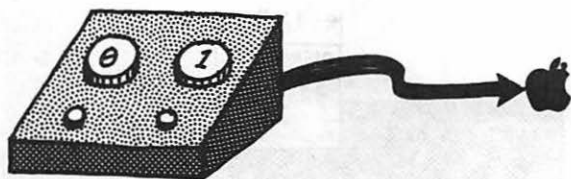
TIP YOUR COMPUTER AND DISKS

Before you put your Dymo away, why not stick your favorite tip above your keyboard or on your disk drive? CALL 1002 fits nicely on your disk drive and will remind you to call that number when you blow your DOS. Likewise 67 68 AF B0 above your keyboard will help you remember the

monitor addresses for the pointers to the beginning and ending addresses of your BASIC program in memory.

PADDLE PANEL

Ever think of disassembling and mounting your paddles on a control panel, like so?



Kind of like electric-train transformers; all of the buttons where you want them. Uncle Louie mounted his paddles on both sides of his Apple to make some nifty *Raster Blaster* flippers.

DISK DRIVE PLACEMENT

To find out on which side of your monitor or TV set to place your disk drives, use a radio. Turn on your radio and place it on the left and right sides of your monitor or TV. The side that causes interference on your screen is *not* the side to place your disk drives.

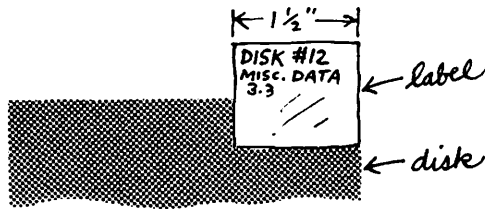
UNSTABLE LABELS

We suspect that most disks are made of Teflon just to keep labels and write-protect tabs from sticking. We've had better luck with certain "per-

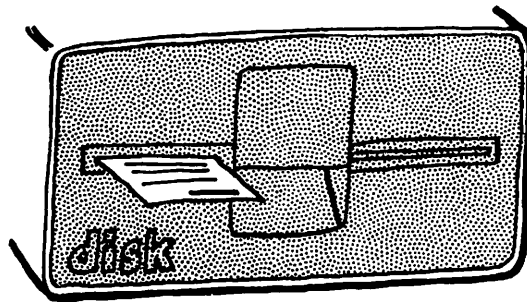
manent" pressure sensitive labels (not "removable"). Visit your stationery store.

VISI-LABELS

You can make nice 1½" x 2" disk labels from business-card-weight cardboard, like so. . . .



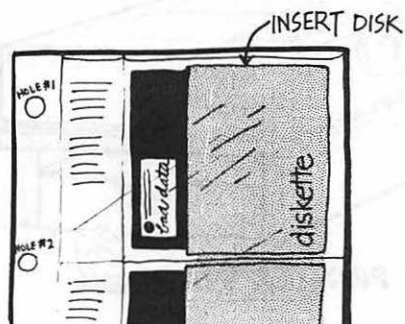
Now you can see what disk is in each of your drives.



EASY INSERT

If you use three-ring vinyl inserts to store your disks, you may think you have to remove each disk from its protective sleeve for it to fit in the page pocket. Have you tried inserting the disk plus sleeve in the pocket *sideways*?

It works *and* gives your disks double protection. Now you know what to do with all of those extra disk sleeves!

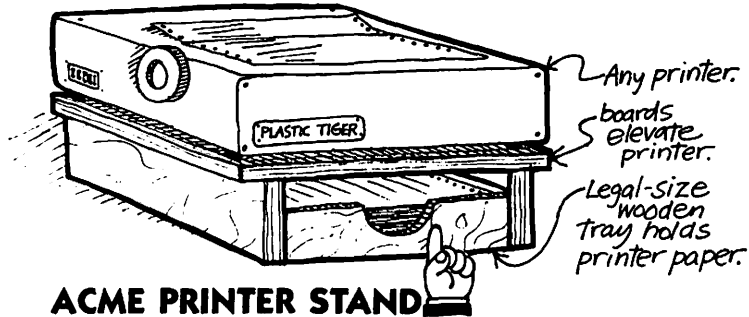


MONITOR BLUES



They say (and I tend to believe them) that a black monitor screen with white text is hard on your eyes, if not your brain. A better choice is one of the "green screen" monitors; some claim that amber is even better (especially under fluorescent light). If you do a lot of *VisiCalc*-ing or word processing, an easy-eye monitor may be well worth the investment. The only thing worse than a black-and-white monitor (for the eyes, that is) is a TV, particularly a color TV.

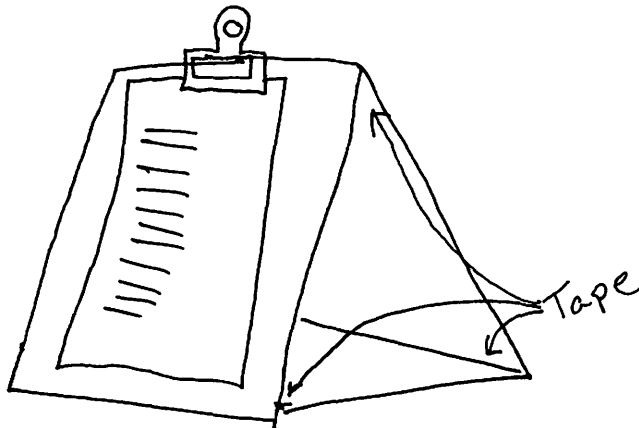
ACME PRINTER STAND



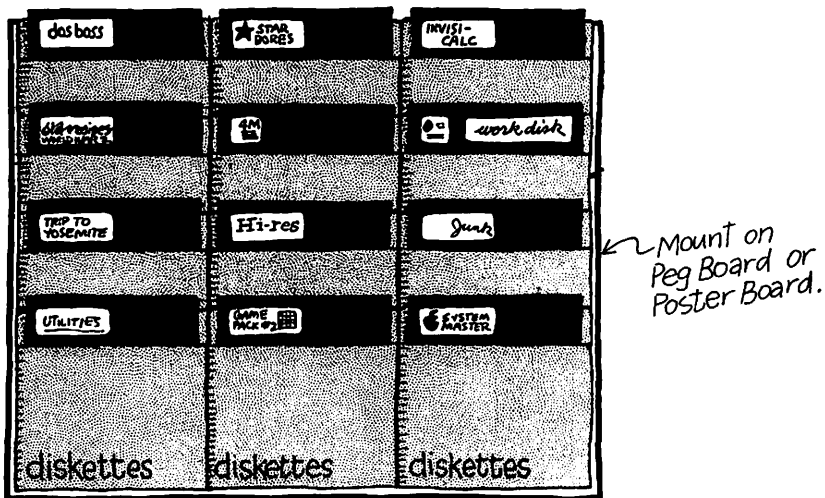
Uncle Louie built this dandy printer stand for our printer. Maybe it will work for yours.

PROGRAM LIST STANDS

If you have ever tried to key in programs from a book, magazine, newsletter, or just about any other hardcopy listing, you've noticed that they don't stand up too well by themselves. In most office-supply stores you can find stands used by secretaries to hold documents they copy. These work much better. For those of you on a budget, the following stand works just fine, too.



DISK RACK



Tape or glue a bunch of disk sleeves together and mount them on a board for a handy disk rack. This way, all of your most commonly used disks are always findable and reachable!

NO-TEAR SLEEVES!

Next to creating shapes from scratch, one of the hardest things to do is tear up a disk sleeve! Try it sometime. Certain ones are made of special no-tear paper. Extensive research by an independent testing lab shows that no one knows why, although dust prevention is undoubtedly the answer.

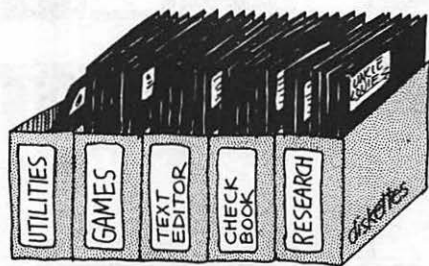
TWO-SIDED DISKS

No problem. Just punch a write-protect notch with a 1/4" punch directly opposite the existing notch. (Put two disks front to back and pencil in the spot for the notch.) Then INIT the new side of the disk.

Warning: Some disks have flaws on their wrong sides.

But: Most don't.

DISK BOXES

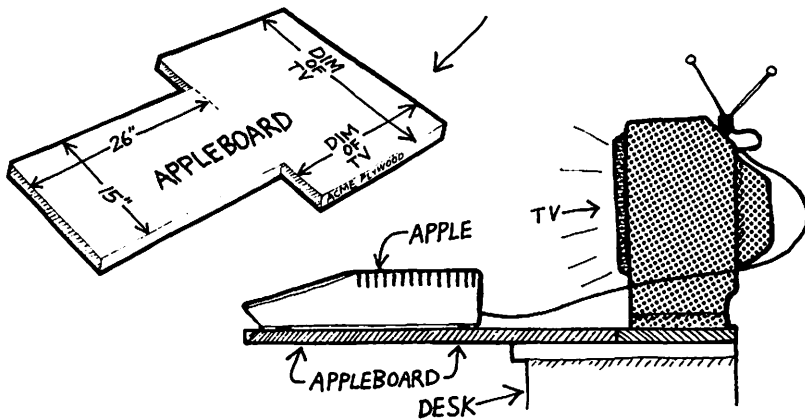


Here's another tasty way to store your disks. Use the ten-disk boxes with the lids removed and store your disks sorted by category. Labels on the sides of the boxes finish things off nicely.

AN APPLEBOARD

If you're using a 15" or bigger TV monitor with your Apple, you will probably agree with us that placing it on top of the computer puts it much too close to your eyes. (If only your arms were longer. . . .) And your desk isn't deep enough to leave room for the TV behind the Apple. And it's a pain trying to type while you look sideways, right?

Okay, cut a piece of $\frac{3}{4}$ " plywood, like so.

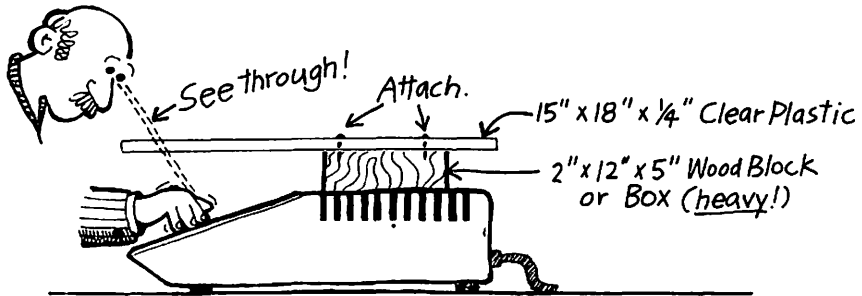


That way, your Apple can overhang the front of your desk, and you are far enough from your monitor to prevent TV eyeballs. The whole unit can be pivoted or slid out of the way when not in use.

DON'T DRINK AND COMPUTE! (UNLESS YOU HAVE A //c)

Computers provide great party entertainment, but *beware* of liquid whatever getting onto and into your keyboard (not to mention cigarette ashes, toothpicks, cracker crumbs, cat hair . . .).

A Solution: Obtain a 15" x 18" piece of 1/4" clear plastic and cover your keyboard.



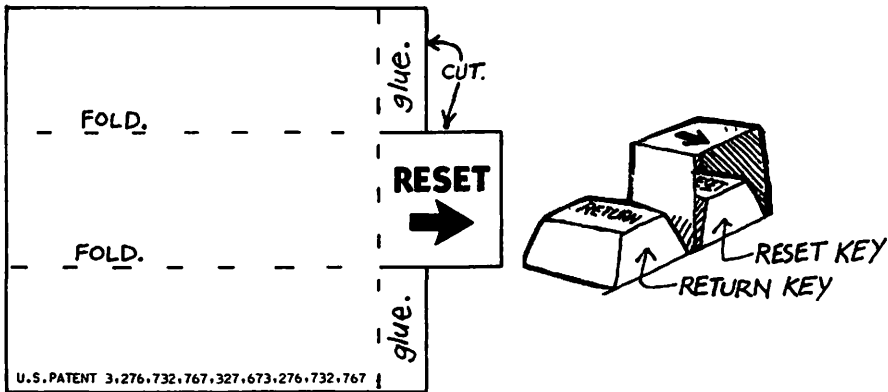
Then enjoy the party! Of course, the Apple //c users have a liquidproof keyboard already.

RESET PROTECTION!

In the olden days (computer terminology for a couple of years ago), a 1/4" error when hitting RETURN could cause you to hit Reset and kill a program. The new Apples require a two-handed* Control-Reset. Hooray! A terrible flaw is eliminated! For those of us with our ancient Apples, here are some solutions:

1. Ask your Apple dealer. There are several hardware Reset protectors for sale, including a dealie that requires two Reset hits within a second and a Reset switch that attaches to the back of your Apple. (Go to an old Apple dealer.)
2. Make your own Reset protector.

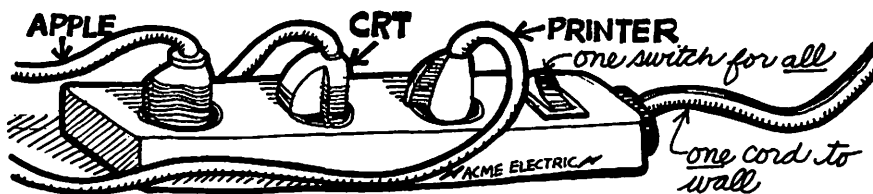
*Our neighbor, Kareem Abdul-Murphy, can do a Control-Reset with one hand.



Let's forgive Apple for one B-L-U-N-D-E-R!

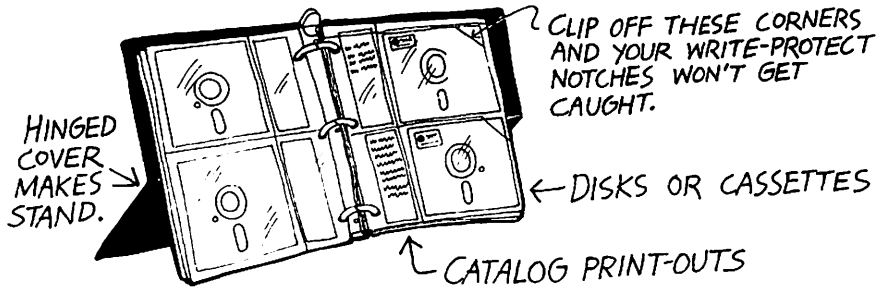
MULTI-PLUG

Tired of having to turn on your Apple, your monitor, *and* your printer every time you power up? Why not buy a multiple outlet box with a powerswitch? Then you can set everything into action with one switch. You won't have to remember to turn everything *off* either. One switch does it all!



You can do all of this and get a fan to boot with a device that attaches to the side of your Apple. They come under a lot of names, but the one we know best is called System Saver made by Kensington Microware.

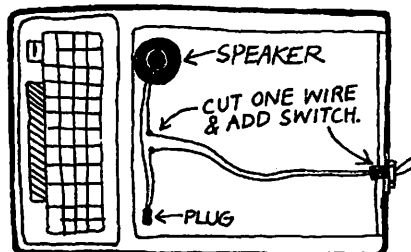
AN EFFICIENT SOFTWARE FILE



K & M Co. makes a nice three-ring binder (#ES 311-1) that stands up when open. Use it with some disk-holder inserts and you've got a super disk file!

THINGS THAT GOSUB IN THE NIGHT

Around 2:00 A.M. a Control-G can sound like an air-raid siren. And sometimes, even on Saturday afternoon, a little silence is nice. It's a simple thing to put a switch on your Apple speaker. Just cut one of the speaker wires and run new wires to a toggle switch.



Certain types of switches will attach through the slots in the back of the Apple, and make hole drilling (gasp!) unnecessary. Of course, on the Apple //c, things are different.

VARIABLE VOLUME

If you like arcade games with lots of sound as your spaceship gobbles aliens, your puny Apple speaker's squeak may not do the job. (Or if it's too loud it may overdo the job.) A surefire solution to that problem is to cut the wire from your minispeaker and attach to a big speaker with volume control. You can splice a longer wire onto the one from your speaker to its plug (see preceding diagram) to take it out the back of your Apple. The sound can be turned up for your shoot-outs and then down while everyone else in the world sleeps.

GRAND PRIX APPLE

Peter Lake, from somewhere-out-there, called us here at the Beagle Building to report on his customized Apple environment. It seems Peter is a professional writer and uses an Apple as a word processor all day, every day. So he mounted his system on a drawing board tilted about 15 degrees. To level his monitor, he built a wedge-shaped mount that rests on his Apple. He painted his Apple's case black to reduce glare (I never noticed the problem) and, inspired by one of our tips, glued sequins to his *K* and *D* keys so he can feel his fingers' positions on the keys without looking. Now the best part: he sits in a special race-car seat, headrest and all, leans back, and types away. I asked him if he wears a seat belt; he said, "Only when I type fast."

AUDIO RADIANCE

Take a portable radio that's turned on and place it against one side of your Apple monitor. Now try the other side. You'll probably notice that one side causes more interference than the other. If I were you, I'd keep my disks away from that side.

CHAPTER 2

KEYBOARD

WATCH YOUR 3's AND 2's

Don't fret if you have to look at the keys when you type numbers. Mistyped numbers are the source of some of the toughest programming mistakes to find. I've found it pays to slow your typing down about two beats when numbers come up, and it doesn't hurt to *watch what you're doing* either.

ON ANCIENT APPLES THERE'S MORE THAN ONE WAY TO . . .

There are ways to type Apple's three keyless characters: the backslash, the left square bracket, and the underscore. Here's one of them: try pushing any three or four keys at the same time. It's difficult to predict what will appear on the screen. If you have a lowercase adapter (which is on all the //e's and //c's), you can often produce lowercase letters by pressing two

keys. With this in mind, we set out to find our elusive keyless characters, figuring that some combination of other keys would produce what we wanted. Sure enough. Simultaneously press the SHIFT, *U*, and *I* keys and *hold them down* while you type a *Y*, then an *H*, and then a *J*. There they are; the three elusive characters, each with a normal character thrown in. Kind of strange, huh? You can make them usable by typing these characters as described here, and then deleting the characters you don't want by normal editing methods.

MISSING-CHARACTER DEPARTMENT

When they designed the Apple II and II+ keyboard, they forgot to print the right bracket above the *M*. If you want to type a right bracket, type a SHIFT-M. (Apple, Inc., read this tip and fixed up the //e and //c so you don't have to go through this.)

HOW TO GET RICH

Market replacement Apple *K* and *D* key caps with little lumps on them. Your fingers can then feel their way to the correct keys. The Apple //e and //c use this method, and it works.

&, @, ETC.

All right, if you're going to be serious about this computer stuff, it's time you got your character names straight. Here's the rundown; there will be a quiz on Monday.

- & This is called an *ampersand*.
- @ This is called an *at sign*.
- * This is called an *asterisk*.
-] This is called a *square bracket*.
-) This is called a *parenthesis*.
- # This is called a *pound sign*.
- ^ This is called a *little pointy thing*.

PROGRAMMING THE RESET KEY

There are a couple of POKes you can do into DOS that will make Reset act something like a Control-C. That makes it trappable by ONERR GOTO, and thereby makes it programmable to do anything you want it to. The following program demonstrates.

Wait a minute! Be careful out there Hill Streeters. If you make a typo, you could really foul things up. After you type it in, SAVE it, *then* RUN it. Try to stop it with a Control-C or a Reset. You can't. You have to let it run its course. The two POKes at the beginning of the program even program your electric plug so you can't disconnect your Apple (just kidding).

```
20 ONERR GOTO 1000
25 POKE 40286,35 : POKE 40287,216
30 X = X + 1
40 PRINT X;" "; : IF X < 400 THEN 30
50 POKE 40286,60 : POKE 40287,212 : END
1000 PRINT : PRINT "START COUNTING..."; : X
    = 0 : GOTO 30
```

Line 1000 could just as easily contain a NEW or LIST or RUN or DEL 0,999 or PRINT CHR\$(4); "PR#6" or PRINT CHR\$(4); "CATALOG".

Here's a set of POKes that will make the Reset key boot when it's pressed:

```
FOR X=1011 TO 1015 : POKE X,0 : NEXT
```

Or:

```
*3F3: 00 00 00 00 00
```

By the way, Reset resets almost everything, but it won't reset SPEED to its normal 255 value.

RESET TO THE MONITOR!

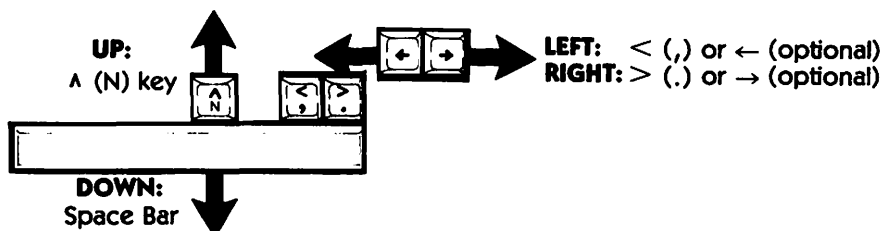
We know a guy who chucked his Autostart ROM in the garbage disposal just because it makes Reset return him to BASIC instead of the monitor. He should have typed:

```
CALL -151
*3F2: 69 FF 5A
```

Make the 5A a 51 and Reset will *reboot!* (Normal is *3F2: 00 97 32.)

N,S,W, OR E?

To move a figure up, down, left, or right on an Apple II+, the *U*, *D*, *L*, and *R* arrows or *N*, *S*, *W*, and *E* keys are often used, but they are a real pain for nontypers. *I*, *M*, *J*, and *K* or some similar grouping isn't much better. We prefer this easy-to-remember, uncrowded set:



Five of these six keys are already marked with “arrows” (mark your space bar too, if you want), and all are conveniently spaced for two-handed use.

ESCAPE B ?

Everybody who ever made a SYNTAX ERROR knows how to hit Escape and climb up a wall of line numbers to edit. The cursor always seems to go up the second digit of the line numbers, and you have to hit *J* to get it over the first digit and then remember to hit the space bar to get out of the edit mode, *and then* you can run the cursor over the program to change your comma to a colon. To save exactly one stroke, instead of hitting *J*, hit *B*. The cursor pops back exactly one space and takes you out of the editor. (If you have the newfangled Apples with the up and down keys, you won't appreciate this.)

MYSTERY CHARACTERS ON //e AND //c

To impress your girlfriend or boyfriend with *inverse lowercase* on your //e or //c, POKE 49167,0. Now POKE your character set into the screen. If you don't want to do that and break your screen, the following program does it for you.

```
10 TEXT : HOME
20 FOR X=0 TO 255
30 POKE X+1024,X
40 NEXT X
50 FOR P=1 TO 2000 : NEXT P
60 PRINT CHR$(7) :POKE 49167,0
```

Watch things change when you hear the bell. If you want to get it back to normal, POKE 49166,0.

INCREASE YOUR TYPING SPEED ON THE //c

You probably don't realize this, but the QWERTY keyboard was designed to slow typists down. The old machines (I think they called them typewriters) just couldn't keep up with Miss Nerty's speed typists. This embarrassed the typewriters and clogged their keys; so they made the QWERTY keyboards to tangle up fingers.

With word processors came type-ahead buffers that could keep up with Attila the Typist, so they decided to put in the Dvorak keyboard that will allow you to type 40 percent (some claim) faster. On the Apple //c, you can have either the QWERTY or the Dvorak by punching a little toggle switch on top of the computer with a pen. All you have to do is remember which keyboard you have going since the keys are still marked in QWERTY. If you switch back and forth, you may get *lock finger*.

CHAPTER 3

MEMORY STRUCTURE

SCREEN SHIFTER

This program performs a fast, simple memory move from Applesoft, similar to the one on the PEEKs and POKEs chart. RUN it and press the left arrow a few times until any character other than a space is in the upper-left corner of the screen. Then press the *right* arrow!

```
10 POKE 768,216: POKE 769,160: POKE 770,0:
   POKE 771,76: POKE 772,44: POKE 773,254:
   LIST : LIST
20 HTAB 1: PRINT "<- ->:";: GET A$:ST =
   1024:EN = 2046
30 DEST = 1024 + (A$ = CHR$ (21)) - (A$ =
   CHR$ (8)): GOSUB 100: GOTO 20
100 I = INT (ST / 256): POKE 60,ST - 256 *
   I: POKE 61,I
110 I = INT (EN / 256): POKE 62,EN - 256 *
   I: POKE 63,I
120 I = INT (DEST / 256): POKE 66,DEST -
   256 * I: POKE 67,I: CALL 768: RETURN
```

The left arrow moves the text screen memory *down* one byte. The right

arrow moves it *up*. Due to the nature of this memory move, the upward move *copies* the value of the first byte (VTAB 1, HTAB 1) all the way through the screen!

Try pressing the right arrow when a character is in the upper-left corner of the screen.

SCREEN FILLER

This program works like the preceding one. Its task is to instantly fill the screen with any character you print in line 10.

```

5  POKE 768,216: POKE 769,160: POKE 770,0:
   POKE 771,76: POKE 772,44: POKE 773,254
10 FOR X = 1 TO 12: VTAB 1: PRINT MID$
   (":-+*@*+-:",X,1)
20 POKE 60,0: POKE 61,4: POKE 62,254: POKE
   63,7: POKE 66,1: POKE 67,4: CALL 768:
   NEXT : RUN

```

SHIFTY PICTURES

This little program does tricks with your hi-res screen by shifting memory. First, RUN B.B. LOGO from the DOS BOSS disk or BLOAD some lesser hi-res effort. Enter the monitor with a CALL -151, and type:

```

*2000<2005.3FFFFM <RETURN>
*2000<2010.3FFFFM <RETURN>
*2000<2025.3FFFFM <RETURN>

```

These commands tell the Apple to *move memory* from the address range on the right to the range starting with the address on the left. Substitute your own number after the "<". Fun, huh?

AHHH, NOW JUST A MINUTE . . .

Enter the following program in its entirety *exactly* as shown. First, why does A\$ appear three different ways? Second, what happened to the "<>" the second time A\$ was printed?

```
10 A$ = "1234567890ABCDEFGHIJKLMNQPQRSTUVWXYZ?+<>#$$%&*"  
15 TEXT  
20 HOME  
25 POKE 32,2  
30 PRINT A$  
35 VTAB 4  
40 PRINT A$  
45 TEXT  
50 VTAB 7  
55 PRINT A$  
60 LIST
```

Everyone knows why the "<>" disappears. It's the Jimmy Hoffa effect. But what about that first A\$?

YOUR BASIC PROGRAM AT WORK

Let's take a look at a very small program. Type this in and follow along.

```
1    REM COUNT  
2    FOR I = 1 TO 10  
100  PRINT I : NEXT  
1000 PRINT "END" : END
```

Most Applesoft programs start at 2049 (\$801). To find the end of a program—

```
PRINT PEEK (175) + PEEK(176) * 256
```

The longer your program, the higher this number will be. In this case, we get 2097, or hex \$831. Now, let's look at the entire program in the monitor, but listing just before and just after the start and end points:

```
CALL -151 <RETURN>
*800.832 <RETURN>
```

You will now see something like:

```
800- 00 0C 08 01 00 B2 43 4F
808- 55 4E 54 00 18 08 02 00
810- 81 49 D0 31 C1 31 30 00
818- 21 08 64 00 BA 49 3A 82
820- 00 2E 08 E8 03 BA 22 45
828- 4E 44 22 3A 80 00 00 00
830- 0A 4E 44
```

Each two-digit hex number represents part of your program: a line number, character, or command token. Let's take a look at each one.

<i>Location</i>	<i>Value</i>	
800	00	There is always a zero before the start of a program.
801-802	0C 08	These two numbers tell us that the <i>next</i> program line starts at location \$080C (Apple's two-byte hex numbers are always backward in this strange form).
803-804	01 00	This is the first line number, 1 (0001).
805	B2	This is the symbol or <i>token</i> for the word "REM." Each word Applesoft knows has a token number.
806-80A	43 4F 55 4E 54 00	These five numbers are the ASCII values for the characters C-O-U-N-T. A zero token means end of line, so line 1 is finished.
80C-80D	18 08	This is the start of the next program line, the location that was referred to up in 801. The values 18 and 08 tell us that the <i>next</i> program line starts at \$818.
80E-80F	02 00	Line number 2 (\$0002).
810	81	Token for FOR.
811	49	Character I.
812	D0	Token for =.
813	31	Character 1.
814	C1	Token for TO.

<i>Location</i>	<i>Value</i>	
815-816	31 30	Characters 1 and 0. Applesoft handles numbers in a program as words, spelling them out digit by digit.
817	00	End of line.
818-819	21 08	Next line starts at \$821.
81A-81B	64 00	Line number 100 (\$0064).
81C	BA	Token for PRINT.
81D	49	Character <i>I</i> .
81E	3A	Character <i>:</i> .
81F	82	Token for NEXT.
820	00	End of line.
821-822	2E 08	Next line starts at \$82E.
823-824	E8 03	Line number 1000 (\$3E8).
825	BA	Token for PRINT.
826	22	Quote mark.
827-829	45 4E 44	Characters <i>E</i> , <i>N</i> , and <i>D</i> .
82A	22	Quote mark.
82B	3A	Character <i>:</i> .
82C	80	Token for END.
82D	00	End of line.
82E-82F	00 00	Two zeros as at this point mean end of program.
830-832	?? ?? ??	Could be part of old program.

TAKING OUT THE GARBAGE

Occasionally, a program will be zapped by mysterious forces and become inoperable. In the listing, you will find illegal line numbers or nonsense statements, like "COLOR = GOSUB H PLOT." What has probably happened is that *one byte* has been changed in a line number or *line location number*. If the latter is true, your program will jump to wherever it's told, which is now some random place in memory. The numbers it finds there are interpreted as tokens, and thus the garbage. Use line search to find the memory location of the garbaged line and/or the last good line in the program. Check the pair of numbers just before each line number. Each one should point to a location *just after a 00* (the end of the current line). When you find an incorrect number, POKE in a correct one. For example,

to POKE a 100 (\$64) into location 2049 (\$801), type POKE 2049,100, *or* enter the monitor with a CALL -151 and type 801:64.

APPLESOFT TOKENS

Hex	Dec	Chr	Hex	Dec	Chr	Hex	Dec	Chr
\$00	0	c@	\$1F	31	c__	\$3E	62	>
\$01	1	cA	\$20	32	sp	\$3F	63	?
\$02	2	cB	\$21	33	!	\$40	64	@
\$03	3	cC	\$22	34	''	\$41	65	A
\$04	4	cD	\$23	35	#	\$42	66	B
\$05	5	cE	\$24	36	\$	\$43	67	C
\$06	6	cF	\$25	37	%	\$44	68	D
\$07	7	cG	\$26	38	&	\$45	69	E
\$08	8	cH	\$27	39	'	\$46	70	F
\$09	9	cI	\$28	40	(\$47	71	G
\$0A	10	cJ	\$29	41)	\$48	72	H
\$0B	11	cK	\$2A	42	*	\$49	73	I
\$0C	12	cL	\$2B	43	+	\$4A	74	J
\$0D	13	cM	\$2C	44	,	\$4B	75	K
\$0E	14	cN	\$2D	45	-	\$4C	76	L
\$0F	15	cO	\$2E	46	.	\$4D	77	M
\$10	16	cP	\$2F	47	/	\$4E	78	N
\$11	17	cQ	\$30	48	0	\$4F	79	O
\$12	18	cR	\$31	49	1	\$50	80	P
\$13	19	cS	\$32	50	2	\$51	81	Q
\$14	20	cT	\$33	51	3	\$52	82	R
\$15	21	cU	\$34	52	4	\$53	83	S
\$16	22	cV	\$35	53	5	\$54	84	T
\$17	23	cW	\$36	54	6	\$55	85	U
\$18	24	cX	\$37	55	7	\$56	86	V
\$19	25	cY	\$38	56	8	\$57	87	W
\$1A	26	cZ	\$39	57	9	\$58	88	X
\$1B	27	c[\$3A	58	:	\$59	89	Y
\$1C	28	c\	\$3B	59	;	\$5A	90	Z
\$1D	29	c]	\$3C	60	<	\$5B	91	[
\$1E	30	c^	\$3D	61	=	\$5C	92	\

Hex	Dec	Chr	Hex	Dec	Chr	Hex	Dec	Chr
\$5D	93]	\$84	132	INPUT	\$AB	171	GOTO
\$5E	94	^	\$85	133	DEL	\$AC	172	RUN
\$5F	95	—	\$86	134	DIM	\$AD	173	IF
\$60	96	`	\$87	135	READ	\$AE	174	RESTORE
\$61	97	a	\$88	136	GR	\$AF	175	&
\$62	98	b	\$89	137	TEXT	\$B0	176	GOSUB
\$63	99	c	\$8A	138	PR#	\$B1	177	RETURN
\$64	100	d	\$8B	139	IN#	\$B2	178	REM
\$65	101	e	\$8C	140	CALL	\$B3	179	STOP
\$66	102	f	\$8D	141	PLOT	\$B4	180	ON
\$67	103	g	\$8E	142	HLIN	\$B5	181	WAIT
\$68	104	h	\$8F	143	VLIN	\$B6	182	LOAD
\$69	105	i	\$90	144	HGR2	\$B7	183	SAVE
\$6A	106	j	\$91	145	HGR	\$B8	184	DEF
\$6B	107	k	\$92	146	HCOLOR =	\$B9	185	POKE
\$6C	108	l	\$93	147	HPlot	\$BA	186	PRINT
\$6D	109	m	\$94	148	DRAW	\$BB	187	CONT
\$6E	110	n	\$95	149	XDRAW	\$BC	188	LIST
\$6F	111	o	\$96	150	HTAB	\$BD	189	CLEAR
\$70	112	p	\$97	151	HOME	\$BE	190	GET
\$71	113	q	\$98	152	ROT =	\$BF	191	NEW
\$72	114	r	\$99	153	SCALE =	\$C0	192	TAB(
\$73	115	s	\$9A	154	SHLOAD	\$C1	193	TO
\$74	116	t	\$9B	155	TRACE	\$C2	194	FN
\$75	117	u	\$9C	156	NOTRACE	\$C3	195	SPC(
\$76	118	v	\$9D	157	NORMAL	\$C4	196	THEN
\$77	119	w	\$9E	158	INVERSE	\$C5	197	AT
\$78	120	x	\$9F	159	FLASH	\$C6	198	NOT
\$79	121	y	\$A0	160	COLOR =	\$C7	199	STEP
\$7A	122	z	\$A1	161	POP	\$C8	200	+
\$7B	123	{	\$A2	162	VTAB	\$C9	201	-
\$7C	124		\$A3	163	HIMEM:	\$CA	202	*
\$7D	125	}	\$A4	164	LOMEM:	\$CB	203	/
\$7E	126	-	\$A5	165	ONERR	\$CC	204	^
\$7F	127	{RUB}	\$A6	166	RESUME	\$CD	205	AND
\$80	128	END	\$A7	167	RECALL	\$CE	206	OR
\$81	129	FOR	\$A8	168	STORE	\$CF	207	>
\$82	130	NEXT	\$A9	169	SPEED =	\$D0	208	=
\$83	131	DATA	\$AA	170	LET	\$D1	209	<

Hex	Dec	Chr	Hex	Dec	Chr	Hex	Dec	Chr
\$D2	210	SGN	\$DB	219	RND	\$E4	228	STR\$
\$D3	211	INT	\$DC	220	LOG	\$E5	229	VAL
\$D4	212	ABS	\$DD	221	EXP	\$E6	230	ASC
\$D5	213	USR	\$DE	222	COS	\$E7	231	CHR\$
\$D6	214	FRE	\$DF	223	SIN	\$E8	232	LEFT\$
\$D7	215	SCRN(\$E0	224	TAN	\$E9	233	RIGHT\$
\$D8	216	PDL	\$E1	225	ATN	\$EA	234	MID\$
\$D9	217	POS	\$E2	226	PEEK			
\$DA	218	SQR	\$E3	227	LEN			

TOKEN VIEWER PROGRAM

The following program will go charging through your tokens and present them on the screen for you. What happens is this: the statement in line 20 is changed by POKEing in various token values into the machine-language listing of your BASIC program (2061 is the address where the statement in line 20 lives). Each time through the loop, we see the different token by LIST 20. Neat, huh?

```

10 TEXT : HOME
20 REM
30 SPEED = 175
40 FOR X = 33 TO 234
50 POKE 2061,X
60 LIST 20
70 NEXT X
80 SPEED = 255

```

After you RUN the program, line 20 will be MID\$ instead of REM.

BYTE ZAP AND HEX

Welcome to the world of tracks, sectors, and bytes. *Byte Zap* is a program that gets you into the nitty-gritty of data storage, letting you examine and change one byte at a time. We will cover some of the more everyday uses of this program, how it works, and how to use it as a tool. If you would like to get into disk repair and copy protection, please do yourself a favor and buy a copy of *Beneath Apple DOS* at your local computer store. This book says it all; half of it is still way over Bert's head (and Bert wrote *Byte Zap*!), but both of us are always referring to it and learning from it.

SOME NOTES ABOUT HEX

Hex is a big pain; don't let anyone tell you differently. It is, however, a *beautiful* number system to use if you happen to be a computer. Most utilities that work with tracks and sectors use only hex. *Byte Zap* gives you the option of using hex *or* decimal. Hex numbers in the program are preceded by a dollar sign and followed by the decimal equivalent in parentheses.

The hexadecimal system is constructed on a *sixteen* base instead of a ten base (like the decimal system). If we had eight fingers on each hand it would be easier. Just remember, in hex you count 0-1-2-3-4-5-6-7-8-9-A-B-C-D-E-F-10-11-12, etc. (that's decimal 0-18). There's a little hex converter program on your PEEKs and POKEs chart that will convert numbers from 0 to 255 (\$00 to \$FF). All of the numbers your *Apple* uses are in that range.

HOW APPLE DATA ARE STORED

Let's start big and work our way down. A standard Apple floppy is divided into thirty-five concentric *tracks* numbered 0-34 (\$00-\$22). Each

track is further divided into sixteen *sectors* numbered 0–15 (\$00–\$0F). Bytes are made up of *bits*, too, but this is where we get off.

A *byte* is a piece of magnetic data we don't really understand; just think of each byte as a *number* from 0 to 255. This number can represent just that, a number (or part of a number), an ASCII character (letters, numbers, etc.), a BASIC token (command words like PRINT, GOSUB, etc.), or some kind of machine-language command. These bytes are *written* or rewritten onto a disk when you SAVE a program, RENAME a file, or perform any of the *writing* functions of DOS (Apple's Disk Operating System). When you LOAD a program, you are only *reading* bytes off the disk and into memory, not changing anything on the disk.

HOW BIG IS THAT PROGRAM?

One way to tell the size of a program is to SAVE it and look in the catalog. Every sector, not counting the first one, is about 256 bytes in size. Divide the sector count by 4 and you have roughly the size of the program in K's. For example, a 48-sector file occupies about 12K or 12,000 bytes.

Another way to tell a program's size is to LOAD it into memory and then subtract its start address from its end address, like so:

```
PRINT (PEEK(175) + PEEK(176)*256) - (PEEK  
      (103) + PEEK(104)*256)
```

This typing exercise will tell you the number of bytes your program occupies. The start address is usually 2049, so you can use 2049 instead of the second pair of PEEKs.

THE BYTE REPORT

For beginners only: Perhaps the following discussion of disks and data will help in your use of Pro-Byter. If not, at least we tried.

A BYTE IS A NUMBER

All Apple II data exist in memory or on disk as a series of bytes, each with its own value ranging from 0 to 255. These bytes “look” much the same on disk as they do in memory. What they really are is beyond us; we just know that they exist. And they do the job.

Each byte is made of eight bits (the 1’s and 0’s that you hear so much about). You don’t need to know about bits now. Forget bits.

<i>byte</i>	<i>byte</i>	<i>byte</i>	<i>byte . . .</i>
value 10	value 99	value 123	value 0 . . .

Each byte has some value, 0–255.

HEX

If you’re going to get into this byte stuff at all, you’re going to have to learn about hexadecimal numbers. Just pretend like you have eight fingers on each hand, numbered 0-1-2-3-4-5-6-7-8-9-A-B-C-D-E-F. A dollar sign in front of the number makes it clear that we’re counting in hex instead of decimal. This chart serves as a “hex converter” for numbers 0–255 (\$00–\$FF). After \$FF (or \$00FF) comes \$0100 (256). . . . Get the picture? Good.

HEX EQUIVALENTS

0-255 (\$00-\$FF)

0 \$00	32 \$20	64 \$40	96 \$60	128 \$80	160 \$A0	192 \$C0	224 \$E0
1 \$01	33 \$21	65 \$41	97 \$61	129 \$81	161 \$A1	193 \$C1	225 \$E1
2 \$02	34 \$22	66 \$42	98 \$62	130 \$82	162 \$A2	194 \$C2	226 \$E2
3 \$03	35 \$23	67 \$43	99 \$63	131 \$83	163 \$A3	195 \$C3	227 \$E3
4 \$04	36 \$24	68 \$44	100 \$64	132 \$84	164 \$A4	196 \$C4	228 \$E4
5 \$05	37 \$25	69 \$45	101 \$65	133 \$85	165 \$A5	197 \$C5	229 \$E5
6 \$06	38 \$26	70 \$46	102 \$66	134 \$86	166 \$A6	198 \$C6	230 \$E6
7 \$07	39 \$27	71 \$47	103 \$67	135 \$87	167 \$A7	199 \$C7	231 \$E7
8 \$08	40 \$28	72 \$48	104 \$68	136 \$88	168 \$A8	200 \$C8	232 \$E8
9 \$09	41 \$29	73 \$49	105 \$69	137 \$89	169 \$A9	201 \$C9	233 \$E9
10 \$0A	42 \$2A	74 \$4A	106 \$6A	138 \$8A	170 \$AA	202 \$CA	234 \$EA
11 \$0B	43 \$2B	75 \$4B	107 \$6B	139 \$8B	171 \$AB	203 \$CB	235 \$EB
12 \$0C	44 \$2C	76 \$4C	108 \$6C	140 \$8C	172 \$AC	204 \$CC	236 \$EC
13 \$0D	45 \$2D	77 \$4D	109 \$6D	141 \$8D	173 \$AD	205 \$CD	237 \$ED
14 \$0E	46 \$2E	78 \$4E	110 \$6E	142 \$8E	174 \$AE	206 \$CE	238 \$EE
15 \$0F	47 \$2F	79 \$4F	111 \$6F	143 \$8F	175 \$AF	207 \$CF	239 \$EF
16 \$10	48 \$30	80 \$50	112 \$70	144 \$90	176 \$B0	208 \$D0	240 \$F0
17 \$11	49 \$31	81 \$51	113 \$71	145 \$91	177 \$B1	209 \$D1	241 \$F1
18 \$12	50 \$32	82 \$52	114 \$72	146 \$92	178 \$B2	210 \$D2	242 \$F2
19 \$13	51 \$33	83 \$53	115 \$73	147 \$93	179 \$B3	211 \$D3	243 \$F3
20 \$14	52 \$34	84 \$54	116 \$74	148 \$94	180 \$B4	212 \$D4	244 \$F4
21 \$15	53 \$35	85 \$55	117 \$75	149 \$95	181 \$B5	213 \$D5	245 \$F5
22 \$16	54 \$36	86 \$56	118 \$76	150 \$96	182 \$B6	214 \$D6	246 \$F6
23 \$17	55 \$37	87 \$57	119 \$77	151 \$97	183 \$B7	215 \$D7	247 \$F7
24 \$18	56 \$38	88 \$58	120 \$78	152 \$98	184 \$B8	216 \$D8	248 \$F8
25 \$19	57 \$39	89 \$59	121 \$79	153 \$99	185 \$B9	217 \$D9	249 \$F9
26 \$1A	58 \$3A	90 \$5A	122 \$7A	154 \$9A	186 \$BA	218 \$DA	250 \$FA
27 \$1B	59 \$3B	91 \$5B	123 \$7B	155 \$9B	187 \$BB	219 \$DB	251 \$FB
28 \$1C	60 \$3C	92 \$5C	124 \$7C	156 \$9C	188 \$BC	220 \$DC	252 \$FC
29 \$1D	61 \$3D	93 \$5D	125 \$7D	157 \$9D	189 \$BD	221 \$DD	253 \$FD
30 \$1E	62 \$3E	94 \$5E	126 \$7E	158 \$9E	190 \$BE	222 \$DE	254 \$FE
31 \$1F	63 \$3F	95 \$5F	127 \$7F	159 \$9F	191 \$BF	223 \$DF	255 \$FF

BYTES IN MEMORY

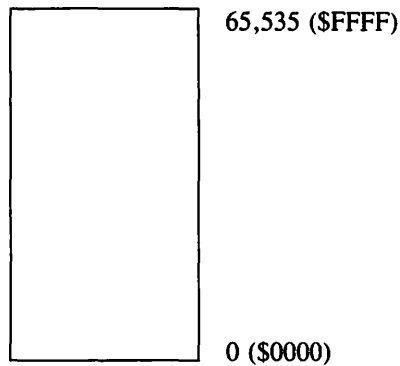
A 64K Apple's memory is simply a long series of bytes, numbered from 0 to 65,535 (\$0000-\$FFFF). Each byte or series of bytes may be interpreted

differently, depending on the program or language that is doing the interpreting.

<i>byte #0</i>	<i>byte #1</i>	<i>byte #2</i>	<i>byte #3 . . .</i>	<i>byte #65,535</i>
value 10	value 99	value 123	value 55 . . .	value 127

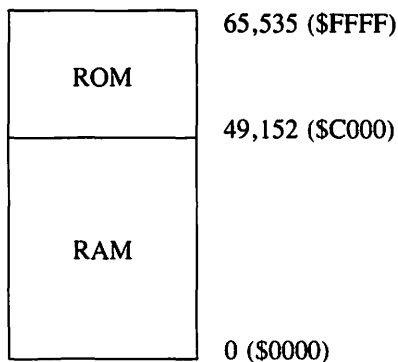
Memory is made up of bytes.

Memory is often illustrated with a rectangle or “map” with byte #0 at the bottom and byte #65,535 at the top:



Think of memory as a tower of bytes.

The first 49,152 bytes in memory are changeable (“Random Access Memory”). The last 16,384 are unchangeable (“Read Only Memory”).



ROM and RAM.

PEEKING AND POKING

You can use your Apple to “peek” at any byte in memory, from lowly byte #0 to big byte #65,535. For example, type:

```
PRINT PEEK(800)
```

You will see an answer printed on the screen. This number, always in the range 0–255, is the value of byte #800.

You can change, or “poke,” any byte from #0 to #49,152. Let’s POKE byte #800. Type:

```
POKE 800,123
```

Now type:

```
PRINT PEEK(800)
```

There. You have changed the value of byte #800. Try changing a byte greater than 49,152—you can POKE the value (or so it seems) but it won’t stick.

Wait a minute! You can PEEK all you want, but you really shouldn’t do any POKEing unless you (think you) know what you’re doing. The worst that can happen is that you’ll ruin a disk or destroy DOS; you can’t hurt your Apple by pressing keys. Ruined disks are no problem if you’ve made a backup copy first. Destroyed DOS is “repaired” by simply rebooting.

CHAPTER 4

APPLESOFT TIPS

"QUOTES "WITHIN" QUOTES"

You probably thought you couldn't put quote marks inside of a quote statement. Well, run this program, then list line 20. The carets will have been replaced by quotes. Trouble is you can't re-RUN the program or reencounter line 20 without hanging.

```
10 HOME : ONERR GOTO 6000
20 A$ = "THIS IS NOT AS ^USEFUL^ AS IT
   LOOKS.": GOSUB QUOTE
30 PRINT A$: LIST 20: END
6000 ER = PEEK (220) + PEEK (221) * 256:
   FOR EE = ER - 2 TO 1 STEP - 1: P = PEEK
   (EE): POKE EE, P - 60 * (P = 94): IF P
   < > 34 THEN NEXT
6001 POKE ER + 1, 178: RESUME
```

The GOSUB QUOTE is illegal, causing an ONERR jump to line 6000.

Line 6001 changes the GOSUB to a REM, so the program can RESUME. The next program has a better way of printing quotes.

GOSUB QUOTE CONVERSION

In this program the string will be printed on the screen, and then the carets will be converted to quote marks. Remember, carets are those little pointy things.

```

5   TEXT : HOME : NORMAL
10  PRINT "PUT THIS IN ^QUOTES^ FOR ME.":
    GOSUB 1000
20  PRINT "AND ^THIS^ AND ^THIS^ AND
    ^THIS^...": GOSUB 1000
100 END
1000 VTAB PEEK (37):L = PEEK (40) + PEEK
    (41) * 256: FOR I = L TO L + 39: IF
    PEEK (I) = 222 THEN POKE I,162
1001 NEXT : PRINT : RETURN

```

SECRET DECODER

To translate what the two guys in the middle are saying, type in line 20 as shown here and add a line 10 that sets A\$ equal to what it looks like each man said.



```
20 HOME : FOR L = 1 TO LEN(A$) : A = ASC  
  (MID$(A$,L,1))  
25 A = A - (A>64) - 19 * (A = 32) : PRINT  
  CHR$(A); : NEXT
```

Remember HAL, the wise computer in the movie *2001: A Space Odyssey*? Rumor has it that he got his name using a program something—but not quite—like this.

?LINE MISMATCH



Running this program will produce a “?Syntax Error in 10” error message.

```
10 DATA 1,2,3,4,FIVE,6,7,8  
20 READ A : PRINT A : GOTO 20
```

Wrong! Line 10's syntax is perfectly legal. The problem is in line 20, where we are trying to read a string (FIVE) as a numeric variable (A). Next time Applesoft accuses you of an improper DATA statement, this might be your problem.

SOME GPLE STUFF

The next two tricks are accomplished with *GPLE* (*Global Program Line Editor*). If you don't have *GPLE*, go get it and come back, or just forget the whole thing!

CONTROL-M COLUMNIZER

GPLE's Edit Mode lets you insert carriage returns (Control-M's) into PRINT statements. Notice this program line:

```
10 HOME : PRINT "THESE" : PRINT "WORDS" :  
    PRINT "WILL" : PRINT "PRINT" : PRINT "IN  
    A" : PRINT "COLUMN"
```

Try this (in Edit Mode):

```
EDIT 10 <RETURN>
```

```
10 HOME : PRINT "THESEmWORDSmWILLm  
    PRINTmIN AmCOLUMN."
```

To insert control characters, see Control-O (Override).

The lowercase *m*'s are Control-M's. Both preceding program lines will produce the same result. The second method will print and LIST in a column; it also takes up just a bit less memory than normal.

REM TRICKS

By injecting some backspaces (Control-H's) and carriage returns (Control-M's) into REM statements, you can make some nice headings for your Applesoft program listings.

```

10 REM
PROGRAM NAME
-----
20 PRINT "START OF PROGRAM"

```

How It's Done

```

10 REMmmPROGRAM NAMEm-----
20 PRINT "START OF PROGRAM"
30 REM m=CONTROL=M


```

Another Example

```

-----
PROGRAM NAME
-----
20 PRINT "START OF PROGRAM"

```

 Notice how the line number is hidden by hyphens. This is done with control-H's (backspaces), which back the cursor up so the hyphens write over the line number!

How It's Done

```

10 REMhhhhhhhhh-----PROGRAM NAME
   m-----
20 PRINT "START OF PROGRAM"
30 REM h=CONTROL=H, m=CONTROL=M

```

You must adjust the number of Control-H's according to the number of digits in the line number.

CONTROL-J REMS

```

10 REM CONTROL=J REMS ARE EASY
15 REM LOOK AT THIS

```

```
20 REM (2 CTRL-J'S AFTER "REM")
30 REM (GPLE NOT NECESSARY)
```

SEE YA LATER TRUNCATOR

If you want to delete some program lines numbered greater than 63999, add line 5000 (it can be any number you want) and GOTO it.

```
5      GOTO 5000: REM TEMPORARY LINE
10     REM THIS LINE WILL STAY
5000   X = PEEK (121) + PEEK (122) * 256:
      POKE X + 1,0: POKE X + 2,0:X = X + 3:
      POKE 175,X -INT (X / 256) * 256: POKE
      176, INT (X / 256): END : REM ADD
      THIS LINE & GOTO IT FROM
      ANOTHER PROGRAM LINE
6000   REM THIS LINE DISAPPEARS
65535  REM SO WILL THIS ONE
```

NO-WORD-BREAK TIP

Printing text on the 40-column screen without breaking words at the right margin isn't always easy, even with *GPLE*. The following subroutine at line 1000 will do the dirty work for you, breaking words only at spaces or hyphens. If you don't like what you see, you can go back and insert a hyphen or two with *GPLE* and then re-RUN. This program also converts lowercase to uppercase on non-//e's and //c's.

NOW YOU SEE IT . . .

In case you want to edit a line, but you don't know what the line number is, set `SPEED=1` and `LIST`. Aha! Set `SPEED=255` to regain normal speed. (Escape `L` on *GPLe* does this for you.)

ABOUT GPLE

And speaking of the *Global Program Line Editor*. . . In case you don't know what *GPLE* does, you should! First, it allows you to edit program lines in no time compared to what you're probably doing now. Second, it has an "Escape Create" function that lets you program any key to perform any function. For example, Escape L can LIST a program, Escape P can type the word "PRINT" for you, Escape N can type "NOW IS THE TIME FOR ALL GOOD MEN TO GOSUB 86," etc., etc.

EXTRA IMPORTANT TIP



EXTRA IMPORTANT TIP

If you own one of the newer Apples with the grey keyboard, you will find that underneath the keyboard panel is one of the most important features you'll ever use. Try it. You'll be amazed! Remember, you read this here!

TEXT WINDOW TRIX

Dewindow

To set your Apple text window to normal, you could type POKE 32,0 : POKE 33,40 : POKE 34,0 : POKE 35,24. Or you could type:

TEXT

CONTROL EQUIVALENTS

Hitting Control-M is the same as hitting RETURN. Control-H is a back-space. Control-U is a forward space. Control-[is Escape. Control-J moves the cursor down one line. Just thought we'd mention it.

USER OPTIONS

In Applesoft, there are different ways to say the same thing:

<i>This . . .</i>	<i>is the same as . . .</i>
LET X = 7	X = 7
IF X > 0 THEN PRINT	IF X THEN PRINT
IF X = 0 THEN END	IF NOT X THEN END
IF X = 8 THEN GOTO 10	IF X = 8 THEN 10
IF X = 2 THEN INVERSE	POKE 50,255-192 * (X=2)
? 2 + 2	PRINT 2 + 2
PRINT "HELLO."	PRINT "HELLO.
NEXT X	NEXT

APPLE'S HEX CONVERTER

You can convert a hex number to decimal in the monitor. Say you want to convert \$0564 to decimal. From Applesoft, enter the monitor with CALL -151, and type:

```
*45: 05 64 N ED24G
```

Your decimal answer, 1380 in this case, will appear! Use the N ED24G for converting any number (from Applesoft only).

You can add and subtract hex in the monitor, too. Just type the equation, say "AB + 3E" (don't type PRINT), and hit RETURN. Apple even throws in a free equal sign.

SAVE/1 SAVE/2 SAVE/3 . . .

Make progressive backup copies as you program, but give each a different name (PROGRAM/1, PROGRAM/2, etc.). That way, you can back up a step or two if some permanent damage occurs. A new SAVE every ten minutes or so is a good idea.

? REMOVER

Nothing rattles my chips as much as seeing a \$40 program ask me "TYPE YOUR ANSWER HERE?" The question mark shouldn't be there, and with the Apple, it doesn't have to be. The question mark is produced in this program:

```
10 HOME : PRINT "TYPE YOUR ANSWER HERE"  
20 PRINT " (8 LETTERS OR LESS)"  
30 VTAB 1 : HTAB 22  
40 INPUT ANS$
```

To eliminate the ?, make line 40—

```
40 INPUT ":";ANS$
```

Or, if you are completely antipunctuation:

```
40 INPUT " ";ANS$
```

STOP

Applesoft's STOP works just like END, but it tells you the line number where your program stopped. Use STOP for debugging.

SEMI-END

If you want to end a program with no distracting flashing cursor, try one of these methods.

If you're in hi-res, make your last line:

```
1000 VTAB 1 : END
```

The flashing cursor is on the screen, but hidden by hi-res.

This method won't work in lo-res or text; so try:

```
1000 GOTO 1000
```

You could also program the cursor to come back when any key is pressed:

```
1000 K=PEEK (-16384) : IF K < 128 THEN 1000  
1001 POKE -16368, 0 : END
```

Or, if you're really smart and using Applesoft:

```
1000 WAIT -16384,128
1001 POKE -16368,0 : END
```

The POKE in 1001 clears the keyboard buffer. There's nothing worse than a stuffed buffer.

CRUISING THROUGH APPLESOFT

In the DOS section there's a cruise through DOS to find DOS's vocabulary. Well, here's a good program that does the same in good old FP!

```
10 REM ** APPLESOFT CRUISE **
20 START = 53456:FIN = 54116
30 TEXT : HOME : INVERSE : PRINT " APPLESOFT
   COMMANDS: ": NORMAL : PRINT "-----
   -----"
40 FOR X = START TO FIN:P = PEEK (X)
50 IF P = 7 THEN INVERSE : PRINT "G";:
   NORMAL
60 PRINT CHR$ (P);: IF P < 128 THEN 110
70 N = N + 1
80 IF ERR THEN HTAB 1 + 22 * ((N - INT (N /
   2) * 2) = 0)
90 IF NOT ERR THEN HTAB 1 + 8 * (N - INT (N
   / 5) * 5)
100 IF PEEK (36) = 0 THEN PRINT
110 IF X = 53854 THEN ERR = 1: INVERSE :
   FOR I = 1 TO 500: NEXT : PRINT : PRINT
   : PRINT " APPLESOFT ERROR MESSAGES: ":
   NORMAL : PRINT "-----
   -----"
120 NEXT X
```

BETTER NEW ON OLD DOS

If you've been doing some weirdness on your Apple and want to start a new program, FP (DOS 3.3 only) is more efficient than NEW for clearing memory and resetting pointers. Another way to "delete" most Applesoft programs is to POKE 2049,0: POKE 2050,0. Put these POKEs at the end of a program and it will erase itself. However, if you have ProDOS up on your Apple //c, it'll laugh at you if you try FP or INT. (Use the System Master from an Apple //e to get INT and FP cranking on your //c.)

PRINT SPC

If you're writing programs for others to read, always use

```
PRINT SPC(10)
```

instead of

```
PRINT "          "
```

The reasons are obvious.

MOD

Bert learned how to program in Integer BASIC, and he has since converted his efforts to Applesoft. The INT feature Bert misses most is the MOD, or remainder, function—3 MOD 2 is 1, 99 MOD 10 is 9, etc. This is a very useful function that can be simulated in Applesoft with many more keystrokes. Here is a MOD program in Integer.

```
10 REM INTEGER BASIC
20 FOR X = 1 TO 100
30 IF X MOD 3 = 0 THEN PRINT X
40 NEXT X: END
```

To convert to Applesoft, just change line 30 to the following:

```
30 IF X - INT(X/3)* 3 = 0 THEN PRINT X
```

FREE MOD

This isn't an Applesoft tip, but it's a neat way to get MOD and a great language for your Apple *free*! One of the hottest languages for microcomputers is FORTH. It's fast, weird, wonderful, and several versions are in the public domain. Go to your nearest Apple user's group and get your copy. FORTH has MOD, naturally.

A\$ = ??

Sometimes, strings can contain hidden characters like leading or trailing spaces or control characters. Setting screen output to inverse will reveal hidden spaces. Type:

```
A$="FISH " : PRINT A$
```

The Apple will hand you a four-letter word apparently *without* the trailing space. Now try INVERSE : PRINT A\$. You can now see all *five* characters, including the space. Another way to search for hidden characters is to PRINT A\$; LEN(A\$). If the number is higher than the number of visible characters in the word, you can suspect some hidden control characters or spaces.

A DIFFERENT APPROACH

If a programming solution is getting you down, try looking at it from a whole new angle. And if you're *really* stuck, start over.

TOKEN TRICKS!

In the last chapter there was a chart of Applesoft's tokens along with a program to list the tokens. Here are two more programs. Both zap their first lines and then list them. Both are nice demos of Applesoft's token system.

```
10 GOTO 256
256 TEXT : POKE 2054,0: FOR I = 128 TO 234:
    POKE 2051,I: POKE 2053,I: VTAB PEEK
    (37) - 1: LIST - 255: NEXT : POKE 2051,
    10: POKE 2053,171: POKE 2054,50
```

```
10 REM <-LOOK AT THIS.....
20 HOME :Q = - 16384
30 START = PEEK (103) + PEEK (104) * 256
40 VTAB 10: PRINT "THIS IS LINE 10--"
50 VTAB 15: PRINT "HIT ANY KEY TO END."
60 FOR X = 234 TO 32 STEP - 1
70 POKE START + 4,X
80 VTAB 11
90 CALL - 868: LIST 10
100 FOR I = 1 TO 40
110 IF PEEK (Q) > 127 THEN 130
120 NEXT I,X
130 POKE - 16368,0: LIST
```

THREE APPLE ERROR MESSAGES

*** SYNTAX ERR	Integer
?SYNTAX ERROR	Applesoft
SYNTAX ERROR	DOS

ERROR SILENCER

Many beginning programmers don't know about Applesoft's ONERR command. Put an ONERR GOTO 999 (or whatever line you want) at the beginning of your program. Anytime (almost) an error occurs, instead of crashing with an error message, your program will jump to line 999 which could instruct the program to jump to your menu, or start over. While you are programming, you will *want* error messages, so stick a REM in front of your ONERR command; or put a POKE 216,0 in the routine you are working on. This POKE disables ONERR.

?ERROR?

When you encounter an error in hi-res, you get a beep, but usually no visible error message. Just hit Reset or type TEXT to reveal the message.

ONERR ERROR LISTER

Here's a good one. This program has an intentional error in line 12345. Type it *exactly* as you see it. Be careful; a typo in line 60000 can really

create a big mess when you RUN the program (we know—we've been there).

```
10      ONERR GOTO 60000
20      REM YOUR PROGRAM STARTS HERE
12345 PRINT :PRINT :PRINT
12346 REM NOTE ERROR TEST IN LINE ABOVE
59999 END
60000 T=256: E=PEEK(220)+T*PEEK(221): E=E+5
      *(PEEK(E)=0):
      V=PEEK(E): POKEE,207:
      L$=RIGHT$("0000"+STR$(PEEK(218)+T*
      PEEK(219)),5):
      L=PEEK(121)+T*PEEK(122)+49: FOR I=1 TO 5:
      POKEL+I,ASC(MID$(L$,I,1)): NEXT: LIST
12345: POKEE,V:
      POKE216,0: RESUME
```

SAVE the program and then RUN it. What you will get is the usual error message *and* a listing of the offending line with an arrow (greater-than sign) pointing to the statement that caused the error! No more searching; a *big* help if you're like us and use six pounds of statements in each line. Add lines 10 and 60000 (could be any line numbers) to one of your programs in progress. It's a great debugger!

ERROR TOKENS

Tokens are machine-language equivalents for the commands and characters in your Applesoft programs. See chapter 3 for more details. The tokens we don't understand are 235–255 (that's why you won't find them in chapter 3). Type this program and RUN it.

```
5 HOME
6 FOR X = 236 TO 252: POKE 2053,X: REM
  CHANGE LINE 5
7 CALL - 998: CALL - 998: LIST 5: NEXT :
  POKE 2053,178: REM CALL-998 MOVES CURSOR
  UP A LINE
```

Line 6 changes line 5 and line 7 LISTs it. But why all those *error messages*? Who knows?

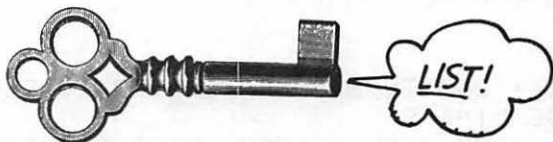
ONERR TELL ME

ONERR GOTO is great for keeping a program on its feet and running when an error is encountered. But you want to *catch* those errors if you are programming. You should either take the ONERR statement temporarily out of your programs or put line 1000 (see following) in. Actually, this line could be put into all of your programs.

```
10  G$ = CHR$ (7): ONERR GOTO 1000
20  X = X + 1: NORMAL : PRINT X
30  IF X = 19 THEN PRIN "NINETEEN": REM
    ERROR: MISPELLED WORD
40  IF X = 30 THEN GOTO 99: REM ERROR
    NONEXISTENT LINE
50  IF X < 35 THEN 20
60  END
1000 INVERSE : PRINT G$;" ERROR #"; PEEK
    (222);" ON LINE "; PEEK (218) + PEEK
    (219) * 256;".": GOTO 20
```

RUN the program and compare the error numbers with the ONERR codes in your Applesoft manual.

ONE-KEY COMMANDS!



Type this:

```
POKE 1013,76 : POKE 1014,110 : POKE 1015,165
```

Now, the command “&” <RETURN> will CATALOG! Change the numbers after the commas to 76, 165, and 214 and “&” will LIST; 76, 18, and 217 will make “&” RUN; 76,112, and 214 will make “&” clear all variables to zero. When you type “&” (that’s called “ampersand”), your Apple jumps to location 1013 (\$3F5) to see what’s happening there. The POKES discussed here tell the Apple what to do.

PLEASE PARSE THE VARIABLES

Applesoft can do some weird things with variables. For instance, the statement,

```
IF X = T AND Y = 0 THEN PRINT
```

will be respaced or “parsed” to read,

```
IF X = TAN DY = 0 THEN PRINT
```

and cause a “?SYNTAX ERROR,” because TAN is a reserved word meaning tangent. To prevent this, rename the variable or put it in parentheses:

```
IF X = (T) AND Y = 0 THEN PRINT
```

More details on parsing are somewhere in your Applesoft manual. We don’t have time right now to check the page number.

SLOT SEARCH

We need your help! Here’s a program that reads your Apple’s slots to see what’s in them. We know that if a disk controller card is in a slot, the value

for that slot is 162. The three printers we have checked out have returned a 24. A Novation Super-Cat modem is a 255. Please let us know your results when you RUN this program with your equipment. Apparently, if a slot is empty, the value keeps changing, so look for a slot with an unchanging number. Of course, the Apple //c has to be different. The internal disk drive thinks it's in slot 6 and none of the other values blink if their "slots" are empty; they just say EMPTY.

```
5    REM
    =====
    SLOT SEARCH
    =====
10   PRINT CHR$(21) : HOME : FOR X = 1 TO 20
    : FOR I = 1 TO 7 : REM (NOT 0 TO 7)
20   N = 49152 + 256 * I : PN = PEEK (N)
30   VTAB 4 + 2 * I : HTAB 1 : PRINT "SLOT
    ";I;": PEEK(";N;")=";PN; SPC( 2)
40   VTAB 4 + 2 * I : HTAB 26
50   B(I) = A(I):A(I) = PN: IF A(I) < > B(I)
    THEN CALL - 868: PRINT "(EMPTY)": GOTO
    500
60   INVERSE
100  IF PN = 162 THEN PRINT "DISK DRIVE"
110  IF PN = 24 THEN PRINT "PRINTER"
500  NORMAL : NEXT I,X: VTAB 22
```

PLAY IT SAFE

SAVE each program you type before you RUN it. Sometimes a bug or typo will cause a program to self-destruct when it's RUN.

THE INCREDIBLE SHRINKING NUMBERS

This program demonstrates Apple's handling of small numbers.

```

10 X = 1
20 X = X / 10 : PRINT X : GOTO 20

```

And *big* numbers:

```

10 X = 1
20 X = X * 10 : PRINT X : GOTO 20

```

We don't know how they came up with the "E." Probably stands for *Extrahuge* and *Extrateensy*, right?

NUMBER UNCRUNCHER

Have you ever wanted to dissect a decimal number into its separate digits? Here are two ways to do it:

```

4  TEXT : HOME : INVERSE
5  PRINT "DIGIT DISSECTOR": NORMAL : PRINT
   "<S> STRING METHOD (FASTER)": PRINT
   "<M> MATH METHOD (SLOWER)": PRINT : GET
   A$: IF A$ = "S" THEN 100
8  IF A$ < > "M" THEN END
9  REM

   DIGIT DISSECTION (MATH)

10 INPUT "ANY WHOLE NUMBER: "; N: N = INT
   (N): L = LEN ( STR$ (N)): IF L > 9 THEN
   10
20 FOR X = 1 TO L: E = L + 1 - X: MOD = INT
   (N / 10 ^ E) * 10 ^ E: MOD = N - MOD: D(X)
   = INT (MOD / (10 ^ (E - 1))): NEXT
30 FOR X = 1 TO L: PRINT "DIGIT #"; X; ":
   "; D(X): NEXT
40 PRINT : GOTO 5
99 REM

```

DIGIT DISSECTION (STRINGS)

```

100 INPUT "ANY WHOLE NUMBER:";N:N = INT
    (N):N$ = STR$ (N):L = LEN (N$): IF L >
    9 THEN 100
110 FOR X = 1 TO L:D(X) = VAL ( MID$
    (N$,X,1)): NEXT
120 FOR X = 1 TO L: PRINT "DIGIT #";X;":
    ";D(X): NEXT
130 PRINT : GOTO 5

```

The first method does the dissecting mathematically, and the second does it with strings. The latter is much, *much* faster.

HOW MUCH DATA?

If you wonder how many items you have in a DATA statement, don't count them; let your Apple count them. Temporarily insert a giant loop at the start of your program, like:

```

1 FOR X = 1 TO 1000000 : READ X$ : NEXT

```

RUN it. Sooner or later, you'll get an "?OUT OF DATA" error message. Now type PRINT X-1 <RETURN>. The value you get for X-1 will tell you the number of items of DATA you have.

If you're smarter than we are, you'll have an error *trap* that will automatically do it for you. Furinstance . . .

```

1  ONERR GOTO 3
2  X = 1 + 1 : READ X$ : GOTO 2
3  PRINT X - 1 " ITEMS " : END
10 FOR X = 1 TO N : READ X$ : NEXT
50 DATA MAN, WOMAN, FISH, BICYCLE

```

DELAY LOOPS

When you want a program to stop and wait, make your Apple sit there and count to some big number (your Apple doesn't mind; in fact, it's *good* for him/her). Insert a statement like:

```
FOR X = 1 TO 2000 : NEXT X
```

Every 1000 is approximately a one-second wait. If your program is going to pause many times, use a delay loop as a subroutine (GOSUB) to save memory.

GET PROMPT

Type PRINT PEEK(51). Better yet, PRINT CHR\$(PEEK (51)) if you're using Applesoft. This little exercise will show you what language you are using (in case you didn't know!) by printing its prompt character. Integer's ">" is represented at location 51 in memory by a value of 190. Applesoft's "]" is a 221. Try to POKE in a different number. . . . Forget it; you can't. Another location that tells your language is 43702. Type PRINT PEEK(43702). If you're in Integer, you'll get in trouble since 43702 is larger than 32767. Use PEEK (-21834) instead; you'll get a zero. Applesoft ROM gives you a 64; Applesoft RAM, a 128.

BETTER GETTERS

Applesoft's blasted INPUT function won't accept (1) commas, (2) colons, or (3) leading spaces. Try answering an INPUT with a comma or colon in the answer and you get the famous "?EXTRA IGNORED" statement. It doesn't make a lot of sense the first time you see it. Well, let's do

something about the problem. Here are two little inputters. Both feature different types of inputs that accept (1), (2), *and* (3)!!

```

10  REM
    =====
    COMMA INPUTTER #1
    =====
20  PRINT CHR$ (21): HOME :BKSP$ = CHR$
    (8):CR$ = CHR$ (13): Q$ = CHR$ (34)
30  PRINT "WHAT'S YOUR NAME? ";
40  NAME$ = " ": GOTO 50
50  GET LTR$: IF LTR$ = BKSP$ THEN NAME$ =
    LEFT$ (NAME$, LEN (NAME$) - 1): NORMAL
    : PRINT LTR$: SPC( 1);
60  IF LTR$ = CR$ THEN 100
70  IF LTR$ > = " " THEN NAME$ = NAME$ +
    LTR$
80  INVERSE : PRINT LTR$;
90  GOTO 50
100 IF LEN (NAME$) THEN EMAN$ =
    "" : FOR X = LEN (NAME$) TO 1 STEP -1:
    EMAN$ = EMAN$ + MID$ (NAME$, X, 1): NEXT X
110 NORMAL : PRINT : PRINT : PRINT Q$; NAME
    $;Q$" SPELLED BACKWARDS": PRINT "IS ";:
    PRINT Q$;EMAN$;".":Q$: PRINT
120 NAME$ = "": GOTO 30
130 GOTO 40

10  REM
    =====
    COMMA INPUTTER 2
    =====
20  PRINT CHR$ (21): HOME
30  PRINT "TYPE YOUR NAME ": PRINT "(LAST,
    FIRST): ";
40  CALL - 657
50  NAME$ = " ": FOR X = 512 TO 767:C$ =
    CHR$ ( PEEK (X)): IF C$ < > CHR$ (141)
    THEN NAME$ = NAME$ + C$: NEXT
60  L = LEN (NAME$): FOR X = 1 TO L: IF
    MID$ (NAME$,X,1) < > CHR$ (172) THEN
    NEXT : PRINT "PLEASE SEPARATE YOUR
    NAMES WITH A COMMA.": GOTO 30
70  LAST$ = LEFT$ (NAME$,X - 1)

```



```
80 FIRST$ = RIGHT$ (NAME$,L - X)
90 L = LEN (FIRST$): IF LEFT$ (FIRST$,1) =
   CHR$ (160) THEN FIRST$ = RIGHT$ (FIRST
   $,L - 1): GOTO 90
100 PRINT : PRINT "HELLO, ";FIRST$;
   LAST$;".": PRINT "MY NAME IS
   COMPUTER, APPLE!"
```

GOSUB POKER

In Apple's other language, the late Integer BASIC, you could set line numbers equal to variables (like LET X = SHUFFLE). Then you could say GOSUB SHUFFLE or IF EMPTY THEN SHUFFLE. When you looked at your program, you could tell what was happening without REMs. Well, you can't do that in Applesoft, not even with this program; it's only about 30 percent as good. Type it in carefully; the first two lines have to be *exactly* as printed here, or the whole thing gets scrambled.

```
10      GOTO 30
20      POKE 2213,48 + G - INT (G / 10) * 10:
        POKE 2212,48 + INT ((G - INT (G /
        100) * 100) / 10): POKE 2211,48 + INT
        ((G - INT (G / 1000) * 1000) / 100):
        POKE 2210,48 + INT ((G -INT (G /
        10000) * 10000) / 1000): POKE 2209,48
        +INT (G / 10000): GOSUB 00678: RETURN
30      SHUFFLE = 12345:COUNT = 678
40      G = SHUFFLE: GOSUB 20
45      G = COUNT: GOSUB 20
50      END
678     PRINT "THIS IS COUNT.": RETURN
12345  PRINT "THIS IS SHUFFLE.": RETURN
```

Notice that instead of "GOSUB SHUFFLE," you have to say G = SHUFFLE : GOSUB 20. When you do, the GOSUB in line 20 is actually *changed* in the listing.

For those of you who are planning on getting an Apple Macintosh, Apple's BASIC for the Mac can jump to labels. But with Mac's BASIC you don't even need line numbers! GOTO MAC.

GET GET STRAIGHT

GET is a funny animal. Here are today's quirks:

- a. If you GET a number with a statement like GET X, and you input a letter, the program bombs with a "?SYNTAX ERROR."
- b. GET interprets a right arrow (Control-U, ASCII 21) as a *space* (ASCII 32), *but* it interprets a left arrow (Control-H, ASCII 8) correctly.
- c. You can't GET an Escape (ASCII 27) keypress.
- d. GET interprets the length of a carriage return as 1, whereas INPUT considers the length zero.
- e. GET won't let Control-C exit a program.

Here is a getter that might teach you something:

```
10 PRINT "ANY KEY:"; : GET A$  
20 PRINT A$, ASC(A$) : GOTO 10
```

PLEASE ANSWER (Y/N)

If your program is looking for a yes or no answer, it is best to GET a one-character answer, something like:

```
10 PRINT "ANSWER (Y/N):" : GET A$
```

Oh, yes—something we learned the hard way: someone could answer with a *lowercase y or n*. We think Murphy's Law says you should be ready for this. GET, by the way, is better than:

```
10 INPUT "ANSWER (YES/NO):";A$
```

The problem with INPUT is that (we don't know *why*) people often put a *space* after their answer, and even the dumbest Apple knows that "YES" is not equal to "YES ".

GET THEIR ATTENTION

The INPUT and GET prompts in some programs are so boring that we end up slamming away on the RETURN key or some other key until we cream an entry and have to start all over. The following subroutine is guaranteed to get your attention:

```
10 TEXT : HOME :V = 22:H$ = " HIT ANY KEY
   TO CONTINUE -> ":L$ = CHR$ (93):R$ =
   CHR$ (91):F = 40:S = 70:FF = 30
20 FOR I = F TO 1 STEP - 2: GOSUB 100:
   NEXT I: GOSUB 300
30 FOR I = 1 TO LEN (H$): FOR N = 1 TO
   50: NEXT N: GOSUB 200: NORMAL : NEXT :
   WAIT - 16384,128: POKE - 16368,0: HOME
   : TEXT : PRINT "THANK YOU" : END
100 VTAB V: HTAB I: PRINT L$: FOR J = 1 TO
   S: NEXT J: VTAB V: HTAB I: PRINT " ":
   VTAB V: HTAB I - 1: PRINT R$: FOR K =
   1 TO S: NEXT : VTAB V: HTAB I - 1:
   PRINT " ": RETURN
200 HTAB I: VTAB V: PRINT L$: FOR R = 1 TO
   FF: NEXT R: HTAB I: VTAB V: PRINT R$:
   FOR L = 1 TO FF: NEXT L: HTAB I: VTAB
   V: INVERSE : PRINT MID$ (H$,I,1);:
   RETURN
```

```
300 BUMP = - 16336: FOR I = 1 TO F:BAP =  
    PEEK (BUMP): NEXT : RETURN
```

(Don't RUN it more than once. It's addictive.)

ELIMINATE THE NEGATIVE

You often see memory locations expressed as negative numbers. Why? Because a bunch of guys sat around in a room one day and decided to subtract 65536 from certain numbers that are larger than 32767. Good grief! So if you see a negative number (as in CALL -958), you can *add* 65536 to it and learn its true identity (as in CALL 64578).

?POP WITHOUT GOSUB ERROR

POP is an Applesoft command to be used when you do a GOSUB but then decide not to RETURN. Without POP, things may work all right for a while, but sooner or later you're going to get an "?OUT OF MEMORY" error, as in—

```
10 PRINT X; " SUIT";: GOSUB 50  
50 PRINT "CASE ": X = X + 1  
60 GOTO 10
```

When X reaches a value of 24, you'll get an error message. Line 60 *should* read:

```
60 POP : GOTO 10
```

If a program encounters a POP when *no* GOSUB has been executed, you will get a "?RETURN WITHOUT GOSUB" error message. Not quite correct, but you get the message.

ULTRA-POP

Someone once told us (we think it was our Pop) that it is *poor programming practice* to exit a subroutine with a POP statement. Instead, you should always exit properly via a RETURN. And *never* jump out of a FOR-NEXT loop without finishing the loop first.

Well, no one around here programs by the book (if they can help it). Heck, we get so deep into dodeca-nested loops and subsubroutines that we wouldn't know how many POPs to use if we had to! Here's a CALL (other than "He-lpp!") that will bail you out of all kinds of trouble:

CALL 54915

This big command "clears the stack" of all un-NEXTed FORs and all un-RETURNed GOSUBs. This program proves it:

```
100  REM CALL 54915
120  GOSUB 500
500  GOSUB 1000
1000 N = N + 1 : HTAB 1 : PRINT N;
1010 GET A$
1020 GOTO 100
```

RUN it and press any key a bunch of times; you won't be able to get the counter above 12. Remove the word REM from line 100 so it reads "100 CALL 54915" and RUN again! It may be poor programming practice, but it works!

PROGRAMMERS JUST WANNA HAVE FUN

Remember these for your repertoire of homilies:

1. Structured programming is a means to an end, not an end in itself.

2. If it RUNs, who cares?
3. If the manual says you can't do it, and you do it, who are you going to believe—your eyes or the manual?
4. People are smarter than computers, but computers are smarter than programmers.

DEL 10

The foregoing statement won't work in a program. DEL needs a range of lines. What *will* work is DEL 10,10. All programs screech (silently) to a halt with a DEL statement, so make it the last one in your program.

SUPER DEL

Sometimes you may want a program to delete its first few lines, especially if those lines BLOAD a file and you don't need it BLOADed the next time you RUN it. The DEL command will work in a program, but as we said the program stops right there. So forget DEL. Instead, why not have your program change the start of the program pointer? You can do this "on the fly," telling your Apple that the third line (or any line) of your program is now the *first* line (an Apple will believe anything). Type this program, SAVE it, RUN it, and finally LIST it. The first two lines, which could be BLOADers, will disappear! To get them back, do two POKEs—POKE 103,1 and POKE 104,8.

```
10 REM THIS IS LINE 10.
20 REM THIS IS LINE 20.
30 LOC = PEEK (121) + PEEK(122) * 256 + 1 :
   POKE 103,LOC - INT (LOC/256) * 256 :
   POKE 104, INT (LOC/ 256)
```

? = PRINT

Applesoft thinks ? means PRINT. Try ?2+2. If you use ? in a program and LIST it, the ?'s will be converted to PRINTs!

"QUOTE

Applesoft doesn't require an end quote mark in most cases. Try PRINT "HELP. (Think of the time you'll save!)

A PRINT-USING GOSUB

Applesoft loves to abbreviate numbers, quite a pain in the chips when you're working with dollars and cents: \$3.30 will print as \$3.3 and \$3.00 will print as \$3. And to make matters worse, \$3.2754 will print as \$3.2754. PRINT-USING is a useful command found on other computers. (We *think*; we just found out that there are other computers.) This command lets you specify ahead of time the decimal format for your screen and printer printouts. Well, the best we can do for you is this little GOSUB that rounds numbers to the nearest cent. Position the cursor where you want it (for example VTAB 20 : HTAB 19), set the variable AMT equal to the number you want printed, and GOSUB 60000. The number will be rounded to the nearest penny and printed for you in dollars-and-cents format.

```
5      TEXT : HOME : PRINT "PRINT-USING  
      DEMO": PRINT "-----":  
      PRINT " LIST 10-60": LIST 10 - 60  
10     SUM = 1.234: GOSUB 60000
```

```

20     SUM = 12.345: GOSUB 60000
30     SUM = 123.456: GOSUB 60000
40     SUM = 1234.567: GOSUB 60000
50     SUM = 12345.678: GOSUB 60000
60     SUM = 123456.789: GOSUB 60000
70     PRINT : PRINT "$ COLUMN = ";HT: PRINT
        "# WIDTH = ";WD;: END
60000 HT = 20: REM "$" COLUMN
60010 WD = 10: REM #WIDTH
60015 S$ = "          ": REM 12 SPACES
60020 SUM = SUM + .005: IF SUM < = .005
        THEN SUM = 0
60030 SUM$ = STR$ (SUM):L = LEN (SUM$): FOR
        CHR = L TO 1 STEP - 1: IF MID$ (SUM$,
        CHR,1) < > "." THEN NEXT :SUM$ = SUM$
        + ".":CHR = L + 1
60040 SUM$ = SUM$ + "00":SUM$ = LEFT$
        (SUM$,CHR + 2):SUM$ = RIGHT$
        S$ + SUM$,WD)
60050 HTAB HT: PRINT "$";SUM$: RETURN

```

WAS THAT -16633 OR -16363 OR -1633 . . . ?

All of those negative PEEK and POKE numbers will be easier to remember if you set a variable equal to one of them early in a program. Say, Q = -16384. Then, those hard to remember numbers become Q, Q+16, Q+48, etc. Much easier on your little gray ROM chips.

THE SECRET TO BETTER PROGRAMMING. . . .

Reprogram.

ON GOTO

If you list BYTEZAP.PRO, you'll notice a lot of ON . . . GOTO statements. This trick often allows more statements per line. Take this example . . . please:

```
10  X = 3 : REM MULTIPLE BRANCHES
20  ON X = 214 GOTO 100: ON X = 21982 GOTO
    200: ON X = 3 GOTO 300: ON X = 12.7
    GOTO 400 END
100 PRINT "CRABGRASS": END
200 PRINT "RANUNCULUS": END
300 PRINT "UNCLE SCRODGE": END
400 PRINT "BURMA SHAVE": END
```

If we had used IFs instead of ONs in line 20, it would have had to be broken into four program lines. Play around with this—it's almost as good as IF-THEN-ELSE . . . but not quite.

CHAPTER 5

MEMORY AND SPEED

IF CITY!

Applesoft sometimes limits you in your use of IF statements. Thus, if an IF statement is not true, Applesoft jumps to the next line number. *Not true* if you're dealing with certain types of information! Instead of

```
10 INPUT A
20 IF A = 0 THEN POKE 50,63 : REM INVERSE
30 IF A>0 THEN POKE 50,255 : REM NORMAL
40 PRINT " BEAGLE "
```

you could say

```
10 INPUT A: POKE 50,63 + 192 * (A>0) :
  PRINT " BEAGLE "
```

The IF statement here is really inside the parentheses which takes on a value of 1 or 0 depending on the truth of $A > 0$. IF $A > 0$ then the POKE becomes POKE 50, (63 + 192) or POKE 50,255 (normal). If not true, it's POKE 50,63 (inverse), *and the rest of the line is still read!* A very simple

example, but some things can be accomplished (like speeding up programs and *saving space*) using this trick.

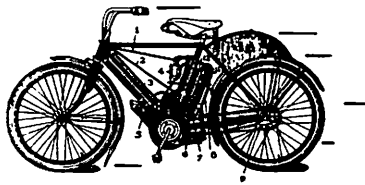
FLUSH RIGHT!

Using the preceding method, flush right numbers are easy. Watch:

```
10 FOR X = 5 TO 1055 STEP 50
20 PRINT SPC( (X<1000); SPC(X<100); SPC
   (X<10));X
30 NEXT X
```

MOST PROGRAMMERS WANT . . .

Most programmers want more speed in their programs and want their programs to occupy less memory. So, most programmers should abide by these tips:

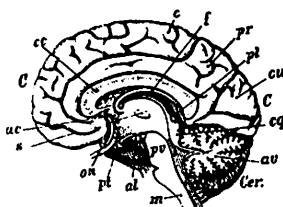


Speed Makers

1. Plan ahead to program efficiently. Write your program down in step-by-step plain English before you attack the keyboard.
2. Assign variables to frequently used constants. Don't use FOR

X = 12 TO 72. Instead LET T = 12 and LET S = 72, and use FOR X = T TO S. The variables assigned values most early in a program seem to execute fastest.

3. Use NEXT instead of NEXT X.
4. Place frequently called lines early in your program.
5. If your variables are going to be manipulated mathematically, use floating point variables instead of integer variables. Since all math is done in FP the variables have to be changed from integer to FP and that takes time.



Memory Savers

1. Renumber your program by 1's starting with line 1. Applesoft stores line numbers as strings, so the fewer digits in your GOTOs and GOSUBs, the less memory used.
2. Change all variable names to one character. Same reason as before.
3. DIM your arrays as small as possible, and don't waste the zeroth array.
4. Delete all REMs when you are through with them. SAVE a REMmed version if you want, and *use* the REMless one.
5. Delete useless words—change each THEN GOTO to a THEN or a GOTO. Don't use LET or END and don't use file names after CLOSE.
6. Use integer arrays if possible. Integer variables take less room than real variables.

Load a small test program. Then type:

```
PRINT FRE(0)
```

This will tell you how many bytes of memory are available. Now make one of the changes mentioned in the preceding list and PRINT FRE(0) again. The number will be slightly higher, indicating more free memory is available.

MEM SAVER

To draw a line of hyphens on the text screen, you can use:

```
10 PRINT "------  
-----"
```

This works fine, of course, but it uses 43 bytes of memory every time you do it. No problem if you've got memory to burn. The following line does the same job and uses just 15 bytes:

```
10 FOR X = 1 TO 40 : PRINT "-"; : NEXT
```

Remember, if you need to draw this line several times in the same program, set it up as a subroutine. Just tack a "RETURN" on the end. The cost? Only 2 more bytes; 1 for the colon and one for the RETURN.

REM OVE

If you're removing REMs from your program to save memory, you might want to instead *replace* each statement with a single colon. That way, if you have any GOTOs to your REM lines, things will still function normally.

SPACE SAVER #65535:

These two programs do the same thing. One uses less memory—

```
10 IF A<0 THEN B=60
20 IF A>0 THEN B=77
```

Or:

```
10 B=77: IF A<0 THEN B=60
20 (not necessary)
```

CLEARMEM

To really clear memory and reestablish normal pointers, you can always reboot. In DOS 3.3 you could simply type FP. In ProDOS you could type “-BASIC.SYSTEM,” but Control-Apple-Reset is easier to spell.

CHAPTER 6

LIST TIPS

DENUMBER

This program will ruin itself, so SAVE before you RUN.

```
10 HOME
20 PRINT "THE PROBLEM WITH THIS PROGRAM IS
   THAT"
30 PRINT "AFTER YOU RUN IT, IT WILL NEVER
   BE THE"
40 PRINT "SAME AGAIN."
50 POKE 2051,50: POKE 2057,40: POKE 2148,
   20: POKE 2167,10: LIST
```

REM EXPANDER

Watch this! A RUN changes the A's (234 of them, please) to all kinds of rude error messages. A *GOTO 100* repairs everything.

```

10  REM AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAA
20  N = 234
30  FOR I = 2054 TO 2287:N = N + 1: IF N >
    251 THEN N = 235
40  POKE I,N: NEXT : LIST : END
100 REM GOTO 100 TO FIX.
110 FOR I = 2054 TO 2287: POKE I, ASC
    ("A"): NEXT : LIST : END

```

REMLESS REMS

You can leave the word REM out of unencountered REM statements, but your Apple will parse the heck out of your remark. Type this in; it RUNS fine but LISTS funny.

```

0 GOTO 2
1 THIS IS LINE ONE, A SIMPLE STATEMENT,
  NOTHING MORE.
2 END

```

QUOTE REM

To keep your Apple from parsing your listing with REMless REMs, you can put a quote mark before your REMless REM. Pay attention:


```

999  END
1000 "*****
1010 "THIS PROGRAM WAS PRODUCED AND
1020 "DIRECTED BY
1030 "YOURS TRULY
1040 "*****

```

Go ahead and LIST it. There'll be no parsing, and since the program stops at line 999, it won't gag on the bad lines.

40-COLUMN FIXER

Extensive research by our Extensive Research Department shows that more people need to know this text tip than any other, so pass it around. To align text characters when they are printed on the screen, simply align them *in the listing when you type*. For example, this

```

10 PRINT "PROGRESS REPORT FOR H.H.LUMPY
--      MONEY IN      MONEY OUT      BILL
S DUE   $    4.00      $44444.00      $  4
44.00"

```

←note left margin alignment.

will appear like this when RUN:

```

PROGRESS REPORT FOR H.H.LUMPY--
MONEY IN      MONEY OUT      BILLS DUE
$    4.00      $44444.00      $  444.00

```

This alignment will take place only when you RUN a PRINT statement, *not* when you LIST it.

80-COLUMN FIXER

This works just like the 40-column fixer except you have 80 columns. Remember where you got this tip.

SHOWY REMS

INVERSE REM STATEMENTS!

We sure *wish* we knew how to make inverse REM statements, but we don't. However, here *is* a shortcut to making your REM statements show up better.

First POKE 33,32 (Integer) or POKE 33,28 (Applesoft). Then your REM statements will LIST formatted the way you type them (POKE 33,40 or Reset before LISTing). We like to underline our REMs with hyphens, like so:

```
LIST
970 INPUT X
980 IF X = 0 THEN PRINT "FIZZLE"
990 REM
      EXPLOSION:
      -----
1000 IF X = 86 THEN PRINT "KA BOOM!"
```

Experiment, and you'll see what we mean.

STEP LIST

During a long listing on an Apple II+, hold the control key and the S key with one hand and the Repeat key with the other hand. You can slow your listing down this way.

Note: The preceding tip has been classified as worthless by the Beagle Bros staff. Please tear it out of your book right away. Thank you.

REM MAKER

Here's the easiest way we know to make REMs show up well—type:

```
100 REM <Control-J> <Control-J> THIS IS A  
    REMARK <Control-J>
```

Now you have some air above and below your REM, making it easier to find in a listing.

LIST PREVENTION

POKE 2049,1 will make LIST list your first line repeatedly, because you have told the Apple that the *next line* starts at \$801 instead of \$8-something else. Try it in a boot program.

CONTROL-AT

Remember how a zero *ends* a program line? You can end one anywhere you want by POKEing a zero (Control-@) into the middle of it. Experiment with LISTing and RUNning after you have done your POKEing.

INVERSE REM STATEMENTS AT LAST!

After minutes of extensive research, our Uncle Louie finally came up with his finest achievement, *inverse REM statements* (flashing if you want!!)! Maybe not as practical as your normal kind of REM, but they sure do show up! First, you have to key in the following program:

```

10 DATA 201,141,240,21
20 DATA 234,234,234,234
30 DATA 201,128,144,13,201,160,176,9,72,
    132,53,56,233
40 DATA 128 : REM 64=FLASH, 0=NORMAL
50 DATA 76,249,253,76,240,253
60 FOR X = 768 TO 795: REM $300 TO $31B
70 READ N: POKE X,N: NEXT
80 POKE 54,0: POKE 55,3
90 CALL 1002: REM RESET OR PR#0 KILLS THE
    EFFECT

```

Now, RUN the program and key in the following:

```

5000 REM <Control-A> <space>
      <Control-T> <Control-E> <Control-S>
      <Control-T>

```

Your control characters show up as inverse, but you'll have trouble with Control-M, -U, or -X since they do special things (e.g., Control-M is a carriage return). Likewise, Control-A and Control-S won't work if you have *GPLE* in memory. Reset or PR#0 will get you out. This is a good way to hide your name or secret info in a program! Thanks, Uncle Louie.

SPLIT-SCREEN TEST

POKE 34,5 : LIST <RETURN> will freeze the top five lines on the screen as you LIST. The 5 can be any number, 1-23, of course. Very handy in many circumstances. Reset will normalize things.



TAKE OUT THE GARBAGE!

Got some Integer lines numbered greater than 32767 that you'd like to delete? Or eight pounds of unwanted HIMEMs at the end of a program? Try this to remove them. The line number should be just above the highest line you want to keep.

```
32767 POKE 76, PEEK (220) : POKE 77, PEEK  
      (221) : END : REM THIS IS INTEGER  
      BASIC, NOT APPLESOFT.
```

Now, GOTO 32767.

To remove garbage from the beginning of a program, the following line might work, depending on your garbage. Make the line number one number less than your first good program line number.

```
50 POKE 202, PEEK (220) : POKE 203, PEEK  
   (221) : END : THIS IS INTEGER BASIC, NOT  
   APPLESOFT.
```

Then GOTO 50. Good luck!

TOO-LONG TOKENS

Long Applesoft token words like INVERSE and NORMAL fail to produce a carriage return when they appear near the right margin of listings—a real pain when you're printing listings.

LINE TRACER

Anytime you want to know what line is operating in a program, put a PRINT PEEK(117) + PEEK (118)*256. To find out on what line an ONERR error was encountered, PRINT PEEK (218) + PEEK (219)*256. This program will demonstrate:

```
10  REM
    =====
    LINE TRACER
    =====
13  ONERR GOTO 5000
14  PRINT "THIS IS LINE "; PEEK (117) +
    256 * PEEK (118);".": FOR I = 1 TO
    500: NEXT
202 PRINT "THIS IS LINE "; PEEK (117) +
    256 * PEEK (118);".": FOR I = 1 TO
    500: NEXT
768 PRINT "THIS IS LINE "; PEEK (117) +
    256 * PEEK (118);".": FOR I = 1 TO
    500: NEXT : GOTO 14
5000 PRINT "PROGRAM STOPPED AT LINE ";
    PEEK (218) + 256 * PEEK (219);".": END
```

PROGRAM DIVIDERS

A “:REM<Control-J><Control-J>” at the end of a program line will put a visual break in a listing at that point. If the line ends in an END, a STOP, a RETURN, an ONERR statement, or a GOTO (the exception is ON . . . GOTO), you don’t even need the REM. In some cases you can even omit the colon.

INVISIBLE PRINTER

If you have no printer connected to your Apple II, II+, or IIe, and you type PR#1, your system will hang, forcing you to Reset. An Apple IIc, however, seems to think there is an invisible printer attached. Type PR#1 and then LIST. A bit later, after your listing is beamed into outer space, you’ll get your cursor back.

COLUMN-2 LISTINGS

The Apple IIc and the enhanced IIe start program listings at the second text column instead of the first. This is probably to facilitate easier Escape editing, since the second column is where you’ll usually find your Applesoft cursor (the prompt character occupies HTAB 1). You can Escape I or Escape up-arrow the cursor to the line number without having to move it one column to the left prior to tracing. Phooey on this kind of editing anyway. *GP*LE is the only way to fly!

CONTROL-L PAGE ADVANCE

The only thing that seems to be consistent about printers is their use of CHR\$(12) <Control-L> to activate a form feed (advance to the start of the next page). Use it to your advantage when making program listings:

1. Align your paper at the start of a new page.
2. Turn your printer power switch on (or off, then back on). This should establish the start-of-page position in your printer's little dot-matrix brain. (Some printers have a "Set Top of Page" switch for this.)
3. Type PR#1 <RETURN> to connect your printer.
4. Type LIST <RETURN>. Wait while your program is listed.
5. Type Control-L <RETURN>. Your paper should advance to the next top-of-page.
6. Type PR#0 to disconnect.
7. Tear off your listing and start over because you ripped the corner off the last page.

POST-LIST FIX

If you are stopping and starting a program listing with Control-S, and you reach the end of the listing, your last Control-S may accidentally become the first character of the next command you type, causing a "?Syntax Error." One solution is to hit a backspace or two before doing any typing.

CHAPTER 7

TEXT TIPS

VERTI-CALC

Actually, this program is *much* too valuable to be given away like this, but here goes, anyway:

```
5 TEXT : HOME : ONERR GOTO 15
10 FOR I = 1 TO 20: POKE 32,2 * I - 1: POKE
   33,1: HOME : READ A$: PRINT A$: NEXT I:
   TEXT : VTAB 23: END
15 POKE 216,0: RESTORE : ONERR GOTO 15
16 RESUME
20 DATA HAVE YOU EVER NEEDED,TO PRINT YOUR
   TEXT IN,VERTICAL COLUMNS,INSTEAD OF,
   HORIZONTAL ONES?,NO?,ME NEITHER!
```

READ THIS IMMEDIATELY

You may have noticed that you can't normally do an immediate-mode READ command. ("Immediate" means without a line number.) That's

only because DOS thinks you mean "READ" as in "Read a text file," instead of "READ" as in "Read data." To fool DOS, type :READ (notice the colon) instead of READ. DOS commands are only recognized at the far left of the screen. The colon could be anything legal, like a PRINT command—anything to separate READ from the left margin. For a quick test, type a program line:

```
10 DATA ONE, TWO, THREE, SEVEN
```

Now type:

```
:READ A$ : PRINT A$
```

Do it a few times. It works!

PAGE-2 HELP PAGE

This program installs a "help page" on text page 2 and lets you view it whenever you want, by typing Control-I (for "Info") or pressing the TAB key on your Apple //e or //c. It's up to you to expand this into a full-blown word processor. Call us when you're finished.

```
1  IF PEEK (103) + PEEK (104) * 256 < >
    3073 THEN POKE 103,1: POKE 104,12:
    POKE 3072,0: PRINT CHR$ (4)"RUN HELP
    SCREEN"
10  X$ = "TAB": IF PEEK ( - 637) < 255
    THEN X$ = "CTRL-I"
15  TEXT : HOME
20  PRINT "HELP PAGE": PRINT : PRINT X$;"
    = HELP": PRINT "CONTROL-Q = QUIT": FOR
    I = 3 TO 9: PRINT "COMMAND #";I;" =":
```



```

NEXT : TEXT : PRINT "<ANY KEY> TO
CONTINUE";: REM PRINT HELP PAGE
50 POKE 768,216: POKE 769,160: POKE
770,0: POKE 771,76: POKE 772,44: POKE
773,254: POKE 60,0: POKE 61,4: POKE
62,255: POKE 63,7: POKE 66,0: POKE
67,8: CALL 768: REM MOVE PG.1 TO PG.2
80 HOME : PRINT "TYPE ON THIS PAGE ("X$"
FOR INFO)": PRINT
100 GET A$: IF A$ = CHR$ (9) THEN GOSUB
2000: GOTO 100
105 IF A$ = CHR$ (17) THEN END : REM QUIT
IF CONTROL-Q
110 PRINT A$;: GOTO 100
2000 POKE - 16299,0: GET A$: POKE -
16300,0: RETURN

```

Warning: Save this program as "HELP SCREEN" before you RUN it—use your program name in line 1. These commands move the program above text page 2.

INVERSE TYPER!

Here's an easy way to type directly to the screen in inverse (or flash):

```

10 INVERSE : REM OR FLASH
20 INPUT A$ : PRINT A$;: GOTO 20

```

But you have to RUN the program and remain in the program for it to work. RUN the following program and you will get inverse alphabetical characters after you are out of the program!

```

10 DATA 201,141,240,21
20 DATA 234,234,234,234
30 DATA 201,192,144,13,201,224,176,9,72,
132,53,56,233
40 DATA 192 : REM 128 FOR FLASH
50 DATA 76,249,253,76,240,253
60 FOR X = 768 TO 795: REM $300 TO $31B

```

```
70 READ N: POKE X,N: NEXT
80 POKE 54,0: POKE 55,3
90 CALL 1002: REM RESET OR PR#0 CANCELS
  THE EFFECT
```

MORE QUOTE MARKS

Applesoft—

```
10 HOME
20 PRINT "SHE SAID, "; CHR$(34); "HELLO!";
  CHR$(34)
```

Integer—

```
10 CALL -936
20 PRINT "SHE SAID, "HELLO!" " : END
```

No tricks in the Applesoft example. The illegal extra quote marks in the Integer program are actually lowercase *B*'s! Without a lowercase adapter, they will appear as quote marks on the screen. (This is for you old-timers. Apple //e and //c owners don't know what they're missing!!!)

BETTER INVERSE

Inverse titles look better if you *frame* them with spaces. Try it.

```
10 HOME: INVERSE: HTAB 17: PRINT "HEADLINE"
20 VTAB 12: HTAB 16: PRINT " HEADLINE " :
  NORMAL
```

If you've got room, you can *really* frame a title—

```
10 HOME : INVERSE : HTAB 18: PRINT SPC(7)
20 VTAB 2: HTAB 18: PRINT " TITLE "
30 HTAB 18: PRINT SPC(7) : NORMAL
```

CATALOG DISK
SAVE FILE
LOAD FILE
ERASE DISK
INITIALIZE DISK
DESTROY DISK
QUIT

INVERSE CURSOR BAR

A lot of software uses an “inverse bar” to select menu items. Pressing an arrow key usually moves the bar, and Return makes the selection. Here’s how it’s done from Applesoft:

```
10 TEXT: HOME: NORMAL: V=1
20 FOR N=1 TO 5: READ A$(N): PRINT A$(N):
   NEXT DATA ONE, TWO, THREE, FOUR, FIVE
30 INVERSE: VTAB V: PRINT A$(V)
40 K=PEEK(-16384): IF K < 128 THEN 40
50 POKE-16368,0: NORMAL: VTAB V: PRINT
   A$(V): IF K=141 THEN VTAB 10: PRINT "YOU
   SELECTED #";V;".".": END
60 IF K=136 OR K=139 THEN V=V-1: IF V=0
   THEN V=5
70 IF K=149 OR K=138 THEN V=V+1: IF V=6
   THEN V=1
80 GOTO 30
```

The K values (lines 60 and 70) of 136 and 139 are the Left and Up arrows. 149 and 138 are Right and Down.

YOU CAN Q\$OTE THIS

Here's another way to get quote marks in a PRINT statement.

```
10 Q$=CHR$(34)
20 PRINT "THIS IS ";Q$;"ILLEGAL.";Q$
```

Did you know that semicolons are often unnecessary in spite of what your mother told you? Line 20 could be:

```
20 PRINT "THIS IS "Q$"ILLEGAL."Q$
```

IF INVERSE THEN GOSUB 99

The preceding statement won't work. What will work is:

```
IF PEEK(50)=63 THEN GOSUB 99
```

To find out the format of the characters about to be output to the screen, PEEK at location 50 by typing PRINT PEEK(50). A 63 answer means inverse; 255 means normal; 127 is flash. You can also create the format you want by POKEing in these numbers (although simple commands do the same thing)—POKE 50,63 will create inverse type and POKE 50,255 will create normal. POKE 50,127 will flash alphabetical characters. You'll have to add a POKE 243,64 to flash numbers and other characters.

If you want to get lowercase inverse on your Apple //e or //c, you have to first

```
POKE 49167,0
```

and then POKE in your screen addresses (beginning at 1024) the values between 97 and 122. To return to the normal character set:

```
POKE 49166,0
```

And you didn't think you could get inverse lowercase!

TEXT-SCREEN FORMAT

In case you aren't clear about the layout of your text screen, RUN this program:

```
10 REM
   =====
   TEXT SCREEN FORMAT
   =====
20 HOME : NORMAL : FOR I = 1 TO 39:A$ = A$
  + CHR$ (95): NEXT : FOR I = 1 TO 24:
  VTAB 1 : HTAB(1): PRINT A$;; NEXT :
  INVERSE : FOR X = 2 TO 23: FOR Y = 5 TO
  40 STEP 5: VTAB X: HTAB Y: PRINT LEFT$
  (A$,1);: NEXT : NEXT
60 FOR Y = 1 TO 24 STEP 23: FOR I = 5 TO 35
  + 5 * (Y = 1) STEP 5: VTAB Y: HTAB I -
  (I > 9): PRINT I;; NEXT : NEXT : NORMAL
  : FOR I = 1 TO 24: VTAB I: HTAB 1: PRINT
  I;; NEXT : POKE 2038,52: POKE 2039,48:
  INVERSE : WAIT - 16384,128: NORMAL
```

COLUMN POKER

To make a flush-left column in the middle of the screen, you can do this:

```
10 HOME : VTAB 10: HTAB 15: PRINT "A
  STITCH": HTAB 15: PRINT "IN TIME": HTAB
  15: PRINT "IS WORTH": HTAB 15: PRINT
  "TWO IN": HTAB 15: PRINT "THE BUSH"
```

But why not POKE locations 32 and 33 and do this?

```
10 VTAB 10: HTAB 15:M$ = CHR$ (13): POKE
   32,14: POKE 33,26: PRINT "A STITCH"M$"IN
   TIME"M$"IS WORTH"M$"TWO IN"M$"THE
   BUSH.": TEXT
```

The second method takes less typing time and less memory. Just POKE 32 with a number *one less* than the column number, and POKE 33 with 40 *minus* PEEK(32). You can sometimes omit the POKE at 33 if you are *not* going to print off the screen (causing wrap-around). If you *do*, and PEEK(32) plus PEEK(33) totals more than 40 (with a 40-column screen), you can:

- a. Zap your program.
- b. Make your drive go "BLAA-AATT!"
- c. Cause wrap-around to the wrong line.
- d. Prove us wrong.

LOWERCASE EQUIVALENTS

In the old days before Apples were born with lowercase, you had to be careful in writing programs with lowercase on the text screen. This is because it would show up as garbage on Apples without lowercase adapters. To see the equivalent characters that others might see when a program says PRINT "Hello," type:

```
10 INVERSE : PRINT "Hello"
```

In this case, you will be presented with H%,,/ (in inverse). That's what a nonlowercase Apple will produce even in normal mode.

If you are about to add lowercase hardware to your old Apple, shop around. Have your dealer *show you* a comparison of the typestyles available. (Let's face it, though, with the built-in lowercase on new Apples, dealers are not going to have an abundant choice of lowercase adapters.) A couple of the popular ones look like they were designed by engineers, not typographers. Poorly designed (ugly) text-screen type can really slow you down, especially if you are doing word processing.

TEXT OR HI-RES

If your Apple monitor is a color TV, you can tell if the screen text you are reading is hi-res or text by looking for little color glitches between vertical strokes on letters such as *m*'s. If you see them, it's hi-res. By the way, you won't see small (normal size) color text on the Apple. There just aren't enough dots available to do the job.

SCREEN FORMULA

We've been trying to come up with this one for years (or at least since last week)—it's the formula for converting HTAB and VTAB coordinates into the corresponding memory addresses. Let V equal the VTAB (1–24) and H equal the HTAB (1–40). To find the memory location, LOC, for the character in that screen position, use (are you ready?):

$$LOC = 128 * V + H - (984 * \text{INT}((V - 1) / 8)) + 895$$

Hey, it works! Now, to find what character is residing at VTAB V, HTAB H, you can PEEK at LOC and get its value, like so:

```
PRINT PEEK(LOC)
```

Or, you can POKE any character you want onto the screen with:

```
POKE LOC, X
```

(Rummage through the appendixes to find the chart for character values.)

BSAVE TEXT

Sure, you can BSAVE the text screen! Just type BSAVE TEXTTEST, A\$400,L\$3FF. Anytime you want it back, type "BLOAD TEXTTEST". Of course, any command you type to save the screen will appear when you BLOAD it. To prevent this, put the BSAVE statement as the last line in your text-printing program:

```
PRINT CHR$(4);"BSAVE TEXTTEST,A$400,L$3FF"
```

ERRORLESS ERRORS?

Applesoft just loves to crash when you specify a number it can't handle. Try HPLOT 0,300 for example. You'll get an "?ILLEGAL QUANTITY" message because 300 is too large a number for a hi-res plot. Wouldn't it be nice sometimes if Applesoft just ignored an error instead of scolding you? Well, it seems to do just that with its MID\$, LEFT\$, and RIGHT\$ commands. Try this:

```
10 A$="FISH" : FOR X = 1 TO 9  
20 PRINT X;"."; LEFT$ (A$,X): NEXT
```

Theoretically, the program should bomb when X becomes greater than the length of A\$, which is 4. But it doesn't. For example, when X is 7, line 20 asks the computer to print the *left 7 letters* of a 4-letter word. MID\$ and RIGHT\$ seem to work the same way, and I don't mind at all; it's one detail I *don't* have to look for when debugging.

BETCHA CAN'T JUST RUN IT ONCE

We found the following program to be one of the all-time dumb programs. However, no matter how hard we try, we can't just RUN it once.

```
5  TEXT : HOME
10 POKE 2000,157: POKE 2001,96
```

It's a great HELLO program for disks.

ANOTHER ONE-BYTE SAVER

Instead of VTAB 5: HTAB 1: PRINT IT, use VTAB 4: PRINT: PRINT IT. You'll save one big byte (or two if you're further down the screen). This is a good 80-column trick that overcomes an HTAB bug in the older //e's.

CHAPTER 8

A FEW BUGS

This is the Apple bug department—we don't explain 'em; we just find 'em!

SQUARE BUG

PRINT 7 * 7 and PRINT 7 ^ 2 will produce different answers!! Watch this. . . .

```
10 TEXT : HOME
20 PRINT "NUMBER SQUARED CUBED"
30 PRINT "-----"
40 POKE 34,2
50 FOR X = 0 TO 255: PRINT X;
60 HTAB 9: PRINT X ^ 2;
70 HTAB 21: PRINT X ^ 3: NEXT X
```

GET BUG

Turn off DOS by booting with no disk and hitting Reset; then type:

```
10 GET G$: V=VAL(G$) : PRINT V
```

RUN and enter a digit, 1-9, for G\$. Look at the answer! Inserting a G\$=G\$ after GET G\$ seems to clear things up. The explanation for this is really boring. Reboot to continue.

WHY DOES THIS DO THIS?

```
10 HOME
20 SPEED= 200
30 FOR N = 0 TO 1 STEP .001
40 PRINT "N = ";N
50 NEXT N
60 SPEED= 255
```

NUMBER BUGS

We really hate bugs like this; programming is hard enough without Applesoft's casual attitude with numbers. What answer does your Apple give for this?

```
PRINT INT(14.4 * 100)
```

It's 1439, right? Heck, our \$8 pocket *calculator* does better than that. (It plays a lousy game of *Raster Blaster*, though.) A better way to perform the calculation seems to be:

```
N=14.4 * 100 : PRINT INT(N)
```

WHY DOES THIS DO THIS?

<pre>>LIST 10 REM WHY THIS? 15 REM (INTEGER) 20 CALL -936: GR 30 COLOR= RND (16)+1 40 N= RND(1280) 50 H=N MOD 32 60 V=N/32 70 PLOT H,V 80 GOTO 30</pre>	<pre>JLIST 10 REM AND THIS? 20 REM (APPLESOFT) 30 FOR X = 1 TO 255 40 PRINT CHR\$(X), 50 REM BOMBS WITH COMMA, BUT NOT WITHOUT! 60 NEXT X</pre>
--	---

SAY IT LIKE IT AIN'T

“Path Not Found” is a dumb-looking error message, undoubtedly invented to confuse beginners. So is “No Buffers Available,” but we won’t mention that here.

BAH, NUMBUG!

Most sorters we know of will rank the numbers 3, 20, and 100 in reverse order—100-20-3—because ASCII-wise that is correct. (Likewise, the words “C,” “BO,” and “ALL” alphabetize ALL-BO-C.) To support proper numeric sorting, you should start short numbers with spaces or zeros so that all numbers in the list have the same length. Our example numbers would then sort properly—003-020-100.

CHAPTER 9

PROTECTION

COPY STOPPERS (DOS 3.3 ONLY)

People want to know how to protect their disks from copying, so they can quit their jobs and sell their computer programs. Well, quit your job anyway, but sooner or later *everything* will be copyable. But don't fret; the recording industry survives, doesn't it? As our friend R. W. T. Sector once said, "You can't stop all of the people from copying a disk all of the time, but you can *slow* some of the people *down* some of the time, depending on *which copy program* they are using." With these wise words in mind, try this trick:

- a. Type: POKE 44033,16
- b. Put a new disk in your drive and INIT it

What you have done is change the track of your catalog from 17 (normal) to 16. Other numbers, 3-15, will work too when POKEd into 44033.

To transfer programs from a normal disk to your new disk:

1. Boot a normal disk.
2. LOAD a program from the disk.

3. Type POKE 44033,16 (assuming you used 16).
4. Insert your protected disk, the one you INITed earlier.
5. SAVE the program.
6. Type POKE 44033,17 (that's normal).
7. Insert your normal disk and continue with step 2.

Remember, this little trick won't work with *all* copy programs, and it won't stop anyone who knows much about copy protection. Copy-protect schemes that work today are *extremely* complicated (and they won't work next August).

Another way to slow down copiers is with our program *DOS BOSS*. It will change DOS 3.3 commands and error messages for you, and slow potential copiers down quite a bit.

The great race is indeed on between the Copiers and the Copy Protectors. Progress is being made every day in both camps, and, if you ree-ealy think about it, there can only be one winner—the Copiers.

SAVE PROTECTING YOUR PROGRAMS

There are four basic methods a person will use for copying your software:

1. LOAD and SAVE after booting your disk.
2. LOAD and SAVE after booting another disk.
3. Use the FID program from the System Master disk.
4. Use a copy program to copy your entire disk, DOS and all.

Using *DOS BOSS* and the following trick, you can foil methods 1, 2, and 3. And method 4 has its drawbacks. The trick involves forcing the user to *boot with your DOS*. The effect is this: suppose Joe Blow wants to make an unauthorized copy of your fantastic new game. After trying method 1, he encounters a (beep!) "NOT COPYABLE" message. He then tries methods 2 and 3, both of which seem to work, but when he tries to RUN any program on the copied disk, it won't work and crashes into the monitor. Joe is getting discouraged. Perhaps he gives up, or perhaps he

goes on to method 4 and copies your entire disk. This works fine, but every time he catalogs your disk, he gets your bold copyright message in the heading, reminding him of his dishonesty. Also, he has had to use up a whole disk with your darn personalized DOS on it. Joe is sorry he ever messed with you!

The procedure to make all this happen goes like so:

- a. RUN *DOS Boss*.
- b. Change the READ command to SAVE and the SAVE command to KEEP.
- c. Replace the "NOT DIRECT COMMAND" error message with "NOT COPYABLE."
- d. Personalize your disk volume heading. Make it fifteen characters or less.
- e. Quit *DOS Boss* and type NEW <RETURN>.
- f. Type POKE 45995,96 <RETURN>. If you have 32K, make it 29611,96. (Does *anybody* still have 32K?)
- g. Insert a new disk and type INIT HELLO <RETURN>. Your new personal DOS will be on this new disk.
- h. Copy your programs onto the new disk, using method 2 or 3.
- i. Somewhere in each program, several times if you want, insert a CALL 45995, with a line number, like:

```
1000 CALL 45995
```

Or tuck this statement amid your existing program statements:

```
1234 INPUT "D.K.?";A$: CALL 45995: IF  
    A$="Y" THEN PRINT "CONTINUE  
    PROGRAM. . ."
```

The CALL in step (i) is like a GOSUB to a machine-language routine at memory location 45995. The 96 you POKEd in with step (f) is actually a machine-language RETURN. If the 96 is encountered at 45995, nothing happens and your program continues, which is what you want. If any other number is encountered (meaning your disk was not booted), the program crashes! An equally good command, instead of CALL 45995, would be IF

PEEK(45995) <> 96 THEN NEW. This will *erase* the program if your disk wasn't booted!

Have fun! And let's hope Joe Blow doesn't have DOS BOSS!

TRICK FILE NAMES

One of the easiest ways to protect your program from a LOAD and SAVE is to put a control character in your file name. Have your HELLO program LOAD your important file. For example, with your program in memory, type:

```
SAVE GAME<Control-E>
```

If someone tries to LOAD GAME, they'll get a "FILE NOT FOUND" error.

MYSTERY POKE

Since your Apple has so many hidey holes, another protection scheme is to POKE a secret number into a location with a *loader* program and then PEEK the location to see if the number is correct. For example:

```
5 REM***LOADER PROGRAM***
10 POKE 768,65
20 PRINT CHR$(4) "RUN MAIN PROGRAM"

5 REM***MAIN PROGRAM
10 IF PEEK(768) <> ASC("A") THEN 60000
20 REM PROGRAM STARTS HERE
59999 END
```

```
60000 REM NERD TRAP #1
60010 TEXT : HOME
60020 PRINT "YOU NERD!!"
60030 NEW
```

Use a title for your loader program that will identify the program as yours. For instance, INIT your disk with the loader program using a name like:

COPYRIGHT (C) 1988 BY IMA PROGRAMMER

Most copiers don't want that on their conscience, so they just go after the main program. When they RUN it, they get called a name, which serves them right.

FOR WARPED MINDS

Now if you really want to get a copier, here's a rotten trick to be used with the "Mystery POKE" method.

```
59998 REM DOS 3.3 ONLY (NOT PRODOS)
59999 END
60000 REM NERD TRAP #2
60010 PRINT "PLEASE REMOVE WRITE-PROTECT
        TABS"
60020 PRINT "=HIT ANY KEY TO CONTINUE:";
60030 GET A$ : PRINT A$
60040 PRINT CHR$(4)"INIT GOODBYE!",D1
60050 PRINT CHR$(4)"INIT GOODBYE!",D2
60050 NEW
```

That will initialize disks in both drives, possibly zapping their copy program. They may be able to copy your program, but in the process, they'll pay dearly.

UNLISTABLE

This next trick is a real gem to use in your loader program so that the mystery POKE cannot be seen. In fact, it's a gem for *any* LIST prevention.

First, crank up *GPLE* and enter:

```
10 REM
```

Now, edit 10 with Control-E and once you're in the edit mode, press Control-O, Control-H *eight* times. (Eight inverse H's will appear on your screen.) Next, enter eight spaces and then Control-O, Control-M. Enter a message and another Control-M at the end of the message. Finally, enter Control-O, Control-D FP. Line 10 now looks like this:

```
10 REM hhhhhhhh      m UNLISTABLE! mdFP
20 REM (h=CONTROL-H, m=CONTROL-M,
      d=CONTROL-D) PROGRAM STARTS HERE
```

Place the rest of your program after line 10. There's no way your program can be LISTed from the beginning after line 10. You can RUN the program and LIST it beyond line 10, but it's a killer if it ever smacks into that line.

What happens is that the line number and REM are backed over by the Control-H's and then "overlaid" with the eight spaces. The Control-M is a carriage return for your message below the invisible line number and REM. The Control-D acts as a DOS command to FP, which will erase your program in memory. If you're using ProDOS, use something other than FP. PR#6 is good.

BANK ROBBER

Here's a wholly irresponsible program. It shows how code breaking might look in a computer bank robbery. (*Do not use this on your modem.* You may generate the untender attention of the authorities!!)

```
10 TEXT : HOME : RESTORE
20 VV$ = " BANK TRANSFER PROGRAM ": HTAB
  20 - LEN (VV$) / 2: INVERSE : PRINT
  VV$: NORMAL
30 FOR I = 1 TO 4: READ BANK$(I): NEXT
40 VTAB 5
50 INPUT "YOUR NAME-> ";NA$: PRINT :
  PRINT
60 INPUT "YOUR BANK-> ";YB$
70 HOME : VTAB 5
80 FOR I = 1 TO 4: PRINT I;". ";BANK$(I):
  PRINT : NEXT
90 PRINT : INVERSE : PRINT "WHICH BANK TO
  ACCESS->";: GET AN: NORMAL : PRINT AN
100 ON AN GOSUB 2000,3000,4000,5000
1000 DATA BANK OF KUWAIT,BANK OF IRAN,BANK
  OF AMERICA,BEAGLE BROS ACCOUNT
2000 HOME : VTAB 7: PRINT "THE BANK OF
  KUWAIT HAS FUNDS IN EXCESS": PRINT "OF
  $100 BILLION. YOU CAN TRANSFER UP":
  PRINT "TO $2.3 MILLION WITHOUT
  NOTICE."
2010 GOTO 9000
3000 HOME : VTAB 7: PRINT "THE ";BANK$(AN);
  " HAS FUNDS IN EXCESS": PRINT "OF $75
  BILLION. YOU CAN TRANSFER UP": PRINT
  "TO $3.3 MILLION WITHOUT NOTICE":
  PRINT "DUE TO THE CONFUSION IN
  LEADERSHIP."
3010 GOTO 9000
4000 HOME : VTAB 7: PRINT "THE ";BANK$(AN);
  " HAS FUNDS IN EXCESS": PRINT "OF $175
  BILLION. YOU CAN TRANSFER UP": PRINT
  "TO $4.2 MILLION WITHOUT NOTICE":
  PRINT "SINCE THEY USE IBM PC'S FOR
  SECURITY."
4010 GOTO 9000
5000 POKE 33,50: LIST
5010 POKE - 16299,0
6000 HOME :MX$ = " BANK TRANSFER IN
  PROGRESS ": HTAB 20 - LEN (MX$) / 2:
  FLASH : PRINT MX$: NORMAL : REM ***
  TRANSFER ROUTINE ***
6010 VTAB 16: PRINT NA$;" 'S": PRINT
```

```

"ACCOUNT": PRINT "AT ";YB$: PRINT :
INVERSE : PRINT "<<<<<<<": NORMAL
6020 BK$ = BANK$(AN): VTAB 16: HTAB (40) -
LEN (BK$): PRINT BK$: PRINT : PRINT :
PRINT :AR$ = ">>>>>>>>>": HTAB (40)
- LEN (AR$): INVERSE : PRINT AR$:
NORMAL
6030 FOR J = 1 TO 30
6040 FOR I = (1488 + 7 + 20) TO (1488 + 7)
STEP - 1
6050 N = INT ( RND (1) * 10) + 176
6060 GOSUB 8000
6070 POKE I,N: NEXT : NEXT
6080 FOR I = (1488 + 7 + 20) TO (1488 + 7)
STEP - 1: POKE I,32: NEXT
6090 MY$ = " BANK TRANSFER COMPLETE ":
VTAB 1: HTAB 20 - LEN (MY$) / 2:
INVERSE : PRINT CHR$ (7);MY$: NORMAL :
VTAB 23
6100 VTAB 23: HTAB 8: PRINT "HIT ANY KEY TO
CONTINUE": WAIT - 16384,128: POKE -
16368,0: RETURN
7000 REM *** PICK LOCK ROUTINE ***
7010 HOME :DC$ = " DECODING KEY ": HTAB 20
- LEN (DC$) / 2: FLASH : PRINT DC$:
NORMAL
7020 BK = INT (RND (1) * 99999) + 10000
7030 BK$ = STR$ (BK)
7040 FOR I = 1 TO 5
7050 PK = INT ( RND (1) * 10)
7060 FOR PP = 1 TO PK: POKE 1192 + PP +
17,163: GOSUB 8000: NEXT PP: FOR PN =
1 TO PK: POKE 1192 + PN + 17,160: NEXT
PN
7070 PK$ = STR$ (PK)
7080 GOSUB 8000
7090 IF PK$ = MID$ (BK$,I,1) THEN 7110
7100 VTAB 22: HTAB I + 17: PRINT PK$: GOTO
7050
7110 PRINT CHR$ (7): VTAB 22: HTAB I + 17:
INVERSE : PRINT PK$: NORMAL : NEXT I
7120 VTAB 22:TB$ = "TODAY'S CODE IS->":
PRINT TB$
7130 UL$ = " DECIPHERING COMPLETED ": VTAB

```

```
      2: HTAB 20 - LEN (UL$) / 2: INVERSE :  
      PRINT UL$: NORMAL  
7140  VTAB 24: HTAB 8: PRINT "HIT ANY KEY TO  
      CONTINUE": WAIT - 16384,128: POKE -  
      16368,0: RETURN  
8000  REM *** CLICKING ***  
8010  CLICK = - 16336  
8020  UL = PEEK (CLICK): RETURN  
9000  PRINT : PRINT : INVERSE : PRINT "HOW  
      MUCH TO TRANSFER TO YOUR ACCOUNT?":  
      NORMAL : INPUT "$";MU  
9010  PRINT : PRINT : INVERSE : PRINT "DO  
      YOU HAVE THE DAILY CODE NUMBER FOR ":  
      PRINT BANK$(AN);" (Y/N)";: GET CD$:  
      PRINT CD$: NORMAL  
9020  IF CD$ = "N" THEN GOSUB 7000  
9030  INPUT "PLEASE ENTER THE CODE->:";DC  
9040  IF DC = BK THEN GOSUB 6000  
9050  HOME : PRINT : VTAB 10: INPUT "  
      ANOTHER TRANSFER?(Y/N) : ";AN$: IF AN$  
      = "N" THEN END  
9060  IF DC = BK THEN GOTO 10  
9070  PRINT : FLASH : PRINT "THAT IS NOT THE  
      CODE";: PRINT CHR$ (7): NORMAL : PRINT  
      : PRINT "HIT ANY KEY TO CONTINUE": GET  
      GT$: PRINT GT$  
9080  GOSUB 7000  
9090  GOTO 9030
```

CHAPTER 10

SOUND

APPLE METRONOME

Psst. . . . Here's a routine that makes your \$2,000 Apple act like a \$29.50 metronome (wait, this metronome flashes; that's worth an extra ten bucks for sure!). The variable TIME controls the metronome's speed. Change it until you like what you see and hear.

```
10 TIME = 300 : REM LOWER=FASTER
20 FOR J = 1 TO TIME: NEXT
30 VTAB 10 : PRINT SPC(80)
40 FOR X = 1 TO 5 : A = PEEK (49200): NEXT
   X
50 HOME : INVERSE : GOTO 20
```

POKE WON'T CLICK!?

Hey! Someone just pointed out that POKE -16336,0 *won't* click his Apple's speaker, and page 123 of the Integer manual (available in antique bookstores) says it *will*. Same is true on our Apple. How about yours?

Meanwhile, X=PEEK(-16336) clicks just fine, but only every other time! Mysteries of the deep!

BEEPNOTES

First of all, to “print” a beep, you type PRINT then a quote mark, then a Control-G (audible, but invisible), and then another quote mark. Follow that with a semicolon if you don’t want the screen to scroll. You will hear the Control-G in the listing as well as in the program, unless . . .

To prevent your program’s Control-G beeps in your listings, type G\$ = CHR\$(7) (same as Control-G). Now when you want a beep, simply PRINT G\$.

And, speaking of beeps, you can use Control-G as a nifty debugging device. Often the reason a certain program line doesn’t work is that the program *never got to* that line. To see if it did, simply put a PRINT CHR\$(7) at the beginning of the statement. RUN the program; if it beeps, you know it encountered the statement in question.

UNWANTED SOUNDS!

(Please don’t RUN this program.)

```
10 GOTO 50
20 "Open the windows! Sound the alarm!
30 " : ALARM 0 30 DO 7 EMIT LOOP ;
40 " Do not CALL -151 and C030L
50 TEXT : HOME
60 FOR I = 768 TO 795: READ J: POKE I,J:
   NEXT
70 POKE 1013,76: POKE 1014,0: POKE 1015,3
80 I = 22:K = 22
85 REM
```

To hear what martians sound like, change the value of I in line 80 to 122 and the variable I to J in line 90.

```

90  FOR I = 1 TO 20: NEXT
100 N = N + 1: IF N = 7 THEN GOTO 200
120 & I,K: GOTO 80
130 DATA 32,70,231,134,81,164,80,152,170,
      202,208,253,44,48,192,69,81,1,70,202,
      208,253,44,48,192,136,208,236,96
200 P00$ = "WELL EXCUUUSSSE ME!"
205 FOR W = 1 TO 1500: NEXT
210 VTAB 10: HTAB 20 - LEN (P00$) / 2:
      SPEED= 50: PRINT P00$: SPEED= 255: VTAB
23

```

CHIRP

User, the Beagle Bros' cat, goes crazy when we play this one for him:

```

10 FOR X = 0 TO 17: READ A: POKE 12345 +
      X,A: NEXT : DATA 173, 48, 192, 136, 208,
      4, 198, 0, 240, 7, 202, 208, 246, 166,
      0, 208, 239, 96
20 N = RND (1) * 6: R = 1 + RND (1) * 66:
      FOR X = 1 TO N: POKE 0,R: CALL 12345:
      NEXT : GOTO 20

```

Try changing N and R to constants. It's all kinds of fun!

MOO CALL



Did you know that under DOS 3.3, typing "CALL 985" will make your Apple moo like a cow? Sometimes once, sometimes twice, sometimes not at all.

CHAPTER 11

GRAPHICS

NIGHT DRIVER

This is a neat program that lets you follow a lo-res car down a long road to nowhere. (If you don't think there's much you can do with lo-res, take a look at this!)

```
10 GR : HOME : NORMAL : A = 1 : B = 21 : D =  
22 : L = 20 : R = 39 : S = 40 : U = 3 : V = 15 : M =  
10000 : Z = 0 : E = 36 : F = 37  
20 COLOR = 2 : FOR Y = 28 TO 34 : HLIN 23,27  
AT Y : NEXT : COLOR = V : HLIN D,28 AT 35 :  
COLOR = 5 : PLOT 25,33 : REM DRAW CAR  
30 FOR C = Z TO V STEP U : IF C = Z OR C = 6  
THEN COLOR = V * (C = Z) : FOR I = 23 TO  
26 : HLIN L,I AT I + A : NEXT I : HLIN 24,  
26 AT 30 : HLIN 24,26 AT 29 : HLIN B,D AT  
28 : PLOT D,29 : REM HEADLIGHTS.ON/OFF  
40 FOR X = L TO R STEP A + (C > U) : COLOR =  
C : HLIN S - X,X AT R - X : VLIN S - X,X  
AT X : VLIN R - X,X AT S - X : REM BKGRD  
50 COLOR = U : M = M + A : HTAB 24 : PRINT M ; :  
PLOT 27,41 : REM MILEAGE
```

```

60 COLOR= X: VLIN E,F AT 23: VLIN E,F AT
  27: REM TIRES
70 COLOR= V: VLIN D,R AT L: COLOR= Z: FOR J
  = B + X - INT (X / U) * U TO R STEP U:
  PLOT L,J: REM WHITE.LINE
80 NEXT : NEXT : NEXT : GOTO 30

```

16 COLORS = 6





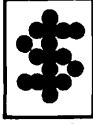



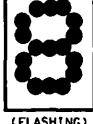
Perhaps you've noticed how some lo-res colors look the same as each other on a black-and-white monitor. Number 2 blue lines with a #4 green background might look fine in color, but be hard to see in black-and-white. Good computer graphics should work *both ways*. Either program in user-definable colors according to monitor type or avoid letting two colors touch that have the same black-and-white shade.

COLOR	B&W SHADE
0 Black	0 Black
1 Magenta	1 Dark Gray
2 Dark blue	1
4 Dark green	1
8 Brown	1
5 Gray	2 Solid gray
10 Gray	2
3 Lavender	3 Med. gray
6 Med. blue	3
9 Orange	3
12 Bright green	3
7 Light blue	4 Light gray
11 Pink	4
13 Yellow	4
14 Aqua	4
15 White	5 White

Here is a chart showing each standard Apple text character and its two corresponding lo-res colors:

		BOTTOM COLOR																	
		0	1	2	3	4	5	6	7	12	13	10	11						
TOP COLOR	0	@	P	sp	0	@	P	sp	0	@	P	sp	0	TOP COLOR	0	@	P	sp	0
	1	A	Q	!	1	A	Q	!	1	A	Q	!	1		1	A	Q	!	1
	2	B	R	"	2	B	R	"	2	B	R	"	2		2	B	R	"	2
	3	C	S	#	3	C	S	#	3	C	S	#	3		3	C	S	#	3
	4	D	T	\$	4	D	T	\$	4	D	T	\$	4		4	D	T	\$	4
	5	E	U	%	5	E	U	%	5	E	U	%	5		5	E	U	%	5
	6	F	V	&	6	F	V	&	6	F	V	&	6		6	F	V	&	6
	7	G	W	'	7	G	W	'	7	G	W	'	7		7	G	W	'	7
	8	H	X	(8	H	X	(8	H	X	(8		8	H	X	(8
	9	I	Y)	9	I	Y)	9	I	Y)	9		9	I	Y)	9
	10	J	Z	*	:	J	Z	*	:	J	Z	*	:		10	J	Z	*	:
	11	K	[+	;	K	[+	;	K	[+	;		11	K	[+	;
	12	L	\	/	<	L	\	/	<	L	\	/	<		12	L	\	/	<
	13	M]	-	=	M]	-	=	M]	-	=		13	M]	-	=
	14	N	^	.	>	N	^	.	>	N	^	.	>		14	N	^	.	>
	15	O	_	/	?	O	_	/	?	O	_	/	?		15	O	_	/	?
		INVERSE				FLASH				NORMAL									

For example:

	=	COLOR=1 (magenta)	COLOR=12 (br.green)		=	COLOR=4 (dk.green)	COLOR=10 (grey)		=	COLOR=8 (brown)	COLOR=11 (pink)
(NORMAL)		(GRAPHICS)		(NORMAL)		(GRAPHICS)		(NORMAL)		(GRAPHICS)	
	=	COLOR=1 (magenta)	COLOR=0 (black)		=	COLOR=4 (dk.green)	COLOR=2 (dk.blue)		=	COLOR=8 (brown)	COLOR=3 (lavender)
(INVERSE)		(GRAPHICS)		(INVERSE)		(GRAPHICS)		(INVERSE)		(GRAPHICS)	
	=	COLOR=1 (magenta)	COLOR=4 (dk.green)		=	COLOR=4 (dk.green)	COLOR=6 (med.blue)		=	COLOR=8 (brown)	COLOR=7 (lt.blue)
(FLASHING)		(GRAPHICS)		(FLASHING)		(GRAPHICS)		(FLASHING)		(GRAPHICS)	

CHARACTER PLOTS

Did you know that you can PLOT characters on the text screen? Don't ask why, just try it. This program plots an "A" in the upper left corner of the screen. You take it from there!

```
10 TEXT: HOME: PRINT CHR$(21)
20 COLOR=1: PLOT 0,0: COLOR=12: PLOT 0,1
```

Here is a chart showing each standard Apple text character and its two corresponding lo-res colors:

PDL ADJUST

To adjust your paddles so they won't send your cursor crashing off the hi-res screen, set variables $PX=279/255$ and $PY=191/255$. Then let $X=PDL(0)*PX$ and $Y=PDL(1)*PY$, and HPLOT X,Y or whatever.

ETCH-A-MESS

Hey kids! Draw some pictures all over your TV sets! To SAVE the hi-res picture, Reset out of the program and type BSAVE MONA LISA, A\$2000,L\$2000. To SAVE the lo-res one, Reset and type BSAVE WHISTLER'S SISTER,A\$400,L\$400.

```
10 REM
   =====
   PADDLE PLOTTER
   (LO-RES)
   =====
15 DIM C$(15)
20 PX = 39 / 255:PY = PX:C = 1
```

```

25 C$(0) = "BLACK":C$(1) = "MAGENTA":C$(2)
   = "DARK BLUE":C$(3) = "VIOLET":C$(4) =
   "DARK GREEN":C$(5) = "GRAY":C$(6) =
   "MEDIUM BLUE":C$(7) = "LIGHT BLUE"
26 C$(8) = "BROWN":C$(9) = "ORANGE":C$(10)
   = "GRAY":C$(11) = "PINK":C$(12) =
   "LIGHT GREEN":C$(13) = "YELLOW":C$(14)
   = "AQUA":C$(15) = "WHITE"
30 HOME : GR
40 XNU = PDL (0) * PX
50 YNU = PDL (1) * PY
60 XOLD = XNU:YOLD = YNU
70 XNU = PDL (0) * PX
80 YNU = PDL (1) * PY
100 COLOR= C: PLOT XNU,YNU
110 IF PEEK ( - 16384) > 127 THEN POKE -
    16368,0:C = C + 1: IF C > 15 THEN C = 0
130 VTAB 21: HTAB 1: PRINT "PADDLES CHANGE
    POSITION-> X:"; INT (XNU);"Y:"; INT
    (YNU);: CALL - 868
145 VTAB 23: HTAB 1: PRINT " ANY KEY
    CHANGES COLOR-> ";C;":":C$(C);: CALL -
    868
150 GOTO 60

```

```

10 REM
   =====
   PADDLE PLOTTER
   (HI-RES)
   =====
20 PX = 279 / 255:PY = 191 / 255
30 HOME : HGR
40 XNU = PDL (0) *PX
50 YNU = PDL (1) * PY
60 XOLD = XNU:YOLD = YNU
70 XNU = PDL (0) * PX
80 YNU = PDL (1) * PY
90 HCOLOR= 3
100 HPLOT XOLD,YOLD TO XNU,YNU
110 REM IF XOLD < > XNU OR YOLD < > YNU
    THEN HCOLOR= 0: HPLOT XOLD,YOLD TO
    XNU,YNU:REM (OMIT 1ST "REM" TO CHANGE)

```

```

120 VTAB 22
130 PRINT "X:"; INT (XNU);" "
140 PRINT "Y:"; INT (YNU);" "
150 GOTO 60

```

HI-RES CIRCLES

One of the most frequently asked questions around here (next to “Who zapped my disk?”) is “How do you draw a hi-res circle?” Well, here’s one way—

```

10 REM
=====
CIRCLE PLOTTER
=====
20 HGR : HCOLOR= 3:ST = .1
30 POKE - 16301,0: HOME : VTAB 22: HTAB 1:
  INPUT "X CENTER, Y CENTER, RADIUS:
  ";XCTR,YCTR,RAD:FLAG = 0:ST = .1: POKE
  - 16302,0
40 FOR I = 0 TO 6.3 STEP ST
50 X = RAD * COS (I) + XCTR
60 Y = (RAD / 4) * SIN (I) / .3 + YCTR
70 IF X > 279 OR X < 0 OR Y > 191 OR Y < 0
  THEN FLAG = 0: GOTO 100
80 IF NOT FLAG THEN HPLLOT X,Y:FLAG = 1
90 HPLLOT TO X,Y
100 NEXT : GOTO 30

```

Note the variable ST in line 30. It determines how many “sides” the circle will have. Change it and see.

LOOKER

Have you ever wondered what’s on your hi-res screens and learned to your dismay that a HGR or HGR2 will erase the screen? Of course, you can POKE your way there, but why bother when the following program will do it for you?


```

10 TEXT : HOME
15 VTAB 7: PRINT "HI-RES PAGE 1 OR 2? ";:
   GET A%: IF A% = 2 THEN GOTO 200
20 POKE - 16304,0: POKE - 16297,0: POKE
   -16300,0: POKE - 16302,0
30 WAIT - 16384,128: PRINT " "
40 TEXT : HOME : VTAB 7: INPUT "DO YOU
   WANT TO LOOK AT PAGE 2? (Y/N) ";AS$: IF
   AS$ = "Y" THEN GOTO 200
50 GOTO 320
200 POKE - 16304,0: POKE - 16297,0: POKE -
   16299,0: POKE - 16302,0
300 WAIT - 16384,128
310 TEXT: HOME : VTAB 7: INPUT "DO YOU WANT
   TO LOOK AT PAGE 1?(Y/N) ";AN$: IF AN$ =
   "Y" THEN GOTO 20
320 TEXT : HOME : VTAB 10: PRINT "   HERE'S
   LOOKIN' AT YOU KID": VTAB 23

```

THE MYSTERIOUS COLOR BIT

You may have wondered if there is a difference between the colors black #0 and black #4 and white #3 and white #7. Well, here's an experiment that proves these different colors really do exist:

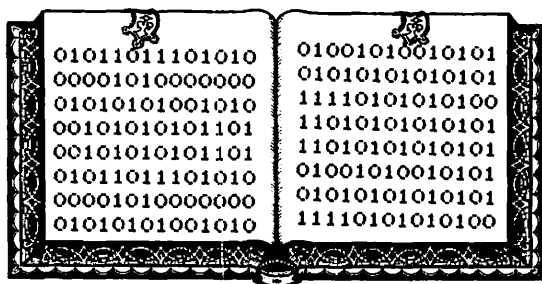
First, RUN *Alpha Plot* (from Beagle Bros, naturally) and *clear the screen* in a color *other than* white or black. Now, switch pages with Option 6-S and *clear the screen in color #0, black*. You now have a solid color on page 2 and a black screen on page 1.

Now, *set the color to #4* (the other black). At various places on the screen, draw some filled boxes and ellipses. Since you are drawing black on black, they won't look like much, but trust us. . . .

Here's the fun part—select Option 6-M for Merge. Select Merge Option #2, XDRAW page 1 onto 2. Now look at page 2. Your invisible figures have *appeared*! If you pick the same merge option again, they will disappear before your very eyes!

Now try this: go back to Drawing Mode on page 1. Use the Rubber

Band Cursor and line it up vertically over some of your invisible figures. Wherever the rubber band intersects your shapes, it will take a little sideways step and change colors. Once again, you can see the reality of the invisible color bit!



HI-RES PAGE 3

I don't know if Apple calls it page 3, but we do. We discovered by accident that you can store a full-screen hi-res image above page 2 and move it to one of the viewing pages (page 1 or page 2) anytime you want. There's a pointer at location 230 (\$E6) that tells you what page you are drawing on. Use the following POKES to change the drawing page:

```
POKE 230,32 (DRAW ON PAGE 1)
POKE 230,64 (DRAW ON PAGE 2)
POKE 230,96 (DRAW ON PAGE 3)
```

The page you are looking at, 1 or 2, is determined by HGR or HGR2, whichever is more recent, or by the display switches at -16299 and 16300:

```
POKE - 16300,0 (LOOK AT PAGE 1)
POKE - 16299,0 (LOOK AT PAGE 2)
```

Now, let's try a test. Type in the following program:

```
10 HGR: HGR2: HCOLOR=3: POKE 230,96: REM
   SET UP DRAW ON PAGE 3
20 HPLLOT 0,0 TO 279,191: HPLLOT 0,191 TO
   279,0: REM DRAW AN X (WON'T SHOW)
30 VTAB 21: END
```

You just drew a big X on page 3, and you're looking at a blank page 1. Page 2 is blank as well. To move the X to page 1 so we can see it, enter the monitor and move the picture in memory, like so:

```
CALL -151 <RETURN>
*2000<6000.7FFFFM <RETURN>
```

Page 1 resides in memory locations \$2000 to \$3FFF (8192 to 16383). Page 2 is \$4000 to \$5FFF (16384 to 24575). Page 3 is \$6000 to \$7FFF (24576 to 32766). Knowing this, you can move an image from one page to another by altering the numbers in the move command. Remember, however, that to *look* at a certain page, you have to throw the -16300 or -16299 switch, and you can't look at page 3.

We guess we should mention "page 0." In hi-res terminology it would encompass memory locations \$0 through \$1FFF, which includes the real page 0 (\$00-\$FF). "Drawing" on this page can be disastrous. This is exactly what happens if you turn on your Apple and start HPLOTting without doing an HGR or POKE 230,32. Because the value at location 230 is 0, you overwrite page 0 and really make a mess. Likewise, "page 4" is unusable, because that's where DOS is. Don't draw on DOS.

BLACK + WHITE = COLOR

You can't trust anything. Here's a program that makes color lines out of black and white. We'd explain it, but it's 2:00 A.M. and we're at a loss for words right now.

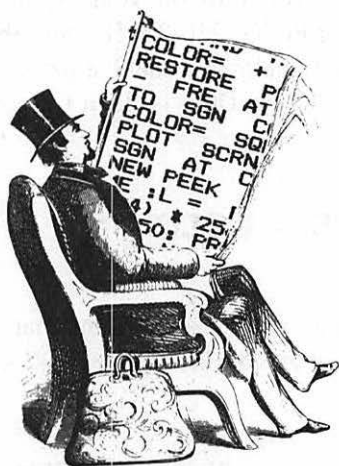
```
10 HOME : HGR : HCOLOR= 3: REM OR HCOLOR=7
20 FOR Y = 125 TO 155 STEP 10: HPLOT 0,Y TO
   279,Y: NEXT : REM PLOT WHITE LINES
30 FOR X = 0 TO 186 STEP 2
40 HCOLOR= 0: HPLOT X,125: HPLOT X + 1,135:
   REM ADD #0 BLACK TO MAKE #1 GREEN & #2
   VIOLET
50 HCOLOR= 4: HPLOT X,145: HPLOT X+ 1,155:
   REM ADD #4 BLACK TO MAKE #5 ORANGE & #6
   BLUE
```

```
60 NEXT X: VTAB 21
70 PRINT"<---- HCOLORS 1,2,5,6 ---->
  <-HCOLOR 3->"
```

AND FURTHERMORE . . .

Not only can you get color from black and white, but we found a way for you to make curves out of straight lines:

```
20 HGR : HOME : FOR C = 1 TO 6: VTAB 22:
  HTAB 18: PRINT "COLOR ";C: HCOLOR= C:
  FOR X = 1 TO 93: HPLT 2 * X + 50,0 TO 3
    * X,191: NEXT: NEXT
```



BEAGLE CARD FILE

If you follow our ads in the Apple magazines, you know that we always include little Applesoft programs to spice things up. Well, here's our all-time favorite, a little hi-res flick—

```

10 HGR: HGR2: POKE 232,120: POKE 233,64:
   POKE 16504,7: SCALE=80: P=16: X=99: FOR
   R=0 TO 31: P=P*-1: POKE 230,48+P
20 FOR Y=0 TO 1: ROT=ABS(64*Y-R):
   HCOLOR=3: FOR A=1 TO 25: DRAW 1 AT
   X+2*A,X: NEXT: HCOLOR=0: DRAW 1 AT X,X:
   ROT=32: DRAW 1: DRAW 1 AT X,X: NEXT Y,R
30 FOR A=0 TO 20: FOR B=0 TO 1: POKE
   49237-B,0: X=PEEK(49200): FOR C=1 TO
   6*A: NEXT C,B,A: GOTO 30

```

HI-RES FLIX

This program lets you flip between two hi-res pictures at a predetermined rate. Variable W1 controls the time page 1 shows, and W2 does the same for page 2.

```

10 TEXT : HOME
20 INPUT "PAUSE 1 (0-500):";W1
30 INPUT "PAUSE 2 (0-500):";W2
40 POKE -16304,0: POKE -16302,0: POKE
   -16297,0
50 POKE -16300,0: FOR X = 1 TO W1: NEXT :
   POKE -16299,0: FOR X = 1 TO W2: NEXT :
   GOTO 50

```

BLOAD any two hi-res pictures before you RUN the program by typing BLOAD PICTURE.1,A\$2000 <RETURN> AND BLOAD PICTURE.2,A\$4000 <RETURN>—\$2000 and \$4000 represent the addresses of pages 1 and 2. “PICTURE.1” and “PICTURE.2” are the names of your two hi-res pictures. RUN the program and notice the effect different rates of flicker have. If you want, remove the delay loops entirely from line 50.

A machine-language version of the same program follows. Notice how (humongously) much faster it is! It is *so* fast that if you use a small wait value (W1 or W2), a picture doesn’t have time to be completely “printed” on the screen before the page switch is made. You’ll have to stop the program with Reset.

```
10 TEXT : HOME : PRINT "FOR BEST EFFECT,  
BLOAD TWO DIFFERENT IMAGES ONTO HI-RES  
PAGES ONE & TWO.": PRINT : PRINT  
20 INPUT "PAUSE 1 (0-80):";W1  
30 INPUT "PAUSE 2 (0-80):";W2: IF W2 > 80  
OR W1 > 80 THEN 10  
35 PRINT : PRINT : PRINT "<CONTROL-RESET>  
TO QUIT...": FOR I = 1 TO 1234: NEXT  
40 POKE 768,141: POKE 769,84: POKE 770,192  
50 POKE 771,62: POKE 772,0: POKE 773,0  
60 FOR I = 4 TO W1 + 3: POKE 770 + I,234:  
NEXT  
70 POKE 770 + I,141: POKE 771 + I,85: POKE  
772 + I,192  
80 FOR J = 1 TO W2: POKE I + J + 772,234:  
NEXT  
90 POKE 772 + J + I,76: POKE 773 + J +  
I,0: POKE 774 + J + I,3  
100 POKE - 16304,0: POKE - 16302,0: POKE -  
16297,0  
110 CALL 768
```

ZERO PAGE PLOT

Turn your Apple off and on and RUN this program. (If you have an Apple //e or Apple //c, wait about thirty seconds after you turn it off.)

```
20 HOME: GR: REM (That's GR, not HGR)  
30 FOR X = 0 TO 255  
40 HPL0T X,Y  
50 PRINT X  
60 NEXT
```

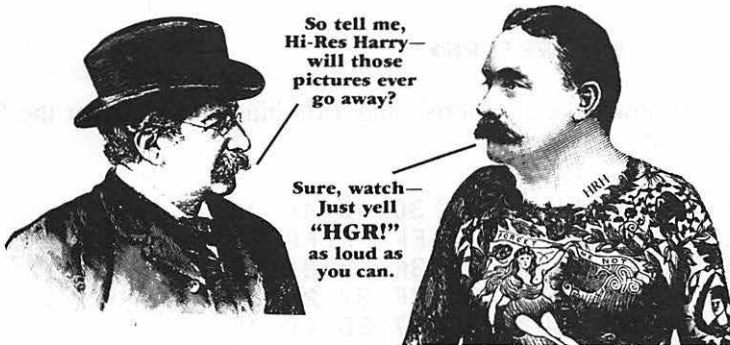
You have just drawn all over page 0! *Weird things* happen if you don't set the page pointer at 230. It should be set to 32 or 64 for hi-res page 1 or 2. HGR and HGR2 do this for you, but when you turn on your Apple it's set to 0. See your PEEKs and POKEs chart. (Reboot after RUNning.)

560-PLOT HI-RES

This program proves the existence of 560 horizontal hi-res plots. First, a normal bumpy hi-res line is drawn from 0,0 to 75,150 (DX,DY). Then we draw a parallel supersmooth (but slow) line.

```
10 HGR : HOME : VTAB 21: HTAB 12: PRINT
   "!! !": HTAB 12: PRINT "!! !DOUBLE (560-
   PLOT) HI-RES": HTAB 12: PRINT "!!": HTAB
   12: PRINT "!!NORMAL HI-RES";
20 DX = 80:DY = 160
40 FOR Y = 0 TO DY
50 X = 27 + 2 * (DX * Y / DY)
60 ODD = INT (X) - INT (X / 2) * 2
70 HCOLOR= 3 + 4 * ODD
80 X = X / 2
90 HPOINT X,Y: REM OR HPOINT X,Y TO X+1,Y
100 NEXT : VTAB 23
110 HCOLOR= 3: HPOINT 0,0 TO DX,DY: REM
    NORMAL LINE
```

If the column number is *odd*, we use HCOLOR #7 white. If it's *even*, we use #3 white. These two colors plot in either the left or right half of one of the 280 hi-res columns. Our example here cheats just a little by picking special coordinates for extra smoothness.



HI-RES PULSE METER

```

10 HGR2 : HCOLOR= 3: FOR X = 768 TO 789:
    READ J: POKE X,J: NEXT: DATA 0,0,96,173,
    48,192,136,208,4,198,1,240,8,202,208,
    246,166,0,76,3,3,96
20 HPLOT 0,0: FOR P = 1 TO 254 STEP 3: POKE
    0,P: POKE 1,4: CALL 771: POKE 0,255 - P:
    POKE 1,4: CALL 771: HPLOT TO INT ( RND
    (1) * P),.7 * P: NEXT

```

SHAPE-TABLE TALK

Let's assume you know nothing about shape tables (you're not the only one). Here are a few little experiments that will at least give you a taste of the fantastic animation possibilities.

Using shape tables is easy; it's *making* them that is usually difficult. If you've looked in the Applesoft manual in the shape-table chapter, you'll see why. We're not going to go into how to make shape tables here. Beagle Bros has a disk called *Shape Mechanic* that will write shape tables *for you*; all you do is draw your shape on the screen and the shape table is automatically written by your Apple. Many good features are included.

Here is a sample shape table containing five shapes. Type FP <RETURN> to clear memory. Now enter the monitor by typing:

```
CALL -151 <RETURN>
```

You should now have an asterisk and a flashing cursor. Enter the following shape table:

```

6000- 05 00 10 00 30 00 40 00
6008- 50 00 58 00 FF FF FF FF
6010- 2D 2D 2D 3E 3F 3F 37 2D
6018- 2D 2D 3E 3F 3F 37 2D 2D
6020- 2D 3E 3F 3F 37 2D 2D 2D
6028- 3E 3F 3F 3F 00 FF FF FF
6030- 08 0C D6 AA DF 63 1C 08
6038- 1C 07 00 FF FF FF FF FF

```



```
6040- 12 3F 20 64 2D 15 36 1E
6048- 07 00 FF FF FF FF FF FF
6050- 07 00 FF FF FF FF FF FF
6058- 3F 37 2D 35 37 4D C0 2C
6060- 1C 3F 3F 4C E5 07 00 FF
6068- FF FF FF FF FF FF FF FF
```

In case you've never entered machine code before, just type:

```
6000: 05 00 10 00 30 00 40 00 <RETURN>
```

and

```
6008: 50 00 58 00 FF FF FF FF <RETURN>
```

and so on. Notice the *colon* (not a *hyphen*) after the four-digit number; very important! Now save the shape table by typing:

```
BSAVE SHAPES,A$6000,L$70
```

Anytime you want these shapes back, type:

```
BLOAD SHAPES
```

Now tell your Apple where you put the shape table by typing:

```
POKE 232,0: POKE 233,96
```

You could make these POKES part of a program. 232–233 is the start of the shape-table pointer. See your PEEKs and POKES chart.

Now we can play with the Apple commands DRAW, XDRAW, ROT, and SCALE. Try each of the following programs.

```
10  REM
    =====
    SHAPE DISPLAY
    =====
15  TEXT : HGR : ROT= 0: SCALE= 1: POKE
    232,0: POKE 233,96: PRINT CHR$ (4);
    "BLOAD SHAPES"
```

```
16 ROT= 0: SCALE= 1: HCOLOR= 3
20 FOR SH = 1 TO 5
30 DRAW SH AT SH * 21,150
40 NEXT SH
50 HOME : VTAB 21
60 HTAB 4: PRINT "1 2 3 4 5"

20 REM
=====
ROT TEST
=====
30 HOME : HGR : SCALE= 1: HCOLOR= 3: POKE
  232,0: POKE 233,96: PRINT CHR$(4);
  "BLOAD SHAPES"
35 HPLOT 0,150 TO 279,150
40 FOR X = 50 TO 250 STEP 1
80 R = R + 16: IF R > 48 THEN R = 0: REM
  INCREASES ROTATION 90 DEGREES
82 VTAB 21: HTAB X / 7 - 4: PRINT SPC(1);
  "ROT";R;: CALL - 868: REM PRINTS ROT
  VALUE
83 VTAB 22: HTAB X / 7 - 2: PRINT SPC
  (1);"X:";X: REM PRINTS X POSITION
85 ROT= R: XDRAW 5 AT X,140: REM DRAWS
  ARROW
90 FOR I = 1 TO 99: NEXT I
95 XDRAW 5 AT X,140: REM ERASES ARROW
100 NEXT X

20 REM
=====
SCALE TEST
=====
30 TEXT : HOME : HGR : ROT= 0: POKE 232,0:
  POKE 233,96: PRINT CHR$(4);"BLOAD
  SHAPES"
40 FOR S = 1 TO 10
80 SCALE= S
82 VTAB 21: HTAB 10: PRINT "SCALE:";S;:
  CALL - 958
85 XDRAW 1 AT 70,30: REM DRAWS BOX
90 IF S = 1 THEN FOR I = 1 TO 500: NEXT I
```

```

95  XDRAW 1 AT 70,30: REM ERASES BOX
100 NEXT S: GOTO 40

```

Fool around with these programs and change some of the numbers. Substitute DRAW for XDRAW and see what happens (be sure you have HCOLOR set for something other than black). XDRAW draws in the *opposite* of whatever the background color is. That's how you can erase a drawing; just XDRAW over it in exactly the same position! Notice how SCALE, when it's bigger than 1, blows up a solid shape in a linear fashion. Too bad it won't blow up solids.

HI-RES COLLISIONS

Here's an experiment to demonstrate hi-res collisions, handy if you feel the urge to write a SDTFS* game. We have read two articles, each more than four blocks long, about how to simulate the lo-res SCRNB function for determining the color of a hi-res point. Well, forget it. There's a handy PEEK you can do of location 234 that tells you if the last XDRAW was done over a black or nonblack point. This program pretty much explains itself. Oh, *be sure the shape table from the previous section is in memory* and the pointers are set—POKE 232,0:POKE 233,96—or this program won't work.

*Shoot Down the Flying Saucers

```

10  REM
=====
COLLISION TEST
=====
20  TEXT : HOME : HGR : ROT= 0: SCALE= 1:N
    = 100: POKE 232,0: POKE 233,96: PRINT
    CHR$(4);"BLOOD SHAPES"
30  HCOLOR= 6: HPLLOT 250,0 TO 250,191:
    HPLLOT 30,0 TO 30,191
40  FOR X = N + 2 TO 279
50  XDRAW 3 AT X,100

```

```
60  XDRAW 3 AT X,100
70  IF PEEK (234) = 0 THEN NEXT X
80  FOR N = X -1 TO 0 STEP - 1
90  XDRAW 2 AT N,100
100 XDRAW 2 AT N,100
110 IF PEEK (234) = 0 THEN NEXT
120 GOTO 40
```

LO-RES MYSTERY

You can't PLOT X,Y where $X > 39$, *but* you can PRINT SCRN(X,Y) with values of X up to 47! Check it out. It seems that there's an invisible lo-res screen to the right of the visible screen that's 48 plots high by 8 plots wide. Maybe Apple is tooling up for CinemaScope!

CHAPTER 12

INTEGER (THE OTHER BASIC)

INTEGER, //c, AND PRODOS

You may not have realized this but we found out the hard way. If you boot Apple's ProDOS on your Apple //c and key in INT, you get a "HUH?" error. You also get a "NEVER HEARD OF IT?" error if you FP. However, if you boot a good old DOS 3.3 System Master, everything works just fine and you can INT and FP to your heart's content.

SPEED TEST

Another little-known fact about Integer BASIC is that it is faster than Applesoft BASIC. To see for yourself, time the following program in both Applesoft and Integer with a stopwatch:

```
10 CALL -936
20 PRINT "START"
30 FOR X = 1 TO 30000
40 PRINT X
```

```
50 CALL -936
60 NEXT X
70 PRINT "END"
80 END
```

In the olden days before very many Apple programmers wrote assembly/machine-language games, most of the games were written in Integer for better speed. Look at the old club disks with games and you'll see the "I" next to a lot of them.

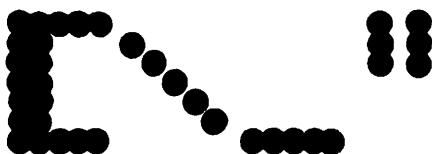
FLASH IN I.B.!

To flash alphabetical keys in Integer BASIC, simply POKE 50,127.

Here is the Beagle Bros Integer flash subroutine that will flash *all* characters:

```
10 REM INTEGER FLASH SUBROUTINE
   BY BERT KERSEY
20 DIM A$ (40) : FLASH=1000
22 CALL -936
25 XTAB= RND (10)+1: YTAB= RND (21)+2:
   VTAB 1: TAB 1
30 PRINT "TYPE SOMETHING:": VTAB YTAB :
   TAB XTAB: INPUT A$
40 VTAB YTAB: TAB XTAB: PRINT A$: GOSUB
   FLASH: GOTO 25
50 END
1000 FOR CHR=1 TO LEN (A$)
1010 XPOS= PEEK (36)-1+XTAB
1020 YPOS= 2* PEEK(37)-1
1030 COL= SCRN (XPOS-1+CHR,YPOS)
1040 COLOR=COL-4
1050 IF COL>11 THEN COLOR=COL-8
1060 PLOT XPOS-1+CHR,YPOS
1070 NEXT CHR: RETURN
```

For clarification of what goes on here, read the following:



Use Graphics Commands in Text Mode to:

PRINT THE UNPRINTABLES

Certain Apple text characters don't seem to want to appear on the video screen on the old Apple II's and II+'s. Namely [, \, —, and “. CHR\$ will retrieve these for you in Applesoft, but not in Integer BASIC. And sure, FLASH flashes fine in Applesoft, and an Integer POKE 50,127, while not widely publicized, will flash characters ASC 192 and above (A through Z and SHIFT-M, -N, and P). But what about flashing numbers and punctuation?

A Solution: As we all know, Apple's lo-res colors have corresponding characters in the text mode. Hit Reset or type TEXT while viewing a lo-res picture, and you will see eight hundred or more text characters—some flashing, some inverse, and some normal. Keep looking. Some of these characters are probably the “unprintables.” Conversely, a POKE -16304,0 will convert a text display to its corresponding lo-res “picture” with each text character represented by a pair of lo-res colors.

Every Apple text character has a corresponding pair of stacked color blocks in the graphics mode.

To put some of our “unprintables” on the screen, let's approach things a bit backward. To print an A at VTAB 1, TAB 1, the hard way, try the program on page 109. Now try this:

```
10 TEXT : CALL -936
20 VTAB 10: PRINT "SHE SAID, 'YOU CAN'T
   PRINT QUOTE MARKS!'"
30 COLOR=2: PLOT 10,18: PLOT 39,18
99 END
```

The apostrophes in line 20 have an upper color of 7 and a lower color of 10. Line 30 changes the upper color to 2 and produces quote marks (upper color 2, lower color 10). Change line 30's COLOR to 11 to produce + 's, or to 15 to get / 's.

To find plotting coordinates X and Y for a text character at TAB XT and VTAB YT, use $X=XT+1$ and $Y=2*YT-2$ (upper half) or $Y=2*YT-1$ (lower half).

This little Integer program flashes some numbers at you, something POKE 50,127 won't accomplish:

```
10 TEXT : CALL -936
20 PRINT "FLASH: 12348"
30 COLOR=7: HLIN 7,11 AT 1 :END
```

Line 30 changes the lower color of all the numbers from 11 to 7 and produces flashing numbers. COLOR=3 would produce inverse numbers. COLOR=13 changes things entirely. Try it, and compare your results with the chart on page 111.

Here is a program that unleashes the truly unprintable (in I.B., anyhow) underscore:

```
10 TEXT: CALL -936
20 FOR N=1 TO 5
30 VTAB N: PRINT "AMOUNT #";N; "???????"
40 COLOR=13: HLIN 10,14 AT 2*N-1
50 VTAB N: TAB 10: INPUT A
60 VTAB N: TAB 10: PRINT":";A;"  ": CALL
  -958
99 NEXT N : END
```

Now you're on your own! As you can see, any character can be printed on the screen flashing, inverse, or normal, with a little work. Some clever use of GOSUB statements in a long program will keep the keyboard banging to a minimum.

Have fun!

INTEGER ADVANTAGE

One of the gigantic advantages of Integer BASIC is the ability to use variables as line numbers, a great self-documentation feature. For example, if you have a card-shuffling routine at line 2416, you can use SHUFFLE=2416. Then every time you want to shuffle, simply GOSUB SHUFFLE. Try that in Applesoft, and the computer goes, "Huh?"

APPLESOFT VS. INTEGER BASIC

One way or another every Apple II (+,e,c) supports two BASICs. (Even if you have a 32K or 48K Apple, there are ways of getting Integer or Applesoft into memory, regardless of your ROM. On club disks there are Integer/Applesoft programs that can load into 48K for sure, and probably 32K if you're very careful. This doesn't mean anything to Apple //e and //c users who have mucho K to play with; so don't worry about it.) The very first BASIC on the Apple was Integer BASIC, but it was replaced by Applesoft BASIC, and while the two BASICs are different, for the most part they are similar.

Unfortunately, there are significant syntax differences between the two languages and to convert a program from one to the other, you must know these differences:

INTEGER	APPLESOFT
TAB	HTAB
POKE 50,63	INVERSE
POKE 50,127	FLASH
POKE 50,255	NORMAL
CALL -936	HOME
<> or # (not =)	<> or ><
X MOD Y	X-INT (X/Y)*Y
X/Y	INT (X/Y)
RND (X)	INT(RND(1)*X)
A\$(LEN(A\$)+1)=	A\$=A\$+
A\$(X,X)	MID\$(A\$,X,1)

INTEGER	APPLESOFT
A\$(4,6)	MID\$(A\$,4,3)
A\$(1,3)	LEFT\$(A\$,3)
NEXT X	NEXT
GOTO A*100	ON A GOTO 100,200 . . .
GOSUB A*100	ON A GOSUB 100,200 . . .
INPUT "WHAT?";W	INPUT "WHAT";W

Other Differences: (1) All ASC values are lower by 128 in Applesoft, although PEEK (-16384) returns the high number in both languages; (2) if an IF statement is *not* true, Applesoft will *not* read the rest of a program line (a big difference!); (3) Applesoft has many more commands than Integer, and Integer has a few not available in Applesoft. Check your Beagle Bros Command chart. It lists all commands and their functions.

To convert to Applesoft with a disk system:

1. Add these lines to your Integer program:

```
32764 D$="": REM (CTRL-D)
32765 PRINT D$; "NOMON C,I,O": PRINT D$; "OPEN
      CONVERTFILE": PRINT D$; "WRITE CONVERTFIL
      E": CALL -936
32766 POKE 33,127
32767 LIST 0,32763: PRINT D$;"CLOSE": TEXT:
      END
```
2. RUN 32764 <RETURN>
3. FP <RETURN>
4. EXEC CONVERTFILE <RETURN>
5. Correct all syntax. We recommend Roger Wagner's *Apple DOC* disk to make this task much, much simpler.

To convert Applesoft to Integer:

1. Correct all syntax.
2. Add the lines of the preceding program *without* line 32766.
3. RUN 32764 <RETURN>
4. INT <RETURN>
5. EXEC CONVERTFILE <RETURN>

CHAPTER 13

MACHINE LANGUAGE

ENTERING MACHINE CODE

As beginners, we were frustrated when presented with even a simple assembly-language program in a magazine because we just *didn't know how to type it in!* Well, here's how! This simple (and useless) little program prints flashing, normal, and inverse fonts on the screen.

```
0300- A0 FF      LDY  #$FF
0302- 98          TYA
0303- 88          DEY
0304- 20 ED FD   JSR  $FDED
0307- C0 00      CPY  #$00
0309- D0 F7      BNE  $0302
030B- 60          RTS
```

Or, you might see it like this:

```
0300- A0 FF 98 88 20 ED FD C0
0308- 00 D0 F7 60 00 00 00 00
```

Here's how you enter this program:

First, CALL -151 <RETURN> to enter the monitor. This will produce an asterisk and a cursor. Now type

```
300:  A0 FF 98 88 20 ED FD C0 00 D0 F7 60
```

and hit RETURN. That's it! Now go back to BASIC with Reset or Control-C! To RUN the program from BASIC, CALL 768 (decimal 768=hex 300, the number of the first program line). To LIST it, type CALL -151 <RETURN> and 300L <RETURN>.

THE MINI-ASSEMBLER

One of the best kept secrets of the Apple is its built-in mini-assembler. To use it, all you have to do is get Integer BASIC into memory. On Apple //c's this requires using a DOS 3.3 System Master instead of ProDOS. The old Apple II comes with Integer in ROM, but for the others, the easiest way to get Integer is just to boot the System Master and enter INT <RETURN>.

To enter the mini-assembler, use one of the following methods:

1. Enter the monitor with CALL -151 <RETURN> and then enter F666G <RETURN>.
2. From INT BASIC enter CALL -2458 <RETURN>.

As just about no one knows, the prompt for the mini-assembler is an exclamation mark (!). When you see that, you are in the mini-assembler. Now, you can enter assembly-language programs using mnemonic opcodes instead of machine-language opcodes (numbers). Here's how, using the same useless program entered earlier.

```
!300: LDY #FF <RETURN>
0300-  A0 FF      LDY    #$FF
! TYA <RETURN>
0302-   98      TYA
! DEY <RETURN>
```

```

0303-    88          DEY
! JSR FDED <RETURN>
0304-    20 ED FD   JSR    $FDED
! TYA <RETURN>
0307-    98          TYA
! BNE $0302 <RETURN>
0308-    D0 F8      BNE    $0302
! RTS <RETURN>
030A-    60          RTS
!$FF69G <RETURN>
*
```

To define the starting address, you enter the hexadecimal value and a colon. Then put a space and start entering your opcodes. For entering opcodes in the mini-assembler, you *must* put a space after the exclamation mark (!) prompt, and a space between the opcode and operand. The \$FF69G (with no space after the prompt) returns you to the monitor. Enter 300G <RETURN> and your program will execute. If this doesn't impress your friends, nothing will!

If you want to SAVE this gem, enter BSAVE THIS GEM,A\$300,L11 <RETURN>. Do it from the monitor to further freak your acquaintances.

BYTE COUNTING

To figure out the number of bytes in a program so that you will know the length of L in a BSAVE, you can do one of the following:

1. Count the machine opcode/operands in your program. (These are the hexadecimal values next to the addresses. There are from one to three after each address.)
2. Subtract the starting address from the ending address and add 1. In our example you'd subtract \$300 from \$30A. From the monitor, enter:

```

30A-300 <RETURN>
=0A
0A + 1 = 0B
```

Your L\$ = B.

DISASSEMBLIST

If you want to list DOS or some humongous binary program to your printer, you have to type:

```
PR#1
CALL -151
*9000L
```

. . . then wait for the screen to fill, and type another L, and so on, until your L finger has had it. But wait! Try this—

```
*9000LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
```

BINARY TO BASIC

With all of the trouble it takes to learn machine/assembly language, why on earth would anyone in their right mind want to convert a binary program to a BASIC one. We don't know either, but just in case you find out, here's a way to do it.

```
10 TEXT : HOME :K = 9000:N = 0:D$ = CHR$
   (4): REM DOS 3.3 ONLY (NOT PRODOS)
20 NORMAL : VTAB 3
30 PRINT "DO YOU WANT A DRIVER IN THE
   BASIC FILE?": PRINT "=> (Y/N) ";; GET
   DRIVER$: PRINT DRIVER$ : PRINT
40 INVERSE : INPUT "NAME OF FILE TO BLOAD
   (OR CAT)=> ";BF$:
50 IF BF$ = "CAT" THEN PRINT D$"CATALOG":
   GOTO 40
```

```

60 IF BF$ = "" THEN TEXT : END
70 PRINT "WHICH DRIVE FOR BLOAD?";: GET
  D%: PRINT D% IF D% >2 OR D% <1 THEN TO
80 IF BF$= "" THEN PRINT : GOTO 100
90 PRINT D$"BLOAD";BF$;","D";D%
100 INPUT "NAME OF FILE TO SAVE: ";FI$
110 PRINT "DRIVE TO SAVE FILE?";: GET DS%:
  PRINT DS%
120 A = PEEK (43634) + 256 * PEEK (43635)
130 B = PEEK (43616) + 256 * PEEK (43617) -
  1
140 PRINT D$"OPEN"FI$;".T,D"D%: PRINT D$
  "WRITE"FI$".T"
150 IF DRIVER$ = "Y" THEN GOSUB 350
160 PRINT K;"DATA";
170 FOR I = A TO A + B
180 N = N + 1
190 IF N = 5 OR I = A + B THEN PRINT PEEK
  (I): PRINT : GOTO 220
200 IF N = 6 THEN K = K + 10:N = 0: PRINT
  K;" DATA ";
210 PRINT PEEK (I);",";
220 NEXT I
230 PRINT D$"CLOSE"FI$".T"
240 PRINT : PRINT "TYPE" CHR$ (34) "NEW"
  CHR$ (34)", THEN ", : PRINT "EXEC
  "FI$". T": PRINT "TO CREMATE YOUR
  BASIC PROGRAM. ":END
300 REM
320 REM***DRIVER***
350 PRINT "10 FOR I= ";A;" TO ";A + B:
  PRINT
360 PRINT "20 READ D: POKE I,D : NEXT I":
  PRINT
370 PRINT "30 CALL ";A: PRINT
380 RETURN

```

65C02

The Apple //c and the newer //e's and //e ROMs support the new machine-language commands: PHX, PHY, PLX, PLY, STZ, TRB, and TSB. Many

new versions of old 6502 commands have also been made available. Here is an alphabetized list of all the 65C02 commands. The first batch works with the 6502s and 65C02s. The second batch is available only on the //c's and souped up //e's with the 65C02 processors.

OPCODE

ADC	Add Memory to Carry a A-reg
AND	Logical AND memory to A-reg
ASL	Arithmetic Shift bits Left
BCC	Branch if Carry Clear
BCS	Branch if Carry Set
BEQ	Branch if Equal Zero
BIT	Test A-reg bits with memory
BMI	Branch if Minus
BNE	Branch if Not Equal to zero
BPL	Branch if Plus
BRK	Software interrupt—BREAK
BVC	Branch if Overflow Clear
BVS	Branch if Overflow Set
CLC	Clear Carry flag
CLD	Clear Decimal flag
CLI	Clear Interrupt flag
CLV	Clear Overflow flag
CMP	Compare Memory and A-reg
CPX	Compare Memory and X-reg
CPY	Compare Memory and Y-reg
DEC	Decrement memory by 1
DEX	Decrement X-reg by 1
DEY	Decrement Y-reg by 1
EOR	Exclusive OR memory and A-reg
INC	Increment memory by 1
INX	Increment X-reg by 1
INY	Increment Y-reg by 1
JMP	Jump to new address
JSR	Jump but Save Return address
LDA	Load A-reg from memory
LDX	Load X-reg from memory
LDY	Load Y-reg from memory
LSR	Logical Shift bits Right
NOP	No Operation

ORA	Logical OR Memory and A-reg
PHA	Push A-reg onto stack
PHP	Push P-reg onto stack
PLA	Pull A-reg from stack
PLP	Pull P-reg from stack
ROL	Rotate bits Left
ROR	Rotate bits Right
RTI	Return from Interrupt
RTS	Return from Subroutine
SBC	Subtract memory from A-reg
SEC	Set Carry flag
SED	Set Decimal flag
SEI	Set Interrupt flag
STA	Store A-reg into memory
STX	Store X-reg into memory
STY	Store Y-reg into memory
TAX	Transfer A-reg into X-reg
TAY	Transfer A-reg into Y-reg
TSX	Transfer S-reg into X-reg
TXA	Transfer X-reg into A-reg
TXS	Transfer X-reg into S-reg
TYA	Transfer Y-reg into A-reg

On 65C02 Only

BRA	Branch Always
PHX	Push X-reg onto stack
PHY	Push Y-reg onto Stack
PLX	Pull X-reg from stack
PLY	Pull Y-reg from stack
STZ	Store Zero
TRB	Test and Reset Bits
TSB	Test and Set Bits

UNSUNG OPCODES

Here are some opcodes we suggested Apple should include, but no one will listen to us. . . .

CAB	Crash And Burn
ASB	Ask Somebody
RTM	Read The Manual
GHL	Go Have Lunch
CYM	Call Your Mother
GBO	Get Bugs Out
BBB	Buy Beagle Bros
DAB	Drink A Beer

UNUSED HEX CODES

We were wondering which of the 256 available hex codes (\$00-\$FF) were used for commands and which were unused. The following chart shows what we found. If you squint, you can see a pattern. . . . Get out your decoder rings; there might be a hidden message here!

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
\$00	x	x	u	u	c	x	x	u	x	x	x	u	c	x	x	u
\$10	x	x	c	u	c	x	x	u	x	x	c	u	c	x	x	u
\$20	x	x	u	u	x	x	x	u	x	x	x	u	x	x	x	u
\$30	x	x	c	u	c	x	x	u	x	x	c	u	c	x	x	u
\$40	x	x	u	u	u	x	x	u	x	x	x	u	x	x	x	u
\$50	x	x	c	u	u	x	x	u	x	x	c	u	u	x	x	u
\$60	x	x	u	u	c	x	x	u	x	x	x	u	x	x	x	u
\$70	x	x	c	u	c	x	x	u	x	x	c	u	c	x	x	u
\$80	c	x	u	u	x	x	x	u	x	c	x	u	x	x	x	u
\$90	x	x	c	u	x	x	x	u	x	x	x	u	c	x	c	u
\$A0	x	x	x	u	x	x	x	u	x	x	x	u	x	x	x	u
\$B0	x	x	c	u	x	x	x	u	x	x	x	u	x	x	x	u
\$C0	x	x	u	u	x	x	x	u	x	x	x	u	x	x	x	u
\$D0	x	x	c	u	u	x	x	u	x	x	c	u	u	x	x	u
\$E0	x	x	u	u	x	x	x	u	x	x	x	u	x	x	x	u
\$F0	x	x	c	u	x	x	x	u	x	x	c	u	u	x	x	u

x = 6502 and 65C02 c = 65C02 only u = unused

WORD.CALL

Running this program will make a subsequent CALL print anything you want. Substitute the word you want in line 20.

```

10 LOC = 16384: REM CALL 16384 TO EXECUTE
20 A$ = "DUDLEY DOORIGHT"
50 FOR A = LOC TO LOC + 13: READ B: POKE
  A,B: NEXT : DATA 160,0,185,0,0,32,237,
  253,200,192,0,208,245,96
60 POKE LOC + 3,A - INT (A / 256) * 256:
  POKE LOC + 4, INT (A / 256)
80 POKE LOC + 10, LEN (A$): FOR J = 1 TO
  LEN (A$): POKE A + J - 1, ASC (MID$
  (A$,J)) + 128: NEXT

```

Here is the machine-language program that is produced by RUNning the preceding Applesoft program. To see it, type CALL -151 <RE-

TURN> and 4000L <RETURN>. You can execute the program by typing 4000G <RETURN>. You can get back to BASIC by typing Control-C <RETURN>.

4000-	A0 00	LDY #\$00	Load Y with a zero.
4002-	B9 0E 40	LDA \$400E,Y	Load A with the value at \$400E+Y.
4005-	20 ED FD	JSR \$FDED	Print the character in A.
4008-	C8	INY	Increase Y by 1.
4009-	C0 0F	CPY #\$0F	Is it equal to the length of the word yet?
400B-	D0 F5	BNE \$4002	If not, go to \$4002.
400D-	60	RTS	The end.
400E-	C4 D5		ASCII values for the text to be printed.
4010-	C4 CC		

BIT SPLITTER

Want to know how to break a decimal number into bits?

```

10 INPUT "NUMBER: ";N
15 IF N>255 OR N<0 THEN 10
20 FOR BIT=1 TO 8: HTAB 9-BIT
30 PRINT N-INT(N/2)*2;
40 N=INT(N/2): NEXT

```

THE WORLD'S CLOSE-TO-SHORTEST HEX CONVERTER

```

10 INPUT "DECIMAL NUMBER:";A$: A=VAL(A$)
20 POKE 71,A/256: POKE 70,A-256*PEEK(71)
30 POKE 58,64: POKE 59,249: PRINT "=$";:
CALL -327

```

SPEED CALL

Why type CALL -151 to enter the monitor? CALL 1 saves three keystrokes!
Bonus: A *beep*!

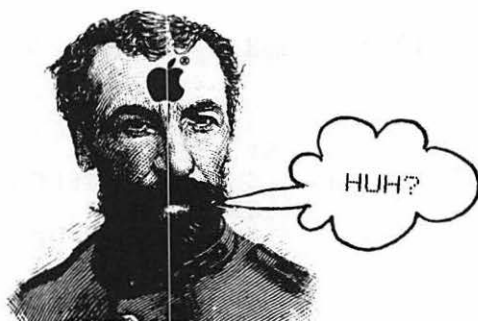
BRUN YOUR PICS!

Normally, if you BRUN (instead of BLOAD) a hi-res pic, it crashes.
To prevent that:

- | | | |
|-----------|----|---------------------------------|
| Pg. 1 Pic | 1. | POKE 8191, 96 |
| | 2. | BSAVE PICNAME, A\$1FFF, L\$2000 |
| Pg. 2 Pic | 1. | POKE 16383,96 |
| | 2. | BSAVE PICNAME, A\$3FFF, L\$2000 |

CHAPTER 14

DISKS AND DOS



COME AGAIN?

To get all of you off on the right foot in this chapter, we will have a short lesson on pronouncing words you *read* but may not get enough practice saying:

DOS rhymes with *boss*. Or, if you know German or watch reruns of *Hogan's Heroes*, it is pronounced *DAS*. On the other hand, *DOS* is

not pronounced as the Spanish word *dos*, which means “two.” Therefore, while you may correctly say you have the old DOS (das), three-point-dos (dose), if you refer to your DOS as dose, people may think you’re referring to two disk drives, 3.2, or your IQ.

Modem rhymes with “rode ’em.”

EPROM doesn’t rhyme with anything. In fact, no one has ever said EPROM out loud.

WHAT’S DOS?

DOS is Apple’s “Disk Operating System.” Without it, your Apple simply does not know how to perform any function that involves a disk—load a program from disk, catalog, tell you “FILE LOCKED,” access a text file, etc., etc., let alone start the motor on your disk drive. Without DOS, your Apple *does* know how to execute BASIC functions (we assume you are using Applesoft or Integer BASIC—most likely Applesoft) because BASIC is built in to your Apple in the form of unchangeable hardware, or ROM (“Read Only Memory”). DOS is actually a complex machine-language *program* that is entered into your Apple’s memory in RAM (“Random Access Memory”) each time you boot a disk. DOS normally remains in memory as long as your Apple is turned on and is not affected or changed by anything you ordinarily do—programming, loading, saving, deleting, etc.

So, after you boot a disk (load DOS), your Apple knows *two* sets of instructions, BASIC and DOS. When you enter an instruction through the keyboard, the Apple checks it *first* to see if it is a DOS command, *then* to see if it is a BASIC command. If you type ABCFED with a carriage return, for example, the Apple checks its entire 28-word DOS command vocabulary (words like CATALOG, RENAME, DELETE) to see if it knows ABCFED. If it doesn’t, it then checks its BASIC vocabulary (words like LIST, GOTO, AND, NEXT, POKE). If it can’t find ABCFED there, it gives up and prints “?SYNTAX ERROR” or “*** SYNTAX ERR.” If it does know the word that you have typed, it executes the command

according to the instructions that reside in memory, either DOS or BASIC, depending on where the command was found. All of this takes approximately no time at all.

WHAT'S DOS BOSS?

Since DOS is an *accessible* written program in RAM and not a permanent collection of hardware like BASIC, you can *change* it to suit your desires and to have more control over your computer. *DOS BOSS* (by Beagle Bros, natch) is the key to making these changes. With *DOS BOSS*, you will have immediate access to DOS's most visible functions and features. With this book, you will have even further control and learn a bit more about what goes on inside your Apple's "brain." We have written as much as possible from a beginner's viewpoint, assuming that you know nothing of machine-level programming or the way a computer works. Technical details, whenever possible, have been omitted or written in English. (By the way, if you have an Apple //c, use your System Utilities disk to format a 3.3 disk. ProDOS comes with your //c, but DOS 3.3 works fine, and most of the information we have so far is for DOS 3.3. The easiest way to get DOS 3.3 is to choose DOS 3.3 in the "format" choice on your System Utilities and initialize a blank disk with it. Anytime you boot the initialized 3.3 disk, DOS 3.3 will be placed in your //c's memory.)

CHANGING DOS

Depending on what kind of Apple you have, you've got 32K (real old Apples), 48K (pretty old Apples and no 16K card), 64K (recently old Apples), or 128K (Apple //c or big K card in old Apple slot). Each K stands for 1,024 bytes, but we'll round it off to 1,000 to make things simpler. Thus, you have 32,000, 48,000, 64,000, or 128,000 bytes of

RAM, or changeable memory. (If you have a 48K machine with a 32K RAM card, leave the room.) Since DOS is loaded into RAM from your disk (no DOS doesn't live in your computer), you can change the contents of DOS by changing the values in the area where DOS is stored. Each location, or address, is assigned a value from 0 to 255. It is easy to PEEK, or look at, a value at any location—(example: PRINT PEEK(300) will produce a number, 0–255)—and POKE in a new one if you want—(example: POKE 300,123 will change that number to 123). *DOS BOSS* rearranges DOS according to your commands by POKEing, or inserting, new values into memory for you. Additional possibilities are endless. Here are a few catalog customizations you can make on your own. Any of them may be entered directly or made part of a program. If you want to keep one of these features, INIT with it POKed in, or add the POKes to THE *END OF* your appended POKE file.

A POEM

by Reed Righthead

An Appler we knew, just for fun,
Typed "CALL 6 4 8 7 1."
But every other time,
?Syntax Error was the crime,
Even though he had committed none.

BEWARE!!

Messing around in DOS can cause S-T-R-A-N-G-E things to happen, and before you finish this section, you (or we) may have a malfunctioning computer. Fear not! To fix things, simply turn off the power and reboot. (Remember on Apple //c's to *wait fifteen seconds* before turning on your computer again. You may end up fiddling while ROM burns.) Remember,

no permanent harm can ever be done to your Apple itself by just pressing keys. (Of course, if you press your keys with a jackhammer, you may *ruin* . . . the jackhammer.)

Beware again! There is also a slight chance that you *could* foul up a disk as well (if you typed other than the POKEs given here), so use an expendable disk (a copy) to play around with. If the disk does get messed up, you can always erase and reuse it by INITing it. Before trying each new POKE, we advise you to POKE back in the original values listed after each example.

PRINT PEEK (44611)

Your Apple should answer with a 2. Let's change it to a 1:

```
POKE 44611,1
```

Now PEEK again, and there's your 1! Next, CATALOG a disk and notice the change you have made. This POKE changes the number of digits in your catalog sector numbers to 2 instead of 3. Have you ever seen a file bigger than 99 sectors? Not very often, right? So why clutter up the catalog with extra 0's? (*Note:* If you really like clutter, POKE in a 4 or an 11!)

A possible drawback to 2-digit sector numbers: you will scramble your volume number (on the screen, not in memory). *A solution:* Omit the number with *DOS Boss*. Another drawback: If you are using *DOS Boss*'s multicolumn catalog, this POKE will really make a temporary mess of your columns! (Normal value: 2. POKE a 2 back into 44611 and continue.)

Reminder for Apple IIc'ers: Boot a DOS 3.3 disk. Give your ProDOS a vacation.)

POKE 44459,234: POKE 44460,234: POKE 44461,234

Eliminates the blank line after the word CATALOG; 234 means “do nothing,” and these POKEs do nothing instead of printing a carriage return. (Normal values: 32,47,174.)

POKE 44486,234: POKE 44487,234: POKE 44488,234

Eliminates the blank line after the Disk Volume heading. (Normal values: 32,47,174.)

POKE 44452,24: POKE 44605,23

Lets twenty file names appear before stopping for a keypress instead of the normal eighteen. POKE in any numbers. Always make the first POKE value one number larger than the second. (Normal values: 22,21.)

POKE 44541,173: POKE 44559,186

The first POKE replaces the space (value 160) after the file-type code with a hyphen. The second one puts a colon after the sector numbers. Experiment with other values from the ASCII Screen Chart in the *DOS BOSS Book Appendix*. (Normal values: 160,160.)

POKE 44567,12

Shortens your maximum file-name length to thirteen characters (on the screen, *not* in memory). Normally this number is 29, for thirty maximum characters. The number of characters is always one less than the number POKEd in. Every file name shorter than the maximum fills the remaining space with spaces. (Normal value: 29.)

POKE 44578,234: POKE 44579,234: POKE 44580,234

Cancels all carriage returns after file names. With these 234's POKEd in, play with POKEing some small numbers in at 44567 (like 2 or 12; only certain numbers will work) and you can have your own version of the multicolumn catalog. (Normal value: 32,47,174.)

POKE 44505,234: POKE 44506,234

Shows deleted files in your catalog and throws in a free bonus inverse character to the right of each. (Normal values 48,74.)

The following changes are pretty much worthless, but fun anyway:

POKE 44596,234: POKE 44597,234: POKE 44598,234

Prevents your catalog from stopping when the screen is full. (Normal values: 206,157,179.)

POKE 44599,234: POKE 44600,234

Stops your catalog at each file name, waiting for a keypress on each one. (Normal values; 208,8.)

POKE 50,128

Makes your catalog invisible (in case you're embarrassed by it). Program listings are invisible, too! (Normal value: 255.)

POKE 44617,234: POKE 44618,234: POKE 44619,234

Makes all sector numbers and your volume number appear as 000s! (Normal values: 217,164,179.)

POKE 45620,234: POKE 45621,234

Repeats your first file name forever! (Normal values: 105,35.)

FOR X = 43439 TO 43443: POKE X,7: NEXT X

This obnoxious change replaces the word FILE (+ space) with five Control-G beeps in the "FILE NOT FOUND" error message. You can

POKE beeps (7), carriage returns (13), line feeds (10), or anything into any error message and into the volume message and other unusual places as well. Each POKE, of course, replaces whatever character was there. (Normal values: 70,73,76,69,32.)

POKE 43378,42

Now you've replaced all DOS error message beeps (invisible, but audible Control-G's) with visible, but inaudible asterisks! (Normal value: 7).

MORE DOS NOTES

If you're new to the Apple, all the different DOS's are confusing. With all new Apples you get ProDOS, but your System Utilities disk can be used to format a DOS 3.3 disk. Most of the older software is on DOS 3.3 disks, which run fine on Apple //c's, //e's, and most II/II + 's.

But the Manual Said . . .

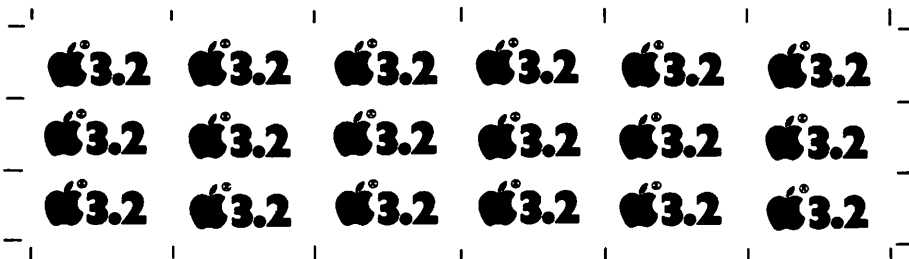
Okay, now hang on to your hats, gang. Your Apple //c manual said the Apple //c does *not* support DOS 3.2. Naturally, we had our Break The Rules Dept. see if that was true. We booted the BASICs disk and then inserted a 3.2 disk and voilà! no problem on the Apple //c. Since you didn't get a BASICs disk with your Apple //c, you're on your own on that score. Maybe Apple, Inc., will sell you one. On the other hand, if you have a DOS 3.3 System Master, BRUN a program called BOOT13, and when the prompt comes up, stick a 3.2 disk in the drive and hit RETURN. This too will get 3.2 disks to operate on your Apple //c. The same works on Apple //e's and II/II + 's.

The Manual's Way . . .

The Apple //c manual says you can change all of your DOS 3.2 files into DOS 3.3. Then you can change from 3.3 to ProDOS if you want. Use the System Utilities disk to "Change Disk Format" to do this. If you really want consistency on your Apple, change everything to ProDOS or DOS 3.3. With Apple //c's, use the System Utilities to do all of that. With Apple //e's and II/II+'s, you can change all the 3.2 disks to 3.3 with a program called MUFFIN, found on your DOS 3.3 System Master.

Stickers

Now, if you don't want to change anything, or if you have so much software it'd take you forever to make all the changes, label all of your disks with stickers designating the DOS type. This may actually be simpler, since some DOS 3.2 disks don't like to be MUFFINed into DOS 3.3, and later on you'll probably have disks with CP/M, Pascal, FORTH, and who knows what else on your disks. You can buy nice little round blank stickers from the stationery store, any color you want. Mark them with the appropriate operating-system designation *or* cut out the following and tape them on your disks:



You should do the same for your other disks with other DOS's.

3.3
ProDOS
Pascal

DE-MUFFIN

There may be situations where you will actually want to change DOS 3.3 files into DOS 3.2. Even with the Apple //c System Utilities that changes all kinds of formats, you can't do that. However, there are several de-MUFFIN (or NIFFUM) programs that have been published. Check back issues of *Nibble* (vol. 1, no. 8 and vol. 2, no. 2). One of the first things to de-MUFFIN is the 3.3 FID program (DIF?), handy to have around in any format. (*Note:* In 3.2 FID, Free Space on Disk will read high by 93 sectors.)

WHY GO BACK?

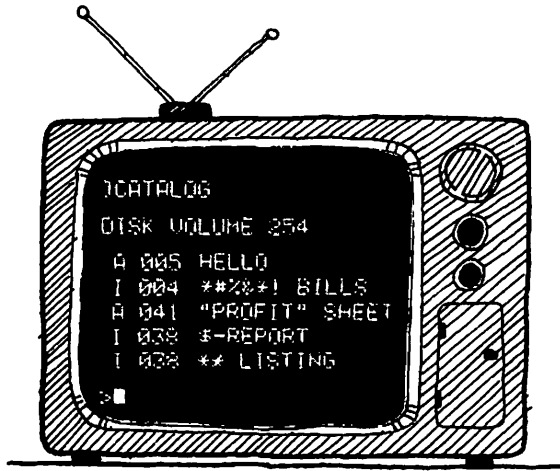
If you have a brand-new sparkling white Apple //c, you may be scratching your head wondering why use any of that "old stuff." With ProDOS you have the state-of-the-art (until a week from Tuesday, at least) DOS, and there's not a whole lot of need for old DOS 3.3 or the ancient DOS 3.2 software. You can get the NEW STUFF!!

Back in the old days when there wasn't much software for the Apple, users used to turn out software by the crateload. They'd place it in the public domain, which is another way of saying, they gave it away *free*. A lot of this early software was pure junk, but a lot of it was pretty good and some was fantastic. The various Apple user's groups collected thousands of these programs and helped give them away on club disks. The gamut runs from 3.2 to FORTH and Pascal operating systems, and there's plenty more; so if you want to get some software cheap, don't stick your nose up at non-ProDOS software.

FILE.NAMES AND DOS

If you use ProDOS, you can't have spaces between your file.names. Your Apple //c manual suggests that you name all of your files without

spaces, even on DOS 3.3 where spaces are welcomed. Then if you change the DOS 3.3 files to ProDOS, you won't get messed up. Your manual has a good idea there.



FILE-NAME VIOLATIONS

For those of you who still have old Apples with no lowercase adapters, there is *finally* a trick you can pull that is *not* possible with the newfangled Apples that have lowercase built in. You'll need Beagle Bros *Global Program Line Editor* for this one. (Why do you think we're telling you about it?) The DOS 3.2.1 manual says that file names must start with a letter, but with no lowercase adapter, you can include lowercase characters in a file name and they will appear as nonletter characters on the video screen. For example, a lowercase *D* will appear as a \$, etc. So it's easy to call a file "\$REPORT" or "*LISTING" or "%#&@BILLS." If you want to indent a file name in your catalog, a lowercase @ is a space! Here are the equivalent upper- and lower-case characters:

UC	LC	UC	LC	UC	LC
A	!	J	*	S	3
B	"	K	+	T	4
C	#	L	,	U	5
D	\$	M	-	V	6
E	%	N	.	W	7
F	&	O	/	X	8
G	'	P	0	Y	9
H	(Q	1	Z	:
I)	R	2	@	sp

By the way, if you don't have the *Global Program Line Editor* disk, go buy one now. It's worth every penny!

FLASHING AND INVERSE FILE NAMES

To get flashing and inverse file names, in the immediate mode, type <RETURN> or FLASH <RETURN>. Then PRINT "FILE NAME" <RETURN>. Type SAVE and use the cursor moves to trace over your flashing or inverse file name and <RETURN>. CATALOG your catalog, and take a look. The only way to LOAD such a program is the same way you SAVED it. Type LOAD or RUN, and trace over the file name with the cursor. It works!



MECHANICAL MEMORY MOVER

Have you ever wanted to move a chunk of memory from within a program without going into machine language or using Applesoft's clunky CALL -468? Well, here's an idea; it may be a little slow, but it works. Just BSAVE the memory chunk and then BLOAD it where you want it. For example, to store the text screen (\$400-\$7FF) at A\$6000, these lines could be in your program:

```
1000 PRINT CHR$(4); "BSAVE TEXT,A$400,  
      L$3FF"  
1010 PRINT CHR$(4); "BLOAD TEXT, A$6000"
```

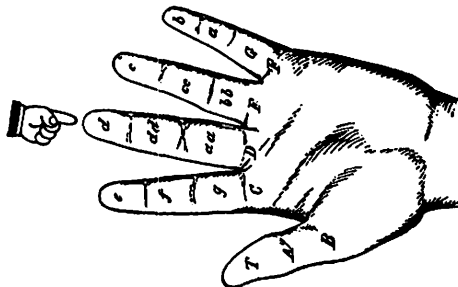
The following will clear the screen, pause, and then bring the text screen back where it belongs.

```
2000 HOME : FOR W = 1 TO 2345: NEXT  
2010 PRINT CHR$(4); "BLOAD TEXT"
```

PREBOOT NEWS

You can turn your Apple on or do a PR#6 *before* inserting a disk. It's reasonably safe to insert a disk into a drive that is running . . . we think. On Apple //c's we avoid the CHECK DISK DRIVE message by flipping the switch on with the door open and the disk in the doorway (what else would you call it?). When the drive starts spinning, we insert the disk all the way and slam the door shut . . . gently. This allows the spinning head to center the floppy disk and prevent a CHECK DISK DRIVE message.

MIDDLE FINGER SAVER



Think of the time you'll save! In spite of what the DOS 3.3 manual says, *no commas* or spaces are necessary when you do a MON I,C,O or a NOMON I,C,O. Just write MONICO or NOMONICO. Another comma recommended by the manual is the one after the CATALOG command and before the drive number. CATALOGD1 gets the job done fine!

CLOSED DEPARTMENT

You don't need to use a file name when you CLOSE a file.

DELETE HELLO

If you need more space on a disk, consider deleting your HELLO or STARTUP program to save the amount of space it occupies. You won't be able to boot the disk, but you can still use it!

CASSETTE TIP FOR DISKERS

Keep your cassette player within reach in case you lose DOS 3.3 and are unable to SAVE to disk. Just follow the instructions in your Apple

manual, SAVE to tape, reboot, and LOAD from tape. (This tip is only for old Apples and DOS 3.3.)

MULTISTATEMENTS

In Applesoft you can type multiple commands in the immediate mode separated by colons. For example:

```
HOME: INVERSE: VTAB 10: PRINT "FUB FUB"
```

It doesn't work so well with DOS commands, though under DOS 3.3, try this, then CATALOG your disk:

```
SAVE PROGRAM: DELETE PROGRAM
```

TRY THIS, TOO

```
BSAVE LONG PROGRAM,A$9000,L$7FFF
```

This BSAVE will zap your text screen into lo-res and make little "vibration" marks on some of the lo-res color plots and even click the speaker (sometimes).

A We-Hope-We-re-Not-Too-Late Warning: Don't hit Reset during disk access; you could foul up your disk. A good rule is never to hit Reset while the "In Use" light is on. Open the drive door (gasp!) first if you must.

INVERSE INIT

You can INIT a DOS 3.3 disk with an inverse greeting program name, but you can't Master Create it without changing the name to normal text.

B-READER

Some word processors, like *Bank Street Writer*, produce Binary files instead of Text files. We all know that *ProntoDOS*'s TYPE command lets you print Text files, but how do you read a Binary file? Run this DOS 3.3 program to find out.

```
10 PRINT CHR$(4);"BLOAD FILE, A9000"  
20 LENGTH=PEEK(43616)+PEEK (43617)*256  
30 FOR L=9000 TO 8999+LENGTH: PRINT  
   CHR$(PEEK(L));: NEXT
```

If you're working in ProDOS instead of DOS 3.3, substitute the numbers 48857 and 48858 in the parentheses in line 20.

CRUISING THROUGH DOS

Boot DOS 3.3 and take a cruise through DOS! The usual "trip" of this kind would be in the monitor, looking at a bunch of two-digit hex numbers—a real drag. Let's make things more interesting! First, boot a normal disk whose DOS hasn't been altered by *DOS Boss*. Now write this little program:

```
10 FOR X=43380 TO 43401  
20 PRINT PEEK(X);" ";  
30 NEXT X
```

RUN it and you will see a string of numbers. Pretty exciting, huh? Now, change line 20 to:

```
20 PRINT CHR$(PEEK(X));
```

Now RUN it again! "LANGUAGE NOT AVAILABLE" magically appears! What's this? You have just uncovered DOS's first error message! CHR\$(PEEK(X)) means the "character whose ASCII value is X." Now change line 10 to:

```
10 FOR X=43380 TO 43581
```

and you'll see *all fourteen* DOS 3.3 error messages strung together! To further examine these mysterious characters, let's add two more lines to our program:

```
15 NORMAL: IF PEEK(X)> 127 THEN INVERSE  
25 IF PEEK(X)>127 THEN PRINT
```

RUN again and you'll notice that the *last character* of every error message is inverse. Actually, line 15 tells it to be inverse *if* it has an ASCII value higher than 127. Each Apple keyboard character and control character has *two* ASCII values, sometimes called the "low-byte" value and the "high-byte" value. The high-byte character at the end of each error message tells the Apple where the end of the message is and when to quit printing letters to the screen.

Now try these immediate mode commands:

```
LOAD ZYZX
```

You get a "FILE NOT FOUND," unless you have a program named "ZZYX." Now . . .

```
POKE 43452,68  
LOAD ZYZX
```

You should get a *double* error message because you have POKEd in, or changed, the high-byte *D* in "FOUND" to a low-byte *D* (value 68 at location 43452). Now the Apple thinks that error message #5 is "FILE NOT FOUND VOLUME MISMATCH." It prints until it finds a high-byte character, the *H* in "MISMATCH" instead of the *D* in "FOUND"! Repair the damage before continuing by POKeing the high-byte value for *D* (196) back in where it belongs:

```
POKE 43452,196
```

There are other places to look, of course. If you want a really *long* trip through *all* parts of memory, change line 10 to:

```
10 FOR X=0 TO 65535
```

You will see *all kinds* of stuff—beeps, backspaces, line feeds, carriage returns, little pictures of animals (just kidding), and the big feature every so often: *real words*! If you've been RUNning some programs, you'll probably see parts of the old program listings. If you've just BRUN or BLOADed the 3.3 Master Create program, you'll even find a funny message from the author somewhere between locations 2500 and 3900, only meant to be seen by prying eyes. So go ahead and pry; SAVE your program, BLOAD MASTER CREATE. LOAD your program and RUN.

Back to DOS 3.3—change the values in line 10 to 43140 and 43338 and you'll find the twenty-eight DOS commands followed by some garbage and "VDSL RBACIO." Those are the initials for VOLUME, DRIVE, SLOT, LENGTH, etc.; the one-letter codes used in DOS! POKE in new letter values if you want.

Change line 10's numbers to 43700 and 43715. That's "APPLESOFT," the name of the program Apple tries to run if you type FP and don't have Applesoft in ROM! (Applesoft II+'s, //e's, and //c's have Applesoft in ROM and Apple II's have Integer in ROM.)

Now, change line 10 to:

```
10 FOR X=46010 TO 45991 STEP-1
```

RUN it and "DISK VOLUME BARSBAIT" appears! That's where the Apple gets the message to print at the top of your catalogs! "BARS" seem to be throwaway characters, so we've used them in *DOS BOSS* to lengthen the "Disk Volume" message. Have you guessed "BAIT" yet? How about Binary, Applesoft, Integer, and Text, the codes for your catalog files!

Look around some more; DOS goes clear up to 49151. Above 49151 is BASIC. You can look there, too, if you want! It's your computer!

25 WAYS TO BOOT A DISK

More useless Apple info from Kevin Barr:

RULES

All assume DOS 3.3 is booted (!)

"]" means typed from Applesoft

"*" means typed from the monitor (*CALL-151* to enter monitor)

```

1. Turn your Apple off and back on.
2. IPR#000000000000000006
3. IPR#$6
4. IN#6
5. *C600G
6. *6 (control-P)
7. *300: 4C 00 C6 N 300G
8. ICALL 50688
9. IPOKE 1012,0
   I(control-Reset)
10. *300: A9 C5 48 A9 FF 48 60 N 300G
11. IPRINT CHR$(4);"PR#";PEEK(57375)
12. IOREM6
   ICALL-151
   *805: 8A N D566G
13. *36: 00 C6
14. *3F9: 00 C6 N (control-Y)
15. IPOKE-23183,149: POKE-23182,254
   ICATALOG
16. *B1: 4C 00 C6
   *3D0G
17. *9E81: A9 06 4C 95 FE
18. I0 PRINT "This is useless."
   I5 FOR X=2053 TO 2060
   I10 READ P: POKE X,P: NEXT
   I5 CALL 54630
   I20 DATA 140,201,49,52,56,52,56,0
   IRUN
19. IPR#0
   I5 A$="BOOT DOS":PRINT A$
   I10 FOR X=1 TO 8:
       P=P+ASC(MID$(A$,X,1)): NEXT
   I20 A$=STR$(P): P=0: FOR X=1 TO 3:
       P=P+ASC(MID$(A$X,1)): NEXT
   I30 A$=STR$(P): A$=RIGHT$(A$,1)
   I40 POKE 2054, ASC (A$):RUN
   IRUN
20. IPOKE 1014,0: POKE 1015,198
   I&BOOTDOS
21. IPOKE 12,198: POKE 11,0
   IPOKE 10,32
   IPRINT USR(BOOTDOS)
22. *44: 6 N A229G
23. *38: 00 C6 N A851G
24. *A884: 42 4F 4F D4 N A909: 00 00 N
   BEAF: A9 06 4C 95 FE
   *BOOT

```

25. *8000: A0 00 AD EA C0 B9 29 80 F0 1C 20
F0 FD AD E9 C0 A9 FF 20 A8 FC 2C E8 C0
A2 0A A9 FF 20 A8 FC CA D0 F8 C8 4C 05
80 4C 00 C6 AD C2 CF CF D4 A0 C4 CF D3
8D 00 N 8000G

BOOT MAKER



Did you ever want to make someone boot your disk before they could RUN your program? Here is a simple way to do it:

1. Boot DOS 3.3
2. Put a blank disk in drive 1
3. Type POKE 47721,123 <RETURN>
4. Type NEW <RETURN>
5. Type INIT HELLO <RETURN>

In a minute or so, you will have a new master disk from which you will make copies. Use LOAD and SAVE (or FID) to transfer your programs to the master disk. Now put this command near the start of each of your programs:

```
1 IF PEEK (47721) <> 123 THEN PRINT  
  CHR$(4); "PR#"; PEEK (43626)
```

Location 47721 is unused by DOS 3.3. PEEK(43626) is the most recently used slot number; 123 could be any number, 0–255, as long as it matches the POKE number in step (3). Odds are 255 to 1 that your disk will be booted before your program is RUN.

MASTER CREATE

Question Number One

The most-asked question over the Beagle Bros phone is “How can I make Key-Cat run automatically when I boot my disk?” (*Key-Cat* is on the *DOS Boss* disk.) The answer is easy. Just BRUN MASTER CREATE. It will ask you what you want to call your greeting program. Enter *Key-Cat* or whatever you want to call it, and that’s it! Now Key-Cat will run when you *boot*! Another way to make it run would be to put a PRINT CHR\$(4);“RUN KEY-CAT” as the last statement in your greeting program.

Disk Fixer

If you’ve got a disk that won’t boot or is giving you a bad time in some other way, there’s just a *chance* that you can repair it with the Master Create program on your DOS 3.3 System Master disk. Just BRUN MASTER CREATE and insert the disk you want to fix. It will rewrite tracks 0–2 (DOS) for you, and maybe that’s where your problem was. While you’re at it, Master Create will let you change the name of your disk’s greeting program, too.

WHY BUY MORE THAN ONE DISK DRIVE?

1. You can make faster disk copies.
2. You can use certain software that requires two drives.
3. You will have a backup drive when one goes bad (it happens).
4. You can have immediate access to twice the data.
5. You can get rid of that extra \$300 in the dresser drawer.
6. If you have an Apple Macintosh and were wondering when this book is going to help you, you have just found out.

WARNING DEPARTMENT

Rumor has it that you should always *open your drive doors* and/or remove all disks from your drives *before* turning off your Apple. You risk disk damage if you don't. And while we're issuing warnings—if you leave your Apple on and unattended for a long time *and* a temporary power failure occurs, the Autostart ROM will attempt to reboot. *If* your drive door is open or no disk is in your drive, your drive could run indefinitely (or until the next power failure, whichever comes first).

DISK ZAP?

Just how fragile are diskettes? Why not find out? Take a disk that is expendable and place it an inch or two from a magnet. Will it still boot? Lean it against your color TV screen. Now call the dog over to the Apple. . . .

Disks can be zapped in many ways but are surprisingly tough when you consider what they consist of. We couldn't resist cutting one open. Can you?

DOS TRICKERY

Here's a DOS change you can do without *DOS Boss*. If you PEEK location 43698 (\$AAB2), you'll find a 132 (\$84). That is the high-byte value of Control-D, DOS's code character for use in PRINT statements. Now POKE in a 192 (POKE 43698,192), the *high-byte value for a @*. You have now changed the code character from Control-D to @. Now to execute a DOS command like CATALOG from a program, you can type PRINT "@CATALOG."

This is just one more way to confuse people and customize DOS to your needs—and it's kind of nice to be able to *see* the code character!

BUT HOW DO YOU READ THE LABEL?

You will probably ignore this hint; we do. If you are going to leave disks all over your desk, you should place the label-side down so that they aren't damaged by dirt, etc., on your desk. The disk drive head reads the *bottom* of your disk through the oval hole. The hole on top is for the pressure pad.

IN USE?



We just decided last Tuesday that *it is okay* to type keyboard commands while your drive's "In Use" light is lit as long as you have a prompt and flashing cursor. The Apple drives run for about one second after they are finished doing whatever it is they do in there. No one anywhere knows why.

RIGHT PROTECTORS

Most of the write-protect tabs we have used come off or get messed up going in and out of our disk drives. A handy new product called Scotch tape works much better! Some computers rely on a beam of light to check for write protection, so their tabs would have to be opaque to work. Test your drives to see.

TWO-SIDED DISKS: PART I

Sure you can use both sides of your “single-sided” disks. This can save you both money and storage space. Use a regular paper punch to make a half-circle notch on the edge of a disk jacket *exactly opposite* the original write-protect notch. Then INIT the second side just like you did the first. Disk manufacturers don’t usually test both sides of disks, so you do run the risk of a bad sector now and then and you *could* lose some data. If you make backup copies anyhow, this shouldn’t be a problem. By the way, the small off-center hole in the disk jacket isn’t used by Apple’s DOS.

TWO-SIDED DISKS: PART II

We have been known to advocate your using both sides of a disk. Here are two reasons not to—

1. A stack of disks is harder to search through with stuff on both sides. *Solution:* Label both sides’ contents on one side of the disk.

2. When you use the “wrong” side of a disk, it spins the “wrong” way and can mess up the friction pad (or whatever it’s called) inside the disk itself. This is potentially a b-i-g problem, and our good friend, Pete the Pessimist, says you can mess up not only the disk, but your *drive* as well! Pete also refuses to go outside of his closet without a hard hat.

TWO-SIDED DISKS: PART III

If you decide to ignore the preceding warnings, here’s a tip we’ve seen widely used—put ProDOS and 3.3 versions of your favorite utilities or whatever on opposite sides of a disk.

AHA! A USE FOR VERIFY

When you want to READ a text file, you first have to OPEN the file. If the file doesn’t exist, you’ll get an “END OF DATA” message and you’ll acquire a new one-sector file name in your catalog, even if you used ONERR to trap the error. Phooey! To prevent this, VERIFY the file first, *then* OPEN it. If VERIFY doesn’t find the file, ONERR catches your mistake, and life goes on with no file added to your catalog.

D\$ FIX

This program *won’t* catalog a disk:

```
10 D$;CHR$(4): REM CTRL-D
20 PRINT "WATCH...";
30 PRINT D$;"CATALOG"
```

The problem is that D\$ (Control-D) in line 30 is at HTAB 9 because of the semicolon in line 20. One remedy that always gets your D\$ to HTAB 1 where it will function is:

```
10 D$=CHR$(13)+CHR$(4): REM CARRIAGE  
   RETURN + CTRL-D
```

Using that combination you can do things like the following:

```
20 PRINT D$"OPEN FILE": PRINT D$"READ FILE"
```

However, when you CLOSE a file, use the old CHR\$(4) without the carriage return. You know why? If you are WRITEing to a file, and the CHR\$(13) + CHR\$(4) D\$ is being used, you will get the added carriage return, CHR\$(13), written to your disk. Maybe you should have:

```
10 D$ = CHR$(13) + CHR$(4) : DC$ + CHR$(4)
```

The C in DC\$ is to remind you to use it when you CLOSE a file. Also, while we're on the subject . . . *do not* do the following:

```
10 D$ = CHR$(4) + CHR$(13) : REM THE  
   CHR$(4) IS FIRST - WRONG
```

If you place the CHR\$(4) before the CHR\$(13), it won't work. Remember, place the CHR\$(13) first; then the CHR\$(4).

SEMISMART RUN COMMAND

ProDOS's hyphen command is a handy animal. The command -FILENAME will automatically RUN an Applesoft file, BRUN a binary file, or EXEC a text file. What it won't do is BLOAD a picture file or RESTORE a variable file. Hey, no command is perfect.

?ILLEGAL MESSAGE

If you type `RENAME "OLD.FILENAME,ANOTHER.FILENAME"` using ProDOS, you'll get a "Syntax Error" message. The problem is the new file name is longer than fifteen characters. Come on, ProDOS—why not just truncate the name at fifteen like DOS 3.3 does at thirty?

BOOT DRIVE 2!

On an Apple //c, you can boot ProDOS from a slot-6, drive-2 disk by typing `:PR#7`. The leading colon makes the //c take this as an Applesoft command, not a DOS command. If you try to boot DOS 3.3 from drive 2, it won't work, but the results are at least interesting.

DOS CONNECTOR

Certain circumstances will disconnect ProDOS. If commands like `CAT` or `PREFIX` suddenly don't function, try typing `CALL 976`. If ProDOS commands still don't work, you'll probably need to reboot. Don't use this `CALL` in a program unless you want it to end at that point.

One way to disconnect ProDOS intentionally on a //c or //e with an 80-column card is to type `:PR#3` (notice the colon). After doing so, try `RUNning` a program. You'll see line numbers all over the screen, because ProDOS has left you with the `TRACE` function in gear.

NO MONSTERS ALLOWED

Did you know that, with ProDOS booted, you can type a title of this paragraph and not get a "?Syntax Error"? Wow—tell your friends!

/THE/SHOR/TER/THE/BET/TER

ProDOS allows up to fifteen characters for disk (volume), subdirectory, and file names. If you're into actually using subdirectories, you'll be a happier, less-frustrated person if you keep those names short. Long path names are hard to remember, difficult to type, and generally a pain.

PREFIX FIXES

To see what the current ProDOS prefix is, type PREFIX. To set the prefix to the disk in drive 2, PREFIX,D2. To set the prefix to a subdirectory named "SUB," type PREFIX SUB. To cancel all previous prefixes, type PREFIX/.

PRODOS DISQUIK

If you have an Apple //c or a 128K //e, you are equipped with a built-in "RAM disk" in auxiliary memory. To prove that it's there, type CAT,S3,D2. You should see an empty ProDOS catalog named "/RAM," ready to be filled with about 119 blocks' worth of files. (If you only have a 64K Apple, you'll get a "No Device Connected" message.)

To demonstrate the speed of /RAM, LOAD a large file from disk. Now set the prefix by typing PREFIX/RAM or PREFIX,S3,D2. Now type SAVE FILE and then CAT. You should see "FILE" in the /RAM directory. Now type NEW and RUN FILE. Pretty fast, huh?

You can save three different hi-res pictures into /RAM and call them to the screen at lightning-fast speeds. Try this:

```
10 HCOLOR= 3:D$ = CHR$ (4): PRINT  
   D$"PREFIX/RAM"
```

```
30 HGR2 : FOR X = 1 TO 6: HGR2 : FOR Y = X
  * 9 TO X * 9 + 99: HPL0T X * 19,Y TO X *
  19 + 149,Y: NEXT : PRINT D$"BSAVE PIC.
  "X",A$4000,L$1FFF": NEXT
50 RESTORE : FOR X = 1 TO 10: READ P: PRINT
  D$"BLOAD PIC."P",A$4000": NEXT: DATA 1,
  2,3,4,5,6,5,4,3,2: GOTO 50
```

Beagle Bros' *Extra K* disk has a program called "Extra.Screens" that lets you store up to seven hi-res screens in auxiliary memory. It also lets you crop pictures so you can store even more. The possibilities are endless. Check our ads. If you're intrigued by /RAM, you'll really like *Extra K*.

ILLEGAL BLOCK COUNT

If you see a two-block ProDOS file in an unaltered ProDOS catalog, have someone examine your opticals. ProDOS files are always one block or three or more blocks in size. Exception: DIR files.

DISK HANGER

Someone wrote and told us you can punch a hole through the corner of an often-used disk, like a ProDOS boot-up disk, and hang it within reach of your drive. There's a disk you'll never lose!

This is definitely not the worst idea we've ever heard. That one was from the lady in Tulsa who attached her floppies to the side of her monitor with refrigerator magnets.

BLOCK/SECTOR CONVERTER

The strange relationship between ProDOS blocks and DOS 3.3 sectors can be charted by RUNning this program:

```

10  PRINT CHR$(4)"PR#1": REM SET YOUR PRINT
    FOR CONDENSED TYPE
100  DIM S(15),AB(15): FOR I = 0 TO 15: READ
    S(I): NEXT
120  FOR I = 0 TO 15: READ AB(I): NEXT
150  DATA 0,7,6,6,5,5,4,4,3,3,2,2,1,1,0,7
155  DATA 0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,1
310  PRINT SPC(30);"SECTORS": PRINT "TRACK
    ";SPC(2);: FOR I = 0 TO 15: PRINT I;
    SPC(3 + (I < 10));: NEXT
320  FOR TR = 0 TO 34: PRINT : PRINT SPC( TR
    < 10);TR;" - ";
350  FOR SE = 0 TO 15: BL = 8 * TR + S(SE):PT
    = AB(SE)
360  PRINT SPC(1 + (BL < 100) + (BL < 10));
    BL; CHR$(65 + (PT = 1));
370  NEXT : NEXT : PRINT : PRINT CHR$(4)
    "PR#0"

```

PR# YOU-NAME-IT

ProDOS lets any number 1-7 access any slot 1-7. Think hard and maybe you'll come up with a reason for doing this.

```

10  SLOT = 1: REM THE SLOT YOU WANT TO
    ACCESS
20  NUM = 5: REM THE NUMBER YOU WANT TO USE
30  LOC = 48656 + NUM + NUM
40  POKE LOC,0: POKE LOC + 1,SLOT + 192
50  PRINT "TYPE PR#";NUM;" TO ACCESS SLOT
    ";SLOT;"."

```

This program should make PR#5 turn on your printer in slot #1. If it doesn't, phone Steve or Woz.

CHAPTER 15

CATALOG

CAT OR CATALOG WITH PRODOS?

Just in case you didn't read your ProDOS manual or your Apple //c manual or missed it . . . there's a difference between what you get with CAT and CATALOG with ProDOS:

CAT—40-column catalog
CATALOG—80-column catalog

It's simple to remember which does which. CAT is *shorter* just like 40 columns.

DOS 3.3 HOME CAT

From Leif Rudd of Lincoln, Nebraska: POKE 44457,88, POKE 44458,252 will clear the screen before every DOS 3.3 Catalog.

BUZZER



DOS 3.3 FILE-NAME TRICKS

You can use the REM tricks we discussed in the “List Tips” chapter to catalog file names, too. By employing DOS’s RENAME function and a little ingenuity, file-name characters can “back up” over their sector numbers, or have carriage returns or Control-J’s inserted into them.

```
*A 012 FILE #1
FILE #2 ← This file name is actually "A[REDACTED]FILE #2".
            Type control-W and then
            "RENAME FILE #3, A[REDACTED]FILE #2"
*A 027 FILE #3
*A 062 FILE #4
```

Without a utility like *BYTE ZAP* (from Beagle Bros), you can’t have control characters and inverse characters in the same file name.

PEEKING AT THE LAST FILE IN DOS 3.3.

(Don’t try this with ProDOS.) The following one liner will print the name of the last file accessed by DOS, as well as its starting address and

length (valid only if it was a binary file). Problem is, you can't RUN this program, you can only GOTO it (type GOTO 10) or use it within a program.

```
10 PRINT "LAST FILE: ";: FOR C = 43637 TO
43667: PRINT CHR$(PEEK(C));: NEXT :
PRINT "A";PEEK (43634) + 256 * PEEK
(43635);",L"; PEEK(43616) + 256 * PEEK
(43617)
```

If the last disk command did *not* contain a file name ("CATALOG" for example), then the first characters of the file name will be erased.

The handiest way to use this program is as a *GPLE* Escape function. Then no GOTO is necessary! You could replace Escape W with an abbreviated form of line 10.

Oops, almost forgot. If you want it, here's a machine-language version of the same routine, without the address and length:



02DD-	A0	00	LDY	#\$00
02DF-	B9	75 AA	LDA	\$\$A75,Y
02E2-	20	ED FD	JSR	\$FDED
02E5-	C8		INY	
02E6-	C0	1E	CPY	#\$1E
02E8-	90	F5	BCC	\$02DF
02EA-	A9	8D	LDA	#\$8D
02EC-	20	ED FD	JSR	\$FDED
02EF-	60		RTS	

After entering this code, type CALL 733 to execute it. In case you're new at machine language, to enter this program, type:

```
CALL -151 <RETURN>
2DD: A0 00 B9 75 AA 20 ED FD C8 C0 1E 90 F5
      A9 8D 20 ED FD 60 <RETURN>
Control-C <RETURN>
CALL 733 <RETURN> to execute
```

INVISI-CALC?

As mentioned elsewhere in this book (we forgot where), POKE 50,128 will make a listing or catalog (except for inverse file names) invisible in DOS 3.3.

This program reveals all the POKE 50 possibilities—anyone for secret codes?

```
10 FOR X= 0 TO 255
20 POKE 50,255: PRINT "POKE 50,";X;": ";
30 POKE 50,X: PRINT "TESTING 1,2,3,4,5..."
40 NEXT X: POKE 50,255: END
```

DOS 3.3 FILE ARRANGER

If you are creating a DOS 3.3 disk where you want files in a certain order in your catalog, you normally have to SAVE files in the order you want them. Not so with Beagle Bros' File Arranger! First, INIT a new disk. Then type and RUN this:

```
10 D$ = CHR$(4)
20 FOR FILE = 1 TO 15
30 PRINT D$;"SAVE A";FILE
40 NEXT FILE
50 END
```


Now your catalog shows files named A1 through A15 (or as many as you want). To place a file at any position, say position A7, just LOAD PROGRAM from another disk, DELETE A7 from the new disk, and SAVE PROGRAM. It will appear in your catalog at position A7! Enter all of your other files where you want in the same manner. When you are finished, you can simply *delete* all of the A-numbers that are remaining, or RENAME THEM “^” or “^^” as separators.

CONTROL-FIND

Key in the following program:

```
10 DATA 201,141,240,21
20 DATA 234,234,234,234
30 DATA 201,128,144,13,201,160,176,9,72,
   132,53,56,233
40 DATA 128: REM 64= FLASH, 0=NORMAL
50 DATA 76,249,253,76,240,253
60 FOR X = 768 TO 795: REM $300 TO $31B
70 READ N: POKE X,N: NEXT
80 POKE 54,0: POKE 55,3
90 CALL 1002: REM RESET OR PR#0 KILLS THIS
   PROGRAM
```

RUN it (nothing happens). Now, when you start keying in control characters, they will show up on the screen. If you (or a sneaky friend) put a control character in a file name, when you CATALOG the disk, you will see the control characters displayed in inverse video.

EASY RUNNER

To RUN, BRUN, LOAD, BLOAD, EXEC, SAVE, BSAVE, DELETE, VERIFY LOCK, or UNLOCK (whew!) a program from the catalog with-

out having to type the program's name, simply move your cursor up the left margin (with Escape D or Escape I or Escape-up arrow), type RUN (or whatever), *trace over* the file name with the right arrow, and hit RETURN. Guaranteed no spelling errors. (Be sure to knock out the sector size numbers with the space bar.)

```

DISK VOLUME 123
*RUN OLD RECIPES → (RETURN)
*I 075 STAR BORES
*A 033 DATA BASE
*I 011 NAIL FILE
*I 041 KLUGE FILE
*A 024 NOISES
*A 041 ETC.
J

```

NOTRACE CATALOG

The preceding tip can be messed up good if you RUN the following program first.

```

10 PRINT CHR$(4); "CATALOG"
20 FOR I=1 TO 47 STEP 2
30 S = SCRN( 7,I): IF S = 12 OR S = 13 THEN
   COLOR = S - 4: PLOT 7,I
40 NEXT

```

Line 30 changes the first character of each file name to a *control* character (on the screen only). And even though control characters can sometimes be seen, they can't be traced!

DOS 3.3 255-SECTOR HANGMAN?

You can purposely or accidentally have a file take up more space on a disk than it really occupies in memory. To prove it, SAVE LARGE PROGRAM

(say 50 sectors). LOAD TEENY PROGRAM (say 2 sectors). SAVE LARGE PROGRAM. And finally, RENAME LARGE PROGRAM, TEENY PROGRAM. Now TEENY PROGRAM shows 50 sectors in the catalog! If you try the same thing on ProDOS, it won't work. ProDOS is too smart for that.

PRODOS VS. DOS 3.3 SECTOR SIZE

If you write the smallest Applesoft program you can in DOS 3.3, it will be saved as a 2-sector file. In ProDOS it will be saved as a 1-block file. However, the smallest text file saved under DOS 3.3 will be saved as a 1-sector file. Also, ProDOS saves Applesoft files as BAS files instead of A files. You know why? ProDOS does not work with Integer BASIC, and so it doesn't do any good to distinguish between "I" and "A" files. There will never ever be any "I" files on ProDOS. DOS 3.3 treats its old friends better.

LIST CAT

Add this line to the beginning of any DOS 3.3 Applesoft program:

```
10 IF N> 99 THEN N=1: PRINT "CATALOG"
```

(There is a Control-D hidden between the first quote mark and the C of CATALOG.)

Now try to LIST. You can't. Notice you are presented with a "SYNTAX ERROR" Applesoft can't seem to stand having a Control-D at HTAB 1 without trying to *execute* the commands following it! To make it execute, *remove the last quote mark* (completely legal in Applesoft). Now the LIST command will *catalog*!! The IF N>99, etc., is just filler to get the Control-D on the left of the second line.

Creative (or Destructive) Possibilities: Change CATALOG to FP, and a LIST command will murder the program! Serves that guy right for trying to LIST your prized SUPERDATABASE program, right? Of course, *no one* would be rotten enough to change the command to DELETE SUPERDATABASE. (or INIT SUPERDATABASE? Never!!!)

TITLE TIP

You can start a file name with any character whose ASCII number is above 63. The DOS manual says you have to start with a letter but forgets to mention the at sign, underscore, exponent sign, square brackets, backslash, and the lowercase equivalents of these characters.

@DON'T BELIEVE [IT.

According to the DOS Three-Point-Whatever Manual a file name must start with a *letter*. Not so! Shifted letters (*N*, *P*, and *M*) work too, as well as \ and [. Nice for differentiating types of files in your catalogs.

BELIEVE IT

None of the last file-name tricks work on ProDOS. ProDOS takes all the fun out of goofing up your files. Use DOS 3.3 for a good time.

FILE NAMES AS TITLES ON DOS 3.3

You have probably noticed the catalog titles in our multigame disks, where we separate Applesoft games from Integer:

CATALOG

DISK VOLUME 123

INTEGER

*I 053 TEXTTRAIN

*I 036 SUB SEARCH

*I 033 PICK-A-PAIR

APPLESOFT

*A 053 TEXTTRAIN/A

*A 036 SUB SEARCH/A

*A 033 PICK-A-PAIR/A

The flush-left, underlined INTEGER and APPLESOFT headings help organize the catalog and separate one group of files from the other. Here's how we do it!

```
10 D$=CHR$(4) : H$=CHR$(8)
20 H$= H$ + H$ + H$ + H$ + H$ + H$ + H$ +
   H$
30 FILE$ = "X" + H$ + "APPLESOFT"
40 PRINT D$;"SAVE ";FILE$
```

Substitute the word you want for your titles in line 30. You can also replace SAVE in line 40 with DELETE, LOAD, or whatever. Accessing these "titles" is difficult without a program similar to the preceding one.

LEGAL NAMES DEPARTMENT

Variables must start with an uppercase alphabetical character (A-Z). (Pop your Apple //e or //c into lowercase and enter a lowercase variable name and LIST the program. All the variable names are now uppercase.) The second character may be A-Z or a digit 0-9 (or nothing). Characters after that don't count, but they have to be letters or numbers.

File names under DOS 3.3, however, may start with your choice of sixty-four normal or control characters (not to mention inverse and flashing), but *not* characters with ASCII values 32–63. File names may not contain commas.

ILLEGAL MOVES IN PRODOS

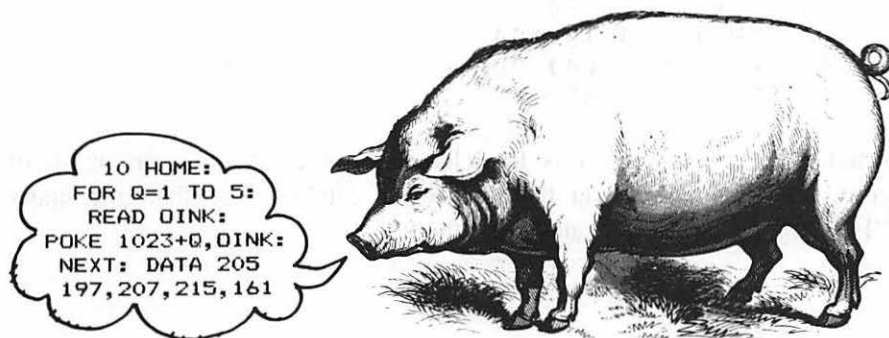
Since you can't have all the fun with ProDOS that you can with DOS 3.3, try this:

1. Put some file names with control characters and other "illegal" characters in some DOS 3.3 files.
2. Using the System Utilities disk from ProDOS, transfer/change the DOS 3.3 files to ProDOS.
3. Forget who gave you this tip.

SPACE CAT

Hey everybody! *Double space your catalogs* with a POKE 33,37! Mail your reasons for doing this along with \$1 to:

Bert & Bill
Beagle Bros World Headquarters
3990 Old Town Avenue
San Diego, CA 92110



DOS 3.2 AND 3.3 FAT CATALOG

Here's a little experiment that might teach you something about catalog formatting in DOS 3.2 and 3.3—INIT a new disk, DELETE your greeting program, and RUN this:

```
10 PRINT CHR$(4);"MONICO"  
20 FOR X = 0 TO 150  
30 PRINT CHR$(4);"SAVE FILE #";X  
40 NEXT X
```

This will take a while, so go make a sandwich or take a shower (you *do* need one). Sooner or later, you will get a "DISK FULL" error message (around the 105th file name; less in 3.2) *not* because the disk is full (you've only occupied about 210 sectors), but because there is no more room for file names on the disk.

CATALOG your new disk and you will see dummy file names FILE #0, FILE #1, FILE #2, etc. Up to FILE #104. Now try LOADING a program that doesn't exist (LOAD MUSH, for instance). Instead of the expected "FILE NOT FOUND" message, you will get a "DISK FULL" message—a protest by DOS that it is stuffed and can't think straight.

You can use this disk to arrange files as we showed in our "File Arranger" tip a few pages back. Once you have all the files you want on the disk, here's how to get rid of the dummy files.

```
10 PRINT CHR$(4);"MONICO"  
20 ONERR GOTO 50  
30 FOR X = 0 TO 104  
40 PRINT CHR$(4);"DELETE FILE #";X  
50 NEXT X : END
```

Another interesting effect is to delete all but every fifteenth or so of your 105 files. Now you have a slower catalog! File that one under "How'd You Do That?" and show the gang.

NO CAT

POKE -21503,0 prevents a DOS 3.3 catalog (17 is the normal value at this address).

CONT CAT

CATALOG a disk with more than twenty file names. The catalog stops when the screen is full, right? Well, *switch disks during the pause*, and press any key to continue. What happens? Let us know; we were afraid to try it.

SAME NAME

By using DOS's RENAME function, two or more files can have the same name. Now only the top same-name file will be recognized until you RENAME it! We like to use the unimaginative but easy-to-spell title, "^", several places in our catalogs as separators.

3.3 FORMAT PRO-CAT

To make the ProDOS CAT command produce a DOS 3.3-looking catalog, type four POKEs—POKE 42200,10: POKE 42238,2: POKE 42324,7: POKE 42348,96.

One of the big benefits of this new format is that you can Escape the cursor up the left side of the screen, type a command like RENAME, and then cursor-trace over the file name, just like in 3.3. Normally, there's only room to type one character (like a hyphen). POKE 42200 sets the file-name column, 42238 sets the file-type column, and 42324 sets the blocks column. The 96 at 42348 makes the Modified Date not print.

CHAPTER 16

APPLE II, APPLE II+, APPLE //e, AND APPLE //c

APPLE I?

Of course there was an Apple I. Back in the old days when computers cost megabucks and only corporations and the government could afford them, the Apple I was one of the first “affordable” computers for the home. It was around 10K (that is, \$10,000). Of course, it only came with about 4K of RAM, and you had to supply your own keyboard and there were no disk drives; only cassettes. Now that was *fun*!

APPLE II

After the Apple I, the Apple II arrived on the scene. It had its own case and keyboard. The early ones came with only 16K of RAM and additional 4K of RAM cost \$100. It had Integer BASIC in ROM and most people were still using cassette tapes until DOS 3.1 and the disk drives arrived. They did not have the Autostart ROM, and upon

turning on your machine you had to boot out of the monitor. However, the Integer ROM also had the mini-assembler and monitor built in. The eight slots in the back of the machine made it expandable beyond early imaginations.

APPLE II+

This was essentially the same as the Apple II except they were more likely to come with 48K. The big difference was the BASIC ROM that had Applesoft BASIC instead of Integer BASIC and the Autostart ROM. Alas, they took the mini-assembler out of the ROM, and you had to get a 16K card or Integer card to get Integer and the mini-assembler. The early Apple II+ owners bought a lot of Integer cards and 16K cards to get Integer, since so much Apple software was written in that version of Apple BASIC. Neither the Apple II nor II+ had lowercase adapters, and so you had to buy a special lowercase chip to be installed behind the keyboard. Since it only had 40 columns, 80-column cards were a big item, and at \$300 a pop, a lot of people learned to live with 40.

APPLE //e

This was the first big change that came in the Apple II. They were loaded with 64K RAM chips, had a new keyboard, a special slot for an 80-column card, built-in lowercase, auto-repeat keys, and could be expanded (easily) up to 128K. There were a lot fewer chips due to improved technology, and while the case and overall computer looked pretty much like the old Apple II/II+, the insides were significantly different. Even though a lot of people don't know (or care), it is possible to replace the 6502 microprocessor chip with the newer 65C02 chip that comes installed in the //c. This will allow //e owners to run software that takes advantage of the new opcodes in the 65C02.

APPLE //c

This version of the Apple II really looks different from the others. However, the main differences are inside. First, the Apple //c comes with a built-in disk drive, the 65C02 microprocessor, and it has *no* slots. It has a built-in RF modulator, serial ports (no parallel ports!), 40/80 columns, switchable keyboards (QWERTY and Dvorak), and a mouse/joystick/paddles port. It comes with 128K instead of 64K, and you get ProDOS instead of DOS 3.3 in the System Utilities disk that comes with the computer. However, it works with DOS 3.3 and even DOS 3.2, which the manual says it does not support. (What do they know?) The power transformer has been taken out of the computer and put on the cord to keep it cooler inside.

PROGRESS AND REGRESS

The early Apples were built for what have become known as “hackers.” The hackers were undisciplined, self-taught programmers who would write programs by “hacking” their way through code. They were tinkers, gadget freaks, and all-around fun-loving weirdos. Then came programs like *Visi-Calc*, and the Apple became a practical machine for business users, as programs became “friendlier” (“user friendly” has the same connotations as “idiotproof”). Apple, Inc. began attending more to those people who couldn’t care less about learning how to program and wanted to do practical things like word processing, spreadsheet analysis, and database management. The mini-assembler in ROM was removed, but since Applesoft was so much more powerful and user friendly (there’s that term again), there weren’t too many complaints. With the Apple //c the slots were taken away so that the hackers were unable to easily hang everything from a Z-80 card to a speech synthesizer onto their machines. However, the Apple is still one of the few microcomputers with a built-in monitor, and with all of the built-in goodies on the //c, a lot of the peripherals that had to be purchased separately now come standard.

CONVERT PROBLEMS?

If you're having problems with Apple's CONVERT program (it converts 3.3 programs to ProDOS), you're not alone. We'd go over some of the problems here but, in a rage, someone destroyed our only copy of CONVERT.

ENHANCED //e?

You'll often see references to "enhanced" //e's or the "new //e ROM." If you're not sure which model you have, the boot-up message at the top of the screen helps date your Apple. Pre-//e models reported "APPLE][." "Apple][" means an older model //e. "Apple II" means newer. The Apple //c boots up saying "Apple //c" but the following program shows that the old "Apple II" message is still hidden in ROM (but why?).

```
10 REM APPLE //c ONLY
20 ST=64265: EN=ST+7: GOSUB 50
30 ST=64771: EN=ST+8: GOSUB 50: END
50 FOR I=ST TO EN: PRINT CHR$(PEEK(I));
60 NEXT: PRINT: RETURN
```

Hmmm. Wonder if anyone at Apple ever considered making the Apple II+ boot up saying "Apple II+" and the //e boot up saying "Apple //e"—nah, too simple!

//c POWER OFF?

Did it ever occur to you that turning a //c off using the built-in switch doesn't turn off the power supply, which is halfway down the cord . . . or does it?

NO-SCROLL RESET

The //c and enhanced //e text display doesn't scroll up a line when you hit Control-Reset. Nice feature; too bad *GPLe* can't support the improvement.

WHICH APPLE'S WHICH?

Here we go again. . . .

```
10 A=PEEK(64435) : B=PEEK(64448)
20 IF A=6 AND B=0 THEN PRINT "APPLE //c"
30 IF A=6 AND B=234 THEN PRINT "APPLE //e"
40 IF A=6 AND B=224 THEN PRINT "Enhanced
   APPLE //e"
50 IF A<>6 THEN PRINT "APPLE II OR II+"
```

CHAPTER 17

MAINTENANCE

COLOR = ?

If you're not getting the lo-res colors the Apple manual says you should (for example, color #8 should be brown, not red), try flipping your TV's "Automatic Color" switch *off*, and play with your contrast and brightness controls. When you have your colors where you want them, mark your controls for future reference. Another set of marks for your regular TV picture is handy, too.

SQUISHY INVERSE?

If your inverse characters are bleeding together and difficult to read, try turning the little black knurled screw near the paddle socket until you see what you want.

CLEAN YOUR HEAD!

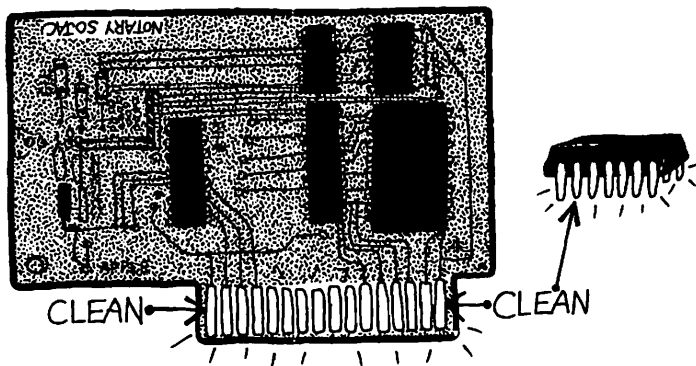
If you buy the disk drive head-cleaning kit that we bought, you'll notice that they forgot to tell you *where* the head is (top or bottom) on the Apple. It's on the *bottom*. Don't get cleaning fluid on your pressure pad; it's on the *top*.

By the way, if you're not sure how to get the head spinning, try the following:

POKE -16151,0

To stop the spinning:

POKE -16152,0



AND FLOSS YOUR DOS ON YOUR APPLE II/II+ AND //e!

Certain unexplainable, unpredictable, and completely maddening memory losses and other errors can be caused by an accumulation of Gunk (technically, "Crud") on the little feet of your ROM chips and the metal

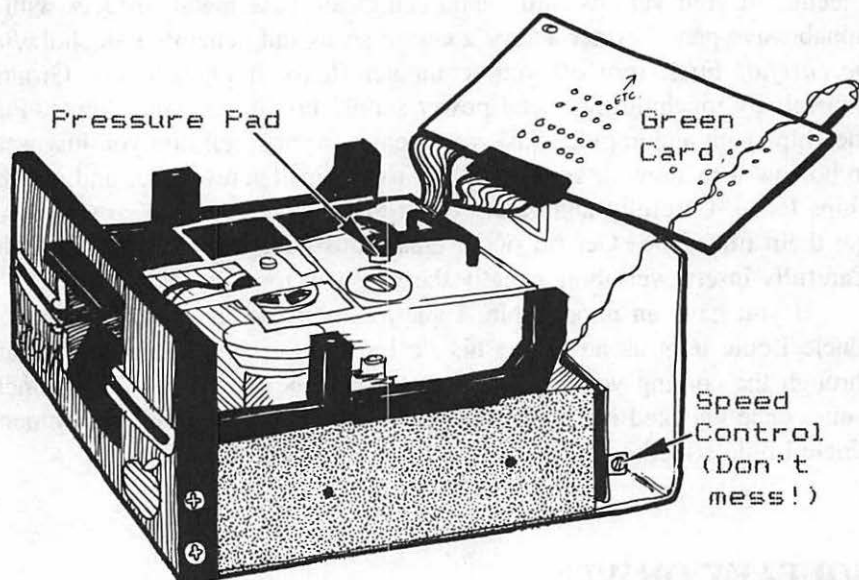
“teeth” of your various cards. You can clean these metal surfaces with a nonabrasive pencil eraser and/or a cotton swab and denatured alcohol. *But be careful!* First, turn off your computer (leave it plugged in). Ground yourself by touching the metal power supply box inside your Apple. Pull the chips with a chip puller (ask your dealer for one; tell him you just want to borrow it to show to your wife). Or use a small screwdriver and pry the chips loose. Carefully and *gently* clean things one at a time so you don’t get them mixed up. Get rid of all eraser dust, fingerprints, and cat hair. Carefully insert everything exactly the way you found it.

If you have an unopenable Apple //c, *none of the preceding* works. Uncle Louie tells us he cleans his //c by giving it a whiff of his breath through the cooling vents. This tends to melt the plastic case, and Uncle Louie once knocked out a Breathalyzer machine; so we don’t recommend Uncle Louie’s method. The dishwasher didn’t work either.

SQUEAKY DRIVE?

If your Disk II drive is driving you up the wall with a squee-eee-eak as it does its thing, it probably has a worn pressure pad. If you’re good at taking things apart (aren’t we all?) *and* getting them back together properly, take the four Phillips screws out of the bottom of your drive and slide off the metal case. Remove the two Phillips screws that hold down the horizontal green board inside. Carefully unmount the green board (watch what you’re doing; you have to put it back!). It will remain connected to the drive by wires. Now, insert a disk in the drive and close the drive door. You will see the pressure pad (a little cottony thing about 1/4” in diameter mounted on the end of a black arm) on top of the oval slot of the disk. Turn the pressure pad’s black screw about a quarter turn. This should rotate it just enough to stop the squeaking. Or you can fluff the pad up with a screwdriver or something.

Warning: All of this will undoubtedly void your warranty if anyone finds out what you’ve done . . . but it’s your equipment, right?



If you mess up your drive, forget where you read this.

While you've got the cover off, watch your drive work; it's fun! INIT a disk, DELETE some files, CATALOG, etc., and watch the pressure pad move. The read/write head is directly under the pressure pad contacting the bottom of your disk. Disk drives are indeed amazing.

Warning: Never pour cream soda in your disk drive.

Cassette Users Tip-of-the-Year (wherever you are): Buy a disk drive.

THE LIGHT TIME!

If you want to set the speed of your drive (and you still have it open), turn it upside down and you'll see a little ring with lines on it. When your drive is at the proper speed and spinning, under a fluorescent light, the lines on the ring will appear to be still. (They'll be spinning, but they

won't appear to be. Some stereo turntables use the same method to set their speeds.) Using a small screwdriver, adjust the Speed Control.

CHECK DISK DRIVE

The Apple //c main drive will only run for a few seconds if you try to boot with the door open or with no disk in the drive (it could happen because of a power failure). On other Apples it's a good idea to keep a disk installed when you're away. Otherwise, you may come home to a very hot disk drive.

CHAPTER 18

INSIGNIFICA

DON'T EVER DO THIS

```
CALL -151  
*C055
```

Thank you.

CALL ?SYNTAX ERROR

Every *other* time you CALL 64871 you get a “?SYNTAX ERROR.”

RAM, ROM, PEEK, POKE

You can PEEK at RAM and ROM, but you can only POKE into RAM. That's why you can change DOS, but not Applesoft. (If you move

Applesoft into RAM, you can change it, but then there are other problems.)

BACKSCROLL

If you have a flashing cursor near the bottom of the screen, you can scroll text up *one* line with a backspace or up *two* lines with a carriage return. To get the cursor to the bottom, you can type TEXT.

CALL THIS NUMBER

CALL-1184 will retrieve a message for you from the Autostart ROM.

WHO CARES DEPARTMENT

PRINT. will print a zero. PRINT . . will print two zeroes. LET N=. sets N equal to zero. LET N=. . hits you with a “?SYNTAX ERROR.”

AND PEOPLE COMPLAIN ABOUT METRIC!

Have you noticed that to get into this computer stuff, you’ve got to be constantly *converting* things? Decimal to hex, hex to decimal, 3.2 to 3.3 to ProDOS, Applesoft to Integer, machine code to BASIC, screen characters to ASCII code, negative memory addresses to positive, 48K to 64K to 128K. . . . Good grief!

Here are two rules of pinky that we'll pass along at no charge: 4 Sectors used in a program = approximately 1K of memory (a 24-sector program is about 6K). Also, 4000 decimal = approximately 1000 hex.

WE HAVE FOUND . . .

1. If you LIST any program line, the cursor will appear *three* lines below the listing; except if it's the *last* program line, the cursor will appear *two* lines below. If you think of a use for this tip, let us know.
2. Extra colons are often ignored. This statement works even though it looks a little buggy:
IF X=0 THEN :::: PRINT "I TOLD YOU!"
3. Lots of minuses are okay, too, as in:
PRINT-----10
Each pair of negs make a pos.
4. Tracing over a REM statement will add an *extra space* after the word "REM" when you list. The Apple adds a space after all tokens. When you trace over a REM, the Apple considers the traced-over space part of your remark. The same rule applies to tracing over DATA statements.
5. You can't boot from drive #2. But if you have an Apple //c with two disk drives (one internal and one external), put a ProDOS disk in the external drive, press Control-Reset, and then PR#7 <RETURN>. That'll boot from the external drive.
6. Escape @ clears the text screen and leaves you with a cursor, but *no prompt*.
7. In 80 columns, PRINT CHR\$(12) clears the screen, but not in 40 columns. (You can also clear the screen with PRINT "CONTROL-L" in 80 columns.)
8. You can use a GOSUB as a direct keyboard command; good for testing a subroutine.

9. PR36 won't boot a disk.
10. NEXT:NEXT uses less memory than NEXT X,Y and might even be faster (check it and let us know).
11. Everything in a program line after an ONERR statement is ignored by Applesoft.
12. If you put a quote mark at the beginning of a line and key in a message, your characters won't be parsed. Try this:

```
10 GOTO 30
20 "HIYA JACK - THIS IS SOMETHING ELSE
30 END
```

13. Everything in a program line after a REM statement is ignored by Applesoft.
14. Everything in a program line after a GOTO statement is ignored by Applesoft, but everything after a GOSUB is *not*!
15. A language card has to go in slot #0 on the Apple II/II+, but other cards, except the 80-column card on the //e, may go anywhere. They have to make do with a measly 128K. (?) Traditionally, printers are connected to slot #1 and disk drives are connected to slot #6. Even the Apple //c, which has "ports" instead of real slots, thinks PR#6 will boot a disk, and PR#1 will kick the printer into action.

DO SOMETHING MEAN TODAY!

While strolling by someone's Apple, reach over, POKE 33,90 and keep walking. No permanent damage will be done (Reset cures all), but LIST-ing will be, let's say, interesting. Other POKES to try to temporarily mess things up are POKE 50,99, POKE 50,250, and POKE 50,127.

Beagle Bros is *not* responsible for any possible repercussions.

OUR FAVORITE TYPOS

RUB	PRITN
RIM	LIAR
RYB	KUST
RUIN	;OST
LSIT	LIDY
CATAKIG	CATALOAG

WHAT NO END?

On Applesoft programs, you don't need an END statement, but in Integer you do.

ERROR BREAK

If you try to load a nonexistent file with an immediate (not in a program) command, LOAD XYZZY, you get a normal "FILE NOT FOUND" message. If you have a program in memory that has been run, and type an immediate (again, not in your program) RUN XYZZY, you get a "FILE NOT FOUND, BREAK IN 123" message, where 123 is the last line executed.

FUNNY INVERSE

We just bought a new Apple and it puts out inverse type that looks like this

INVERSE

instead of our old Apple's style

INVERSE

Nobody knows why. We all prefer the old version, probably because we're used to it. Can anyone out there tell us how to convert our hardware to the old-style inverse?

80-COLUMNS OR 40?

If $\text{PEEK}(4925) + \text{PEEK}(49927) = 80$ then an 80-column card is installed. To discover if the user is looking at 80-columns or 40, clear the screen (HOME) and print a period. If $\text{PEEK}(1024)$ (upper-left screen character) is equal to 160 (a space) then 80-columns is in effect.

PRINTER ON?

If your printer is off and a program does a PR#1, everything will screech silently to a halt until someone turns on the printer or does a control-Reset. The problem is, there is *no* message on the screen to tell you what to do. WHY doesn't more software (like *AppleWorks*, for example), use this simple trick?

```
100 FLASH: PRINT "FIX PRINTER"  
110 NORMAL  
120 PRINT CHR$(4); "PR#1": PRINT  
130 PRINT CHR$(4); "PR#0": HOME  
(Program continues)
```



ANOTHER FREE CHART!?

If you've got an 80-column printer, here's a handy chart program. Just type it in, run it, and get out of the way—

```

10  FOR I = 1 TO 69:L$ = L$ + "-": NEXT
15  PRINT CHR$(4) PR# 1: REM PRINTER SLOT
20  FOR I = 1 TO 6: PRINT : NEXT
30  PRINT SPC( 3);"INVERSE"; SPC( 8);"
    FLASH"; SPC( 10); "CONTROL"; SPC( 12
    );"NORMAL": PRINT L$
40  FOR V = 0 TO 63: FOR N = V TO V + 192
    STEP 64
50  IF N > 159 AND N < 255 THEN X$ = CHR$
    (N): GOTO 90
60  IF N = 255 THEN X$ = " ": GOTO 90: REM
    MAKES CHR$(255) PRINT AS A SPACE.
    DEPENDS ON YOUR PRINTER.
70  X = N: X = X + 192 * (N < 32) + 128 *
    (N > 31 AND N < 96) + 64 * (N > 95)
80  X$ = CHR$(X)
90  PRINT " ";: GOSUB 140: PRINT " ";X$;
    " ";N; SPC( (N < 100) + (N < 10));" ";:
    IF N > 127 THEN PRINT "(";N - 128;") ";
    SPC( (N - 128 < 100) + (N - 128 < 10));
100 PRINT " ";: NEXT N: PRINT : IF V + 1 -
    INT ((V + 1) / 16) * 16 > 0 THEN 130
120 PRINT :T = T + 1: IF T = 2 THEN PRINT
    SPC( 3);"INVERSE"; SPC( 7);"FLASH";

```

```

      SPC( 9);"NORMAL"; SPC( 13);"LOWER CASE
      ": PRINT L$
130 NEXT V: FOR I = 1 TO 6: PRINT : NEXT :
      PRINT CHR$(4);"PR#0": END
140 PRINT "$";:N% = N / 16: GOSUB 150: N% =
      N - N% * 16: GOSUB 150: RETURN
150 PRINT CHR$ (48 + N% + 7 * (N% > 9 )),:
      RETURN

```

This chart shows you the values for all 256 normal, inverse, flashing and control characters. It also serves as a handy ASCII conversion chart and 0-255 hex converter. The program *was* going to include a complete biorhythm chart for everyone born in this century and a full-size hi-res rendering of Zasu Pitts, but we're saving those features for Tip Book #32767.

TOWERS OF HANOI

This is a puzzle involving three vertical pegs. On peg #1 there is a stack of disks (not floppy) of decreasing sizes. The object is to transfer these disks, one at a time, in the same order, to peg #2. Use peg #3 as temporary storage. The big stickler is that you can never place a disk on top of one that is smaller. RUN the program that follows and your Apple will solve the puzzle for you. Use the RETURN key to view the process step by step. You may want to alter the program so that you can make the moves yourself.

One of the neat tricks in this program occurs in line 1000, which G:OSUBs itself repeatedly until N(TS) equals zero.

```

20 DIM N(20),X(20),Y(20),Z(20): PRINT
      CHR$(21)
30 TEXT : HOME : INVERSE : PRINT " TOWERS
      OF HANOI ": PRINT : NORMAL : INPUT
      "NUMBER OF DISKS (1-10):";N$:N = VAL
      (N$):N = INT (N): IF N < 1 OR N > 10
      THEN 30

```

```

35 HOME : PRINT "<ANY KEY> TO PAUSE,
   <SPACE> TO CONTINUE": PRINT "<RETURN> TO
   STEP THROUGH"
40 X = 1:Y = 2:Z = 3: GOSUB 1050: GOSUB
   1000: END
1000 TS = TS + 1:N(TS) = N:X(TS) = X:Y(TS)
   = Y:Z(TS) = Z: IF N(TS) THEN N = N(TS)
   - 1:X = X(TS):Y = Z(TS):Z = Y(TS):
   GOSUB 1000:M1 = X(TS):M2 = Y(TS):
   GOSUB 1030:N = N(TS) - 1:X = Z(TS):Y =
   Y(TS):Z = X(TS): GOSUB 1000
1020 HTAB 1: VTAB 2: NORMAL :TS = TS - 1:
   RETURN
1030 T = M1:M = T(T,TP(T)): GOSUB 5000:T =
   M2: GOSUB 4000: IF PEEK ( - 16384) <
   127 THEN RETURN
1040 POKE - 16368,0
1045 A = PEEK ( -16384): IF A < > 160 AND A
   < > 141 THEN 1045
1046 IF A < > 141 THEN POKE - 16368,0
1047 RETURN
1050 TS = 0: INVERSE : FOR I = 1 TO 3: FOR
   J = 4 TO 22: HTAB 13 * I - 12: VTAB J:
   PRINT " ": NEXT J,I: NORMAL
1060 T = 1: FOR I = 10 TO 1 STEP - 1:C(I) =
   T + 2:T = T + 1: NEXT :T = 1: FOR M =
   N TO 1 STEP - 1: GOSUB 4000: NEXT :
   RETURN
4000 TP(T) = TP(T) + 1:T(T,TP(T)) = M:
   INVERSE : HTAB 13 * T - 12: VTAB 24 -
   2 * TP(T): PRINT SPC( M + 1): RETURN
5000 NORMAL : HTAB 13 * T - 12: VTAB 24 - 2
   * TP(T): PRINT SPC( 12): HTAB 13 * T -
   12: VTAB 24 - 2 * TP(T): INVERSE :
   PRINT " ":TP(T) = TP(T) - 1: RETURN

```

ONE LEG AT A TIME

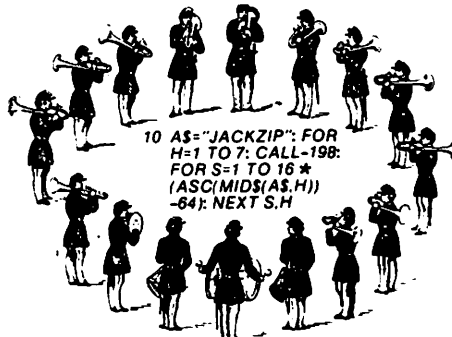
Running this program may help explain the discomfort you're having while sitting at your keyboard. Then again, it may not . . .

```

1 GOTO 2 POS > USR SGN COLOR= = TO OR ABS
  INT COLOR= TO SGN AT COLOR= > OR COLOR=
  FN TO SPC(/ SCRNTD SGN THEN INT RESTORE
  PLOT
2 FOR L=2055 TO 2084: PRINT CHR$(PEEK(L));:
  NEXT

```

In line 2 we're peeking at the garbage in line 1. Each token (like "POS") has an equivalent letter (like "Y"). Strange, huh?



JUST ANOTHER DICE PROGRAM

This program demonstrates two things:

1. Most dice, even computer dice, are normal (REM: Some dice players are not).
2. You can make a nice bar graph on the text screen. (REM: Who needs graphics?).

This program, in the long RUN, should prove that lucky seven comes up more than any other number. Line 20 rolls the dice and line 30 prints the graph. First one to the right side of the screen wins!

```

10 PRINT CHR$(21): DIM N(40): TEXT : HOME :
   NORMAL : FOR V = 2 TO 12: VTAB V * 2 -
   3: HTAB 1 + (V < 10): PRINT V: NEXT
20 D1 = INT ( RND (1) * 6) + 1:D2 = INT (
   RND (1) * 6) + 1:D = D1 + D2:N(D) = N(D)
   + 1
30 VTAB D * 2 - 3: HTAB 3: INVERSE : PRINT
   SPC ( N(D)): IF PEEK (36) THEN 20
40 NORMAL : FOR V = 2 TO 12: VTAB V * 2 -
   3: HTAB 3: PRINT ":";N(V); SPC( N(V) <
   10): NEXT : PRINT CHR$ (7)

```

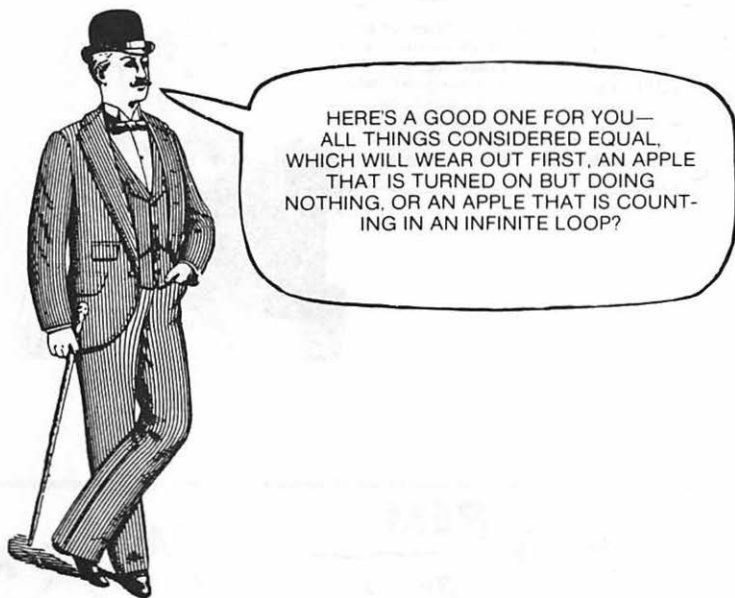
CARD SHUFFLER

This program shuffles cards. It doesn't play cards, it just shuffles them.

```

10 PRINT CHR$(21): DIM CARDS$(52),CHECK(52)
20 SUIT$ = "CDHS"
30 N$ = "A23456789TJQK"
40 FOR S = 0 TO 3: FOR C = 1 TO 13: CARD$(C
   + S * 13) = MID$ (N$,C,1) + MID$ (SUITS
   ,S + 1,1): NEXT : NEXT
45 TEXT : HOME : NORMAL
50 FOR X = 1 TO 52
55 XX = X - 1: VTAB 1 + XX - INT (XX / 13)
   * 13: HTAB 1 + 10 * INT (XX / 13)
60 PRINT SPC( X < 10);X;":";
70 N = INT ( RND (1) * 52) + 1
80 IF CHECK(N) THEN 70
90 CHECK(N) = 1
100 INVERSE : PRINT CARD$(N);
105 NORMAL : PRINT SPC( 5)
110 NEXT X

```



HUMONGOUS TIP BOOK

Buy *The Apple Almanac* by Eric Goetz and William B. Sanders. We don't sell it, but the stores do (and they're not paying for this plug). Ask for it at your computer store or bookstore. It contains 240 pages of all kinds of excellent Apple information, which you are always having to look up, for \$19.95. Don't think about it, *buy it!* You'll be glad you did.

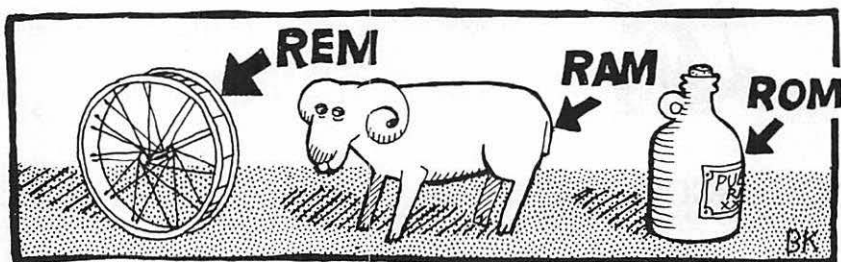
"OH, COME ON!" DEPARTMENT

Our favorite addresses—

Personal Computing Magazine
4 Disk Drive
Philadelphia, PA

Micro Magazine
PO Box 6502
Chelmsford, MA

Your disks? Let's see... They were getting dusty, so I sent them out to the dry cleaners this morning. And I couldn't get Channel 8 on your cute little monitor, so I called the repairman. And... Oh! Remember all those messy WIRES that were sticking out of the back of your Apple?...



VOCABULARY LESSON #255

```
1 GOTO 2 AND POS COLOR=OR
  TO AND AT COLOR=-INT
  COLOR=INT USR AT RESTORE
  PLOT-COLOR= ^ -FRE AT
  COLOR=OR AT TO SGN COLOR=
  ABS+AT COLOR=SQR>>RESTORE
  PLOT PLOT SCRN(+>COLOR=
  TO SGN AT COLOR=POS>USR
  NEW PLOT
2 HOME: FOR L=2055 TO 2105:
  PRINT CHR$(PEEK(L));:NEXT
```

UL APPROVED



● The Beagle Bros Staff ●



Al Gorithm



Len Adollar



Flo Chart



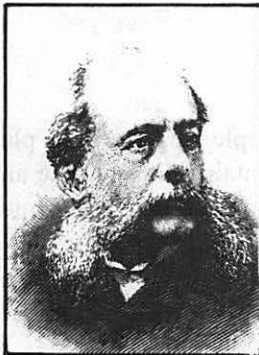
Tex Window



C. Ray Tube



I.O. Socket



Max Files



Minnie Assembler



J. Modulo DeBug

Entire contents copyright ©1981 by Beagle Bros. Micro Software
 Reproduction in any form is illegal without the expressed written permission
 of the Commissioner of Baseball.

No longer with us: Elsie Dee and Daisy Wheel

Photo credits: G. Bell (page 2), R.F. Modulator (page 3), Jose Canyusee (page 5), Roger Overout (page 7), Bill Overdoo (pages 8,13,14), Mandy Lifeboats (pages 9,22,58), Tom P. Ping (page 10), Hi-Res Harry (page 17) courtesy Fly-by-Night Circuses Intl., Apple-0 Computer (page 15) courtesy Beagle Bros Hardware Research Department. Figure 8 illustration (page 26) courtesy National Museum of Overpriced Art. Free Cash offer (page 425) expires December 31, 1999.

MORE-ON-THE-SCREEN TIP

Some word processors display text on the screen in the same column-width that it will be printed; the default is usually around 60 characters. This is fine, but the right 20 screen-columns are wasted, and therefore the total amount of text visible is limited. You'll get more on the screen if you set the column-width to 80 until you are ready to print. With *AppleWorks*, set the right margin to zero and the platen width to 9.

We decided it would be nice to have a triple-height full-page monitor for Apple word processing. You build it and Beagle Bros will handle the marketing. Please phone us when you've built the prototype.

BOOT DATE

If you have a Thunderclock connected to your Apple //e, you can play with this startup program. Line 0 has a REM that contains the last time and date booted. Every time the program is RUN, the time and date are printed on the screen, updated in the listing, and re-*SAVED* on disk.

```
1000 PRINT "LAST BOOT: ";X$ = "": FOR I =
      2054 TO 2054 + 21:X$ = X$ + CHR$ (
      PEEK (I)): NEXT I: PRINT X$: REM READS
      LINE 0 AND PRINTS IT
1010 D$ = CHR$ (4): PRINT D$"PR#7": PRINT
```

```
D$"IN#7": INPUT "%";T$: PRINT D$"IN#0
": PRINT D$"PR#0": REM READS CLOCK
INTO D$
1020 PRINT "THIS BOOT: ";T$: FOR I =1 TO
22: POKE 2053 + I, ASC ( MID$ (T$,I)):
NEXT : REM POKES T$ INTO LINE 0,
CHANGING THE DATE THERE
1030 PRINT D$"SAVE STARTUP": REM SAVE FOR
NEXT TIME YOU BOOT
```

GRID PAPER

The next time you need a piece of graph paper, RUN the following program. Use your printer's special-effect features to adjust the darkness of lines, compactness of grid, etc. Different brands of printers will produce different results; check your printer manual (good luck).

You can print as many grid copies as you like, or—to combat noise pollution and save on ribbons—take a master printout to your local instant printer or photocopy machine.

```
10 TEXT : HOME
20 PRINT CHR$(4) "PR#1"
40 REM SAMPLE GRID PROGRAM
100 V$ = CHR$ (124):L$ = CHR$ (95):S$ =
CHR$ (32)
200 FOR H = 1 TO 24: PRINT S$;L$;L$;: NEXT
: PRINT S$
400 FOR V = 1 TO 40: FOR H = 1 TO 24: PRINT
V$;L$;L$;
500 NEXT : PRINT V$: NEXT
600 PRINT CHR$(4) "PR#0"
```

POKE FIX

If typing RUN gives you a weird message like “?Syntax Error In 65064” or if NEW causes a “?Syntax Error,” a quick POKE 2048,0 should fix things. If you're into causing problems instead of fixing them, type POKE 2048 (the byte prior to start-of-program) with a number other than zero.

CONTROL YOUR PRINTER

The manual that came with your printer probably tells you which control or Escape command creates which special effect (bold characters, condensed, etc.), but I don't trust my printer manual any further than I can throw it (and I've thrown it many times). Why not test your printer? Who knows, maybe it prints some character set or special effect that they never even told you about.

```
10 D$ = CHR$ (4)
20 HOME : PRINT "CODE:";: GET A$
30 IF A$ = CHR$ (27) THEN PRINT "ESC-";:
   GET B$
40 PRINT D$"PR#1"
50 PRINT A$;B$;" <-WHAT DOES THIS DO?"
60 PRINT D$"PR#0": RUN
```

To clear previously installed printer commands, you should click your printer's power switch off and back on again each time you see the word "CODE:". Then enter a control character (like Control-A), or Escape followed by another character (for example, Escape A). If "What does this do?" changes appearance, you've discovered something. By the way, Escape is the same as Control-[]; that's CHR\$(27).

EXPONENT

Several Apple II hardware and software configurations now accept lower-case direct and indirect commands. Try typing "print 123" in lowercase. If you get a "?Syntax Error," don't bother reading the next paragraph.

Oh boy, another quirk to write about! Type the following command. It should produce a "?Syntax Error" but it doesn't:

```
print 2~2
```

ASCII-ly speaking, the tilde (~) is a lowercase caret (^). Remember, you read it here!

WORTHLESS TRIVIA (DO NOT READ!)

If you're using all positive values, you can substitute a *minus sign* for ">". Ferinstance

IF X>Y THEN PRINT "HICCUP"

is the same as

IF X-Y THEN PRINT "HICCUP"

BEAGLE BROS LOCATIONS



WHO ARE THE BEAGLE BROS?

Applers from all over the world have written and asked who the Beagle Bros really are and how we got our start in the software business. Well, it was quite by accident, really. Just before the war, we bought a . . .

CHAPTER 19

TWO LINERS

IN ONLY TWO LINES

For those of you who are unfamiliar with the Beagle Bros Perpetual (or until it's cancelled, whichever comes first) Two-Liner Contest, you're going to be amazed at what your trusty Apple can do in two lines. This chapter presents the unabridged collection of two liners collected up until about a week ago. The first line doesn't count, since it is used to identify the program's author. If you have a two-line program you think is something else, send it to:

Beagle Bros Micro Software, Inc.
2-Liner Department
3990 Old Town Avenue
San Diego, CA 92110

The nice thing about two liners is they're easy to key in. There is absolutely no joy in keying in a thousand-line program and getting a hidden "?SYNTAX ERROR." With two liners the error is in line 1 or line 2.

XPLOT

```

0 REM DIVAN ROUSE--COUNTRYSIDE,IL
1 HGR2 :P = 3.14:Q = 6.28: FOR T = 1 TO 3:L
  = P / 40: HCOLOR= INT ( RND (1) * 6) +
  1:R = RND (1) * 60:S = 90 - R:B = RND (1)
  * Q - P: FOR A = 0 TO Q STEP L:X = INT (
  COS (A) * S * 1.55) + 140:Y = INT ( SIN
  (A) * S) + 95:H = INT ( COS (A + B) * R):
  HPLOT X + H,Y + H TO X - H,Y - H: HPLLOT X
  - H,Y + H TO X + H,Y - H: NEXT : NEXT :
  RUN

```

YAWNER

```

0 REM THOMAS GILLEY--ELLSWORTH, ME
1 HGR2
2 RESTORE : FOR I = 1 TO 15: READ X: POKE
  24575 + I,X: NEXT : POKE 232,0: POKE 233,
  96: ROT= 00: FOR S = 1 TO 255: SCALE= S:
  XDRAW 1 AT 139,95: NEXT : GOTO 2: DATA 1,
  0,4,0,0,40,53,54,62,63,39,36,44,5,0

```

ZOOM GRAPHICS

```

0 REM CHRIS KOERITZ--FAYETTEVILLE, NY 13066
1 HGR :HC = 3:Y = 191: POKE - 16302,0:X =
  279
2 HCOLOR= HC: HPLLOT X,Y TO X,191 - Y TO 279
  - X,191 - Y TO 279 - X,Y TO X,Y:Y = Y -
  5 + 190 * (Y < 2):X = X - 5 + 275 * (X <
  5):HC = HC - 3 * (HC = 3 AND RND (1) >
  .94) + 3 * (HC = 0 AND RND (1) > .9):
  GOTO 2

```

VIDEO VISION

```

0 REM KEVIN OSTROM--POWAY,CA
1 B = 191:C = 278:P = C / B:S = RND (1) * 9
  + 3: HGR2 : FOR I = 1 TO 0 STEP -1:
  HCOLOR = I * 3: FOR A = 0 TO B STEP S:
  HPLLOT 0,A TO A * P,B TO C,B - A TO C - A

```

```

* P,0 TO 0,A: NEXT : NEXT :X = RND (1) *
C:Y = RND (1) * B: FOR I = 1 TO 0 STEP -
1: HCOLOR= I * 3: FOR A = 0 TO B STEP S:
HPlot 0,A TO X,Y TO C,B - A: HPlot A * P,
B TO X,Y TO C - A * P,0
2 NEXT: NEXT : FOR I = 1 TO 0 STEP - 1:
HCOLOR= I * 3: FOR A = 0 TO B STEP S:
HPlot 0,0 TO C,A: HPlot 0,B TO C,B - A:
HPlot C,0 TO 0,A: HPlot C,B TO 0,B - A:
NEXT : NEXT : FOR I = 1 TO 0 STEP - 1:
HCOLOR= I * 3: FOR A = 0 TO B STEP S:
HPlot 0,A TO X,Y TO C,A: HPlot 0,B - A TO
X,B - Y TO C,B - A: NEXT : NEXT : GOTO 1

```

WAR!

```

0 REM JOHN WOO--BRONX, NY
1 GR : HOME : COLOR= 8: HLin 0,39 AT 38:
COLOR= 11: HLin 14, 21 AT 37: HLin 14,21
AT 33: VLin 34,36 AT 13: VLin 34,36 AT 2
2: PLOT 20,35: PLOT 16,35: HLin 17,20 AT
32: HLin 17,20 AT 31: HLin 19,20 AT 30:
HLin 11,20 AT 29: PLOT 19,28: VLin 28,30
AT 11: PLOT 21,28: COLOR= 12: FOR X = 1
TO 2345: NEXT
2 FOR X = 10 TO 0 STEP - 1: PLOT X,29: FOR
T = 1 TO 20: NEXT T: COLOR= 0: PLOT X,29:
COLOR= 12: NEXT X: FOR X = 39 TO 21 STEP
- 1: PLOT X,29: FOR T = 1 TO 20: NEXT T:
COLOR= 0 : PLOT X,29: COLOR= 12: NEXT X:
TEXT : HOME : VTAB 19: HTAB 15: PRINT
"B O O M !"

```

SARGON LXXXIV

```

0 REM AJIT JOSHI--CANTON,MI
1 FOR A = 1 TO 18: READ X: POKE 767 + A,X:
NEXT : HGR : POKE 28,63: CALL 62454:
HGR2: POKE 28,64: CALL 62454: CALL 768:
TEXT : HOME : POKE -16368,0
2 DATA 173,84,192,173,0,192,201,127,16,7,
234,173,85,192,76,0,3,96

```


TRIPLE WEB

```

0 REM BILL POMPEII--WINDSOR,NY
1 HGR2 : HCOLOR= 3:LI = 63:RI = 0: FOR V =
  1 TO 3:RT = RI:LF = LI:TP = 0:BT = 279:YT
  = RI:YB = LI: FOR LP = 1 TO 16
2 HPLOT TP,YT TO 279,RT TO BT,YB TO 0,LF TO
  TP,YT: LET TP = TP + 17.4:BT = BT -
  17.4:RT = RT + 4:LF = LF - 4: NEXT LP: LI
  = LI + 64:RI = RI + 64: NEXT

```

UL APPROVED

```

0 REM ANDREW CALDWELL--LONG BEACH,CA
1 A = INT ( RND (E) * F):B = INT ( RND (E)
  * G):C = INT ( RND (E) * F):D = INT ( RND
  (E) * G): IF E = 0 THEN HGR2 :E = 1:F =
  279:G = 191:H = 3
2 A + A + SGN (C - A):B = B + SGN (D - B):H
  = ABS (H - 3): HCOLOR= H: HPLOT C,D TO
  A,B: HPLOT F - C,D TO F - A,B: HPLOT C,G
  - D TO A,G - B: HPLOT F - C,G - D TO F -
  A,G - B: ON ABS ((C - A) / 2 + E) GOTO 1:
  ON ABS (D - B + E) GOTO 1: GOTO 2

```

PULSE METER

```

0 REM JOHN ROMERO--ROCKLIN,CA
1 HOME : VTAB 22: PRINT "PLACE FINGER ON
  SPACE BAR TO READ PULSE.": VTAB 20:S =
  49200: HGR : HCOLOR= 3: FOR Y = 0 TO 160
  STEP 10:Y = Y - (Y = 160): HPLOT 0,Y TO
  279,Y: NEXT : FOR X = 0 TO 280 STEP 10:X
  = X - (X = 280): HPLOT X,0 TO X,190: NEXT
  : VTAB 1
2 HCOLOR= 3: HPLOT 0,100: FOR X = 0 TO 276
  STEP 3: GET A$: PRINT CHR$ (7 - (A$ = "
  ))): HPLOT TO X, INT ( RND (1) * 80 +
  35) TO X + 3, INT ( RND (1) * 80 + 35):P
  = PEEK (S): NEXT : HOME : VTAB 22: PRINT
  "THANK YOU. THAT WILL BE 25 CENTS.": END

```

QUEST

```

0 REM SCOTT BROS--FT WORTH,TX
1 DATA 1,0,4,0,18,63,39,36,44,45,45,53,54,
  62,63,0: FOR I = 7676 TO 7691: READ A:
  POKE I, A: NEXT : HGR2 : POKE 232,252:
  POKE 233,29: ROT= 0
2 FOR C = 1 TO 7: HCOLOR= C: FOR R = 1 TO
  140: SCALE= R: DRAW 1 AT 140,96: NEXT :
  FOR S = 140 TO 1 STEP - 1: SCALE= S :
  XDRAW 1 AT 140,96: NEXT : FOR I = 1 TO
  1000: NEXT : NEXT : GOTO 2

```

ROUND ROBIN

```

0 REM JIM ROTH--SAN ANTONIO,TX
1 HGR2 : HCOLOR= 7: HPLLOT 0,0 TO 279,0 TO
  279,191 TO 0,191 TO 0,0
2 X1 = INT (100 * RND (1) + 60) :Y1 = INT
  (80 * RND (1) + 20):X2 =130 * SIN (A) +
  138:Y2 = 90 * COS (A) + 95: HPLLOT X1,Y1
  TO X2,Y2:A = A + .1:C = INT (7 * RND (1)
  + 1): GOTO 2

```

MERCATOR

```

0 REM IMAD DAGHER--ANAHEIM,CA
1 HGR2 : HCOLOR= 7:H = 79.5:G = 159
2 FOR A = 0 TO 12.6 STEP .2:X = 22 * A:Y =
  G - H * ( SIN (A) + 1): HPLLOT X,Y: NEXT
  A:G = G - 7:H = H - 7: IF H > - 79.7
  THEN 2

```

NUMEROSITY

```

0 REM SCOTT BROS--FT WORTH,TX
1 DATA 1,0,4,0,18,63,32,100,45,21,54,30,7,
  0: FOR I = 7676 TO 7689: READ A: POKE I,
  A: NEXT : HGR2: POKE 232,252: POKE 233,
  29: ROT= 0
2 FOR C = 1 TO 7: HCOLOR= C: FOR R = 1 TO
  140: SCALE= R: DRAW 1 AT 140,96: NEXT :

```

```
FOR S = 140 TO 1 STEP - 1: SCALE= S :  
XDRAW 1 AT 140,96: NEXT : GOTO 2
```

DP IMAGES

```
0 REM MORTEN OLSEN--DENMARK  
1 TEXT : HOME : PRINT "(TRY 30, 60, 90, 120  
AND 150 AS INPUT .)": PRINT : : INPUT  
"DIFFERENCE IN DEGREES:";D:D = D / 5  
7.29578: NORMAL : HGR2 : HCOLOR= 3 :  
HPLLOT 140,96: DNERR GOTO 1  
2 FOR R = 0 TO 200 STEP D:X = 140 + R *  
COS (R):Y = 96 + 0.7 * (R * SIN (R)):  
HPLLOT TO X,Y: NEXT
```

JUNKTURE

```
0 REM DMP SOFTWARE--ST.LOUIS,MO  
1 HGR2 : T = 0:B = 160:L = 0:R = 256: FOR X  
= 1 TO 11: FOR C = 1 TO 6 STEP 1: HCOLOR=  
C: HPLLOT R,T TO L,T: HCOLOR= C: HPLLOT R,B  
TO L,B:T = T + 2:B = B - 2:L = L + 2:R =  
R - 2: NEXT C: HPLLOT R,T TO L,T: HPLLOT  
R,B TO L,B: NEXT X  
2 T = 0:B = 165:L = 0:R = 256: FOR X = 1 TO  
40: HCOLOR= 3: HPLLOT R,B TO L,B TO L,T TO  
R,T TO R,B:T = T + 3: B = B - 3:L = L +  
3:R = R - 3: NEXT X: FOR J = 1 TO 39:  
HCOLOR= 4: HPLLOT R,B TO L,B TO L,T TO R,T  
TO R,B:T = T - 3:B = B + 3:L = L - 3:R =  
R + 3: NEXT J: GOTO 1
```

KRASH

```
0 REM YUICHI HANDA--PASADENA,CA  
1 HGR2: HCOLOR= 3: HPLLOT 0,0 TO 279,0 TO  
279,191 TO 0,191 TO 0,0  
2 R = INT ( RND (1) * 255): POKE 228,R: FOR  
T = 1 TO 278 STEP 4: HPLLOT T,0 TO 0,191:  
HPLLOT 278 - T,191 TO 278,0: NEXT : S =  
PEEK ( - 16336):S = PEEK ( - 16336): GOTO  
2
```

LISSAJOUS

```

0 REM JIM ROTH--SAN ANTONIO,TX
1 HGR2 : HCOLOR= 7: HPL0T 0,0 TO 279,0 TO
  279,191 TO 0,191 TO 0,0:A = 0:C = 0
2 X1 = 134:Y1 = 92:X2 = 130 * SIN (A * 2) +
  138:Y2 = 90 * COS (3 * A) + 95: HPL0T X1,
  Y1 TO X2,Y2:A = A + .1: GOTO 2

```

I-CHART

```

0 REM YUICHI HANDA--PASADENA,CA
1 HGR2 : FOR A = 1 TO 140: POKE 228,A:
  HPL0T A,0 TO 0,191 - A: HPL0T 0,0 + A TO
  A,191:B = 140 - A: HPL0T B,0 TO 140,191
  - A: HPL0T 140,0 + A TO B, 191:Z = A +
  139: HPL0T Z,0 TO 139,191 - A: HPL0T
  139,0 + A TO Z,191:Y = 279 - A: HPL0T Y,0
  TO 279,191 - A: HPL0T 279,0 + A TO Y,
  191:S = PEEK ( - 16336): NEXT
2 HCOLOR= 3: HPL0T 67,94 TO 69,94 TO 69,96
  TO 67,96 TO 67,94 : HPL0T 207,94 TO 209,
  94 TO 209,96 TO 207,96 TO 207,94: FOR T =
  1 TO 150: NEXT : HCOLOR= 0: HPL0T 67,94
  TO 69,94 TO 69,96 TO 67,96 TO 67,94:
  HPL0T 207,94 TO 209,94 TO 209,96 TO 207,
  96 TO 207,94: FOR T = 1 TO 150: NEXT :S =
  PEEK ( - 16336): RUN 2

```

GROWIES

```

0 REM VERA & GEOFF WONG--VICTORIA,
  AUSTRALIA
1 HGR2 : HCOLOR= 1: FOR I = 0 TO 275 STEP
  3: HPL0T I,191 TO I + 1,191 - ( RND (1) *
  15): HPL0T I + 1,191 - ( RND (0) * 15) TO
  I + 2,191: NEXT : FOR I = 1 TO 500: NEXT
  : GOTO 2
2 P = 3.14: FOR L = 1 TO 15:A = RND (1) *
  219 + 30:B = RND (1) * 131 + 30: HCOLOR=
  3: FOR Y = 191 TO B STEP - 1: HPL0T A, Y:
  NEXT :S = RND (1) * 20 + 10: FOR T = .5 *
  P TO 2.5 * P STEP 2 * P / S:X = A + SIN

```

```
(T) * S:Y = B - COS (T) * S : HPLOT A,B
TO X,Y: NEXT T,L
```

HPLOT MADNESS

```
0 REM DAN PROTHERO--OSTERVILLE,MA
1 T = 768: POKE 232,0: POKE 233,3 : FOR L =
  T TO T + 24: READ P: POKE L,P: NEXT :
  DATA 5,0,12,0,15,0,18,0,21,0,24,0,53,39,
    0,55,37,0,39,53,0,37,55,0,45,0: HGR :
  POKE - 16302,0: ROT= 0
2 FOR J = 0 TO 7: HCOLOR= J: HPLOT 0,0:
  CALL - 3082: FOR R = 1 TO 2: FOR S = 1 TO
    191: SCALE= S: XDRAW 1 AT S,S: XDRAW 2 AT
    279 - S,S: XDRAW 3 AT 279 - S,191 - S:
    XDRAW 4 AT S,191 - S: NEXT : SCALE= 140:
    FOR Y = 0 TO 191: XDRAW 5 AT 0,Y: NEXT :
    NEXT : NEXT: GOTO 2
```

DAYTER

```
0 REM PHIL BARTON--SAULT STE.MARIE
1 TEXT : HOME : PRINT "DAYTER: INDICATES
  DAY FOR VALID DATE" : INVERSE : VTAB 3:
  PRINT "SAT  SUN  MON  TUE  WED  THU
  FRI": NORMAL : VTAB 6 : INPUT "MMDDYY:";A
  $:A% = VAL ( RIGHT$ (A$,2)):B% = A% / 4
  :C% = VAL ( LEFT$ (A$,2)):E = (B% * 4 =
  A% AND C% < 3 AND A% < > 0):A% = A% + B%
2 D% + (C% = 1) + 4 * (C% = 2 OR C% = 3 OR
  C% = 11) + 2 * (C% = 5) + 5 * (C% = 6) +
  3 * (C% = 8) + 6 * (C% = 9 OR C% = 12) +
  (C% = 10) + - E:A% = A% + D% + VAL ( MID$
  (A$,3,2)):B% = A% / 7:C% = B% * 7: B% =
  A% - C%: VTAB 4: CALL - 868: HTAB 6 * B%
  + 1: PRINT CHR$ (7);"^^^": VTAB 10: PRINT
  "PRESS RETURN ";: CALL - 676: GOTO 1
```

EYE SCAN

```
0 REM STEVEN BLOCK--PASADENA,CA
1 POKE 233,3: POKE 232,2: FOR J = 1 TO 14:
```

```

READ D: POKE 769 + J,D: NEXT : HGR2 :
SCALE= 40 : HCOLOR= 3
2 FOR R = 1 TO 64: ROT= R: XDRAW 1 AT 140,
90: NEXT :I = I + 10* RND (I) * (I <
254): POKE 776,I: RESTORE : GOTO 2: DATA
1,0,4,0,2,4,2,4,2,4,2,4,0,6

```

FLOP CAT

```

0 REM REED RIGHTEAD--SAN DIEGO,CA
1 FOR A = 768 TO 785: READ B: POKE A,B:
NEXT : POKE 54,0: POKE 55,3: CALL 1002:
DATA 201,141,240,9,32,240,253,32,16,252,
76,16,252,169,39,76,100,252
2 PRINT CHR$ (4) "CATALOG"
3 REM CONTROL-RESET TO CANCEL

```

APPLE 3D

```

0 REM MIKE NUZZO--WESTFIELD,NJ
1 HGR2 : HCOLOR= 3:J = 0: CX = 140: CY =
80: A1 = 260: A2 = 14: PI = 3.14159: X1 =
72: Y1 = 13: X2 = 46: Y2 = 69: FOR I = 0 TO
PI * 2 STEP PI / A1: X5 = SIN (I) * X1: Y5
= COS (I) * Y1: GOSUB 2: X7 = INT ((X5 +
X6) / 2): Y7 = INT ((Y5 + Y6) / 2): HPLLOT
X7 + CX, Y7 + CY: NEXT : END
2 J = J + PI / A2: X6 = SIN (J) * X2: Y6 =
COS (J) * Y2: RETURN

```

BOX BOUT

```

0 REM SCOTT BROS--FT WORTH, TX
1 DATA 1,0,4,0,18,63,39,36,44,45,45,53,54,
62,63,0: FOR I = 7676 TO 7691: READ A:
POKE I,A: NEXT : HGR2 : POKE 232,252:
POKE 233,29: ROT= 0
2 FOR C = 1 TO 7: HCOLOR= C: S = 21: FOR R =
1 TO 20: S = S - 1: SCALE= R: DRAW 1 AT
70,48 : DRAW 1 AT 210,144: SCALE= S:
XDRAW 1 AT 210,48: XDRAW 1 AT 70,144:
NEXT : T = 21: FOR U = 1 TO 20: T = T - 1:
SCALE= T: XDRAW 1 AT 70,48: XDRAW 1 AT

```

```
210,144: SCALE= U: DRAW 1 AT 210,48: DRAW
1 AT 70,144: NEXT : NEXT : GOTO 2
```

CHUGGACHUGGA

```
0 REM AJIT JOSHI--CANTON,MI
1 HOME : POKE 50,233: FOR X = 150 TO 255:
  SPEED= X: PRINT PEEK (49385) + PEEK
  (49386);: PRINT "CHUGGA";: PRINT PEEK
  (49387);: NEXT : END
```

ORGAN

```
0 REM MARK BACKMAN--AUSTIN, TX
1 TEXT : HOME : PRINT TAB ( 13); "KEYBOARD
  ORGAN": FOR X = 24576 TO 24591: READ Y:
  POKE X,Y: NEXT X: DATA 172,0,192,173,48,
  192,234,234,234,234,136,208,249,76,0,96
2 CALL 24576
```

NAMER

```
1 NORMAL : VTAB 1: HTAB 1: INPUT "WHAT'S
  YOUR NAME? ";N$: SPEED= 234:SP$ = "":
  ONERR GOTO 1
2 SP$ = SP$ + ".": INVERSE : PRINT " ";N$;"
  ";: NORMAL : PRINT SP$;: GOTO 2
```

UFD #1--INTEGER BASIC

```
0 REM DAVE MADDEN--LISLE, IL
1 CALL -936: TAB 10: PRINT "U.F.O. TAKEOFF
  EFFECT": TAB 13: PRINT " BY DAVE MADDEN"
2 POKE 766,1: POKE 765,16: FOR A=1 TO 255:
  FOR B=1 TO 75: FOR I=B TO 1 STEP -A: POKE
  767, I: CALL -10473: NEXT I,B,A: GOTO 2
3 REM REQUIRES PROGRAMMER'S AID CHIP
```

UFO #1--INTEGER BASIC

```

0 REM DAVE MADDEN--LISLE, IL
1 CALL -936: TAB 9: PRINT "HOVERING U.F.O.
  EFFECT": TAB 13: PRINT "BY DAVE MADDEN"
2 POKE 766,1: POKE 765,16: FOR I=30 TO 1
  STEP -1: POKE 767,I: CALL -10473: NEXT I:
  GOTO 2

```

LD-RES GAME

```

0 REM DAVID LINKER--SEATTLE,WA
1 GR : HOME : COLOR= 15:T = 39: FOR I = 0
  TO 1: VLIN 0, T AT I * T: HLIN 0,T AT I *
  T: NEXT : X = 5:Y = 5:S = 0:T = 38:
  COLOR= 1: PLOT X,Y: FOR C = 1 TO 14 :A =
  RND (1) * T + 1:B = RND (1) * T + 1:
  COLOR= C + 1: PLOT A,B: COLOR= C: VTAB
  23: PRINT "YOUR SCORES";S: FOR I = 1 TO
  50: IF (N = 15) OR (N < = C AND N > 0)
  THEN END
2 N = PEEK ( - 16368):N = PEEK ( - 16384):G
  = (N = 21) - (N = 8):D = (N = 90) - (N =
  65): ON ABS (G + D) + 1 GOTO 2:X = X +
  G:Y = Y + D:N = SCRN (X,Y):S = S + 200 *
  (N = C + 1):Z = PEEK ( - 16336): PLOT
  X,Y: NEXT :S = S + 100: COLOR= 0: PLOT
  A,B: NEXT : VTAB 23: HTAB 12: PRINT S;"
  YOU WIN!": FOR I = 1 TO 9: CALL -1 052:
  NEXT

```

PATTERNS

```

0 REM DAVID GERSHON--SEAL BEACH,CA
1 HGR : FOR C = 1 TO 7:RT = 15: HCOLOR= C:
  FOR R = 5 TO 150 STEP 10: HPlot 0,R TO
  RT,155 TO 270,155 - R TO 270 - RT,0 TO
  0,R:RT = RT + 15: NEXT R: HOME : VTAB 9:
  PRINT " FUN."

```



```
2 FOR R = 10 TO 60 STEP 10: HPLLOT 140,10 +  
  R TO 140 + R,75 TO 140,150 = R TO 140 -  
  R,75 TO 140,10 + R: NEXT R: NEXT C: HGR :  
  TEXT
```

MYSTERY PIC

```
0 REM MARK BACHMAN--AUSTIN, TX  
1 HGR : FOR Z = 0 TO 10 STEP .2 : FOR X = 0  
  TO 10 STEP .2: HCOLOR= 3:Y = - 10 * COS  
  (3 * SQR ((X - 5) ^ 2 + (Z - 5) ^ 2)) / 2  
  + 50: HPLLOT X * 20 + 20 + Z * 3,Y + Z *  
  10  
2 HCOLOR= 0: HPLLOT X * 20 + 20 + Z * 3,Y +  
  Z * 10 + 1 TO X * 20 + 20 + Z * 3,180:  
  NEXT : NEXT : END
```

SPHERE

```
0 REM JERRY KRAMER--PHILADELPHIA, PA  
1 HOME : HGR2: HCOLOR= 3: FOR A = 20 TO 170  
  STEP 7:B = 1.15 * ( SQR (6400 - (A - 95)  
  ^ 2)): FOR C = 0 TO 6.4 STEP . 2:X = B *  
  COS (C) + 139:Y = A - B * SIN (C) / 5: IF  
  C = 0 THEN HPLLOT X,Y: NEXT  
2 HPLLOT TO X,Y: NEXT : NEXT
```

SCROLLER

```
0 REM CHRISTOPHER VOLPE--TRUMBULL, CT  
1 H$ = "303: AD 52 C0 AD 51 C0 A9 22 EA EA  
  EA EA 20 A8 FC AD 50 C0 A9 47 3E 00 00 3E  
  00 00 3E 00 00 3E 00 00 3E 00 00 3E 00 00  
  3E 00 00 3E 00 00 3E 00 00 3E 00 00 20 A8  
  FC 4C 06 03 N 303G"  
2 FOR I = 1 TO 40: PRINT "THIS IS TEXT PAGE  
  ONE. ";: NEXT : FOR C = 1 TO LEN (H$):  
  POKE 511 + C, ASC ( MID$ (H$,C,1)) + 128:  
  NEXT : CALL -144  
10 REM
```

SALARY

```

0 REM CHRIS JOHNSON--LITTLE ROCK, AR
1 TEXT : HOME : PRINT "YOU ARE OFFERED ONE
  CENT ON THE FIRST DAY OF YOUR NEW JOB, TO
  BE DOUBLED EVERYDAY THEREAFTER FOR A
  MONTH.": PRINT : PRINT "J QUESTION:
  SHOULD YOU ACCEPT?";: FLASH : HTAB 2:
  PRINT " ": NORMAL
2 PRINT : PRINT "DAY","PAY": PRINT "---",
  "-----": POKE 34,8:S = 127 / 31:A =
  .01: FOR B = 1 TO 31: SPEED= 2 * S * B:
  PRINT B,"$";A:A = A * 2: NEXT : PRINT :
  PRINT " ANSWER: HOLD OUT FOR MORE.":
  VTAB 23

```

MEM-SHAPES

```

0 REM DAVID GERSHON--SEAL BEACH,CA
1 POKE 232,255: POKE 233,255:X = INT ( RND
  (1) * 120) + 1:R = INT ( RND (1) * 40):
  HGR2:ROT= R: DRAW X AT 90,90:H = INT (
  RND (1) * 8): IF H = 0 OR H = 4 THEN H =
  1
2 HCOLOR= H: XDRAW X AT 90,90: GOTO 1: REM
  TRY THIS EVERY NOW AND THEN JUST TO SEE
  WHAT'S GOING ON.

```

LIGHTNING--INTEGER BASIC

```

0 REM CHRIS VOLPE--TRUMBULL, CT
1 CALL -936: CALL -12288:X0=Y0=COLR:P=
  -11506:D=32767: FOR I=0 TO D:COLR=
  127:X=A:Y=B:A=RND (279):B= RND (191)
  :X0=X:Y0=Y: CALL P
2 X0=A:Y0=B: CALL P+6:COLR=0: X0=X:Y0=
  Y: CALL P:X0=A:Y0=B: CALL P+6: POKE
  -16336, PEEK (-16336): NEXT I

```

ETC.

```
0 REM DAVE POLIDORI--EYNON, PA
1 Y = RND (1) * 191: HGR : HCOLOR= 7: POKE
  - 16302,0: FOR I = 1 TO 750:X = X + Y / 2
  - 279 * (X > 279):X = X - 279 * (X >
    279):Y = Y - X / 4 + 191 * (Y < 0):Y = Y
    + 191 * (Y < 0): HPLLOT X,Y: HPLLOT X,191 -
    Y: HPLLOT 279 - X,191 - Y: HPLLOT 279 - X,Y
    : NEXT I: GOTO 1
```

G.BELL

```
0 REM G.BELL--SAN DIEGO, CA
1 HOME : HGR : POKE - 16304,0: POKE -
  16302,0: POKE - 16297,0: X = 2:Y = 2:XR =
  X + 2:XL = X - 2:YT = Y - 2:YB = Y + 2:
  HCOLOR=3
2 F = F + 1:XL = XL + F / 9: HPLLOT XR,YT TO
  XL,YT:YB = YB + 2: HPLLOT XL,YT TO XL,YB:
  XR = XR + 2: HPLLOT XL,YB TO XR,YB:YT = YT
  + F / 20: HPLLOT XR,YB TO XR,YT: GOTO 2
```

WHAT?

```
0 REM DENNIS MARTINEZ--ALBUQUERQUE, NM
1 HGR2 :E = INT (191 * RND (1)):D = INT
  (279 * RND (1)): C = INT (3 * RND (1)) +
  2: FOR B = 0 TO 191 STEP C: HCOLOR= INT
  (6 * RND (1)) + 1
2 FOR A = 0 TO 270 STEP C: HPLLOT D,E TO A,
  B: NEXT : NEXT : CALL 62454: FOR T = 1 TO
  100:Z = PEEK (- 16336): NEXT : GOTO 1
```

R.F.MON

```
0 REM R.F.MODULATOR--SAN DIEGO,CA
1 POKE - 16304,0: POKE - 16302,0: POKE -
  16297,0:X = 140: Y = 95:XR = X:XL = X:YT
  = Y:YB = Y: HCOLOR= 3: ONERR GOTO 1
2 C = C + 1: HCOLOR= C - INT (C / 6) * 6:XL
  = XL - 3: HPLLOT XR,YT TO XL,YT:YB = YB +
```

```

2: HPLLOT XL,YT TO XL,YB:XR = XR + 3:
HPLLOT XL,YB TO XR,YB:YT = YT - 2: HPLLOT
XR,YB TO XR,YT: GOTO 2

```

CURVES

```

0 REM JERRY KRAMER, PHILADELPHIA, PA
1 HOME : HGR : HCOLOR= 3: DIM X(21),Y(21):
  FOR T = 2 TO 21:X (T) = 78 * SIN (.314 *
    T) + 140:Y(T) = 78 * COS (.314 * T) + 80
2 NEXT : FOR T = 2 TO 21: FOR Q = T TO 21:
  HPLLOT X(T),Y(T) TO X(Q),Y(Q): NEXT : NEXT
  : VTAB 24

```

TONES

```

0 REM BRUCE JOHNSON--REARDAN, WA
1 READ A,B: POKE A,B: DATA 770,173,771,48,
  772,192,773,136,774,208,775,5,776,206,
  777,1,778,3,779,240,780,9,781,202,782,
  208,783,245,784,174,785,0,786,3,787,76,
  788,2,789,3,790,96:N = N + 1: IF N < > 21
  THEN 1
2 HTAB 81: INPUT "FREQUENCY (1 TO 255) ";F:
  INPUT "DURATION (1 TO 255) ";D: POKE 768,
  F: POKE 760,D: CALL 770: PRINT : PRINT
  "ANOTHER TONE (Y/N)?": GET X $: IF X$ =
  "Y" THEN 2

```

WAVE

```

1 DIM A$(80):A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ
  ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNO
  PQRSTUVWXYZ": POKE - 16304,0: POKE -16302
  ,0
2 B=B+1: IF B>35 THEN B=1:A=A+1: FOR I=1
  TO A/9: PRINT " " ;: NEXT I: PRINT A$(B,
  B+10);: GOTO 2

```

SO-LONG

1 CALL - 936

2 NEW : REM LAST YEAR'S LOSER...

THUMP-THUMP

1 HOME : SPEED= 90: PRINT "OH, ARTHUR, TELL
ME YOU LOVE ME." : FOR X = 1 TO 4: FOR Y
= 1 TO 4:S = PEEK (- 16336): NEXT: FOR Y
= 1 TO 123: NEXT : FOR Y = 1 TO 4:S = PEEK
(- 16336): NEXT : FOR Y = 1 TO 345 :
NEXT : NEXT

2 VTAB 5: PRINT "YES, JANET... AS SOON AS
YOU": PRINT "GET OFF OF MY ";: SPEED= 255
:A\$ = "FOOT!": FOR X = 1 TO LEN (A\$):
FLASH : PRINT MID\$ (A\$,X,1); CHR\$ (7);:
NEXT : NORMAL : VTAB 20

APPLE® COMMANDS

A Applesoft I Integer Basic	D DOS 3.3 P ProDOS™	f File Name	A\$ String X Variable	m,n,i,j Integers x,y,z Real numbers
--------------------------------	------------------------	-------------	--------------------------	--

- A I **ABS(x)** Absolute (positive) value of *x*
A I **AND** Logical "and" in an IF statement
D P **APPEND f** Add data to a sequential text file
A I **ASC("A")** ASCII value of a character
A I **ASC(A\$)** ASCII of string's first character
A I **AT** See **DRAW**, **XDRAW**, **HLIN** and **VLIN**.
A **ATN(x)** Arctangent of *x* in radians
I **AUTO n,m** Start auto-line numbering

D P **BLOAD f** Load binary file *f*
D P **BRUN f** Load and Run binary program *f*
D P **BSAVE f,An,Lm** Save data; Address *n*. Length *m*

A I **CALL n** Branch to machine language routine at *n*
P **CAT** Display 40-column ProDOS disk contents
D P **CATALOG** Display disk's contents
D P **CHAIN f** Run file *f* without clearing variables
A **CHR\$(n)** Character whose ASCII value is *n*
A **CLEAR** Clear all variables and strings
D P **CLOSE f** Stop reading or writing a text file
I **CLR** Clear all variables and strings
A I **COLOR=n** Set lo-res color to *n* (0-15)
I **CON** Continue an Integer program
A **CONT** Continue an Applesoft program
A **COS(x)** Cosine of *x* in radians
P **CREATE f** Create a directory file

A **DATA A\$,x,y,z** Strings and values to be READ
A **DEF FN A(X)=f(x)** Define a function
A I **DEL n,m** Delete program lines *n* through *m*
D P **DELETE f** Delete file *f* from a disk
A I **DIM X(n)** Dimension a numerical array
A I **DIM A\$(n)** Dimension a string-array or string
A **DRAW n AT i,j** Draw hi-res shape *n* at *i,j*
I **DSP X** Display variable values and line numbers

A I **END** Stop a program with no message (see **STOP**)
D P **EXEC f** Execute text file *f*
A **EXP(x)** e (2.718289) to the *x*th power

A **FLASH** Set flashing screen output (40-columns)
P **FLUSH** Write buffer to disk without closing file
A **FN** See **DEF FN**
A I **FOR X=n TO m** Let *X*=*n*, *X*=*n*+1... until *X*=*m*
D **FP** Clear memory; switch to Applesoft Basic
P **FRE** Free all available memory (garbage collection)
A **FRE(0)** Amount of free memory available

A **GET A\$** Wait for one-character user input
A **GET X** Wait for one-number user input
A I **GOSUB n** Branch to subroutine at line *n*
I **GOSUB X** Branch to subroutine at line *X*
A I **GOTO n** Branch to line *n*
I **GOTO X** Branch to line *X*
A I **GR** View and clear lo-res page 1

A **HCOLOR=n** Set hi-res color to *n* (0-7)
A **HGR** View and clear upper hi-res page 1
A **HGR2** View and clear full hi-res page 2
A I **HIMEM: n** Set highest memory address available
A I **HLIN n,m AT j** Draw a horizontal lo-res line
A **HOME** Clear text screen to black
A **HPlot i,j** Plot a hi-res point
A **HPlot i,j TO n,m** Draw a hi-res line
A **HTAB n** Position cursor at horizontal position *n*

A I **IF...THEN...** Logical "if" true, "then" execute
A I D P **IN#n** Take input from slot *n*
D **INIT f** Erase and format a disk
A I **INPUT X** (or *A\$*) Wait for user input
I **INPUT "ABC", A\$** (or *X*) Print & wait for input
A **INPUT "ABC"; A\$** (or *X*) Print & wait for input
D **INT** Switch to Integer Basic; clear memory
A **INT(RND(1)*n)** Random integer 0 to *n*-1
A **INT(x)** Integer value of *x*
A **INVERSE** Set black-on-white text output

A **LEFT\$(A\$,n)** Left *n* characters of a string
A I **LEN(A\$)** Number of characters in a string
A I **LET X=Y** Set *X* equal to *Y* ("LET" is optional)
A I **LIST** List program from beginning
A **LIST-n** List to line *n*
A **LIST n-** List from line *n*
A **LIST n-m** List lines *n* through *m*
A I **LIST n,m** List lines *n* through *m*
A I **LOAD** Load a program from tape
D P **LOAD f** Load a file from disk
D P **LOCK f** Protect a file from alteration
A **LOG(x)** Natural logarithm of *x*
A I **LOMEM: n** Set start-of-variables location

I **MAN** Cancel **ALTO**
D **MAXFILES n** Reserve *n* file buffers (1-16)
A **MID\$(A\$,n,m)** *m* characters of *A\$*, starting at *n*
I **A\$(n,m)** Characters *n* through *m* of a string
I **m MOD n** Remainder of *m* divided by *n*
D **MON C,I,O** Display disk functions

A I **NEW** Delete current program from memory
A **NEXT** Define bottom of a FOR-NEXT loop
A I **NEXT X** Define bottom of a FOR-NEXT loop
D **NOMON C,I,O** Cancel **MON**
A **NORMAL** Set normal white-on-black text output
A I **NOT** Logical "not" in an IF statement
A I **NOTRACE** Cancel **TRACE**

A **ON X GOSUB n,m...** GOSUB *X*th line number
A **ON X GOTO n,m** Branch to *X*th line number
A **ONERR GOTO n** Branch to line *n* if error occurs
D P **OPEN f** Begin READ or WRITE of a text file
A I **OR** Logical "or" in an IF statement

A I PDL(n) Value (0-255) of paddle *n* (0-3)
A I PEEK(n) Memory value at location *n*
A I PLOT i,j Plot a lo-res dot
A I POKE n,m Set location *n* to value *m*
A I POP Cancel most recent GOSUB
A POS(0) Horizontal cursor position
D P POSITION f Locate READ or WRITE in file
A I D P PR#n Send output to slot *n*
P PREFIX f Change default directory
A I PRINT Skip a text line
A I PRINT "ABC" Print characters within quotes
A I PRINT X Print value of variable *X*

A READ A\$ Read a DATA string
A READ X Read a DATA value
D P READ f Initiate reading a disk text file
A RECALL X Retrieve array from tape
A I REM Programmer's remark follows
D P RENAME f1,f2 Rename a file on disk
A RESTORE Set pointer to first DATA element
P RESTORE f Retrieve strings & variables from file *f*
A RESUME Continue program where error occurred
A I RETURN Branch back to statement after GOSUB
A RIGHT\$(A\$,n) Last *n* characters of a string
A RND(0) Repeat last random number
A RND(1) Random number (0 to 0.999999999)
I RND(n)+1 Random integer between 1 and *n*
A ROT=n Set rotation of a shape to *n* (0-64)
A I RUN Execute program from beginning
A I RUN n Execute program from line *n*
D P RUN f Load and execute program from disk

A I SAVE Save program to tape
D P SAVE f Save program to disk
A SCALE=n Set scale of DRAW or XDRAW
A I SCRNI(i,j) Lo-res screen color at point *i,j*
A SGN(X) Sign (+1, -1 or 0) of *X*
A SHLOAD Load shape table from tape
A SIN(x) Sine of *x* in radians
A SPC(n) *n* spaces in a PRINT statement
A SPEED=n Character output rate (0-255)
A SQR(x) Square root of *x*
A I STEP n Increment-size in a FOR-NEXT loop
A STOP Halt program and print line number
P STORE f Store variables & strings as file *f*
A STORE X Store array on tape
A STR\$(x) String equivalent of value *x*

A TAB(n) Position cursor in a PRINT statement

I TAB n Position cursor at horizontal position *n*
A TAN(x) Tangent of *x* in radians
A I TEXT Switch to text mode; cancel windows
A I THEN Logical "then" in an IF statement
A I TO See FOR and HPLOT.
A I TRACE Print line numbers as executed

D P UNLOCK f Cancel LOCK
A USR(x) Pass *x* to a machine subroutine

A VAL(A\$) Numeric value of a string
D P VERIFY f Verify a file on disk
A I VLIN n,m AT i Draw a vertical lo-res line
A I VTAB n Move cursor to text line *n*

A WAIT i,j,k Insert conditional pause
D P WRITE f Initiate writing to a disk text file

A XDRAW n AT i,j DRAW in opposite color
A XPLOT (Unused Applesoft reserved word)

I # Not equal to
P -f Execute file *f*, regardless of type
A ? Same as PRINT

CONTROL AND ESCAPE COMMANDS:

A I control-C Stop an Applesoft or Integer program
D P control-D CHR\$(4)—DOS command character
A I control-G Beep the speaker
A I control-H Left-arrow (backspace)
A I control-I Tab (Ile/Ilc software-controlled)
A I control-J Down-arrow (line feed)
A I control-K Up-arrow (Ile/Ilc software-controlled)
A I control-M Carriage return
A I control-U Right-arrow (forward space)
A I control-X Cancel line being typed
A I control-[Escape (see esc commands below)

A I esc-@ Clear text screen; leave no prompt
A I esc-A Move cursor one space right
A I esc-B Move cursor one space left
A I esc-C Move cursor one space down
A I esc-D Move cursor one space up
A I esc-↑ (or esc-I) Move cursor up; recursive
A I esc-← (or esc-J) Move cursor left; recursive
A I esc-→ (or esc-K) Move cursor right; recursive
A I esc-↓ (or esc-M) Move cursor down; recursive



Beagle Bros
 Micro Software Inc.

Peeks, Pokes and Pointers

Apple® Zero-Page

DECIMAL HEX

32 **Text Window Left-Edge** (0-39 / normal is 0) \$20

Example: POKE 32, X freezes the left X columns of text.
Warning: Don't let PEEK(32)+PEEK(33) exceed the screen width

33 **Text Window Width** (1-40 or 1-80 / normal is 40 or 80) \$21

Note: POKE 33,33 scrunches listings to remove extra spaces.

34 **Text Window Top-Edge** (0-23 / normal is 0) \$22

35 **Text Window Bottom** (1-24 / normal is 24) \$23

36 **Horizontal Cursor-Position** (0-39) \$24

Examples: If PEEK(36)=X, then the cursor is in column X+1.
POKE 36,X puts the cursor in column X+1 (useful with 80-columns, for positioning the cursor beyond the 40-column limit of HTAB).
Note: POKE 1403,X works similarly—and more predictably.

37 **Vertical Cursor-Position** (0-23) \$25

Examples: If PEEK(37)=Y, then the cursor is on text line Y+1.

43 **Boot Slot *16** (after boot) \$2B

44 **Lo-Res Line End-Point** \$2C

48 **Lo-Res COLOR *17** \$30

50 **Text Output Format** \$32

POKE 50, 63=INVERSE. POKE 50, 255=NORMAL.
POKE 50, 127=FLASH (for ASCII 64-95).

51 **Prompt-Character** \$33

Note: POKE 51,X GOTO line# will prevent a false "Not Direct Command" message caused by an immediate GOTO line# command.

78-79 **Random-Number Field** \$4E.4F

103-104 **Start of Applesoft Program** \$67.68

To Load a program at a non-standard location LOC—
POKE LOC-1, 0; POKE 103, LOC-INT(LOC/256)*256;
POKE 104, INT(LOC/256) Then LOAD PROGRAM.
Note: FP (DOS 3.3 only) sets start-of-program to normal 2049 (\$801).

105-106 **LOMEM** \$69.6A

Note: LOMEM is the Start of Variable-Space, equivalent to End-of-Program (approx.) unless changed with the LOMEM command.

107-108 **Start of Array-Space** \$6B.6C

109-110 **End of Array-Space** \$6D.6E

111-112 **Start of String-Storage** \$6F.70

115-116 **HIMEM** \$73.74

Note: HIMEM-1 is the highest address available for use by an Applesoft program. May be changed with the HIMEM command.

117-118 **Line-Number Being Executed** \$75.76

119-120 **Line-No. Where Program Stopped** \$77.78

121-122 **Address of Line Executing** \$79.7A

123-124 **Current DATA Line-Number** \$7B.7C

125-126 **Next DATA Address** \$7D.7E

127-128 **INPUT or DATA Address** \$7F.80

129-130 **Last-Used Variable Name** \$81.82

131-132 **Last-Used-Variable Address** \$83.84

175-176 **End of Applesoft Program** \$AF.B0

214 **RUN Flag** \$D6

Example: POKE 214, 255 makes any command RUN a program.

216 **ONERR Flag** \$D8

Example: POKE 216, 0 cancels the ONERR function.

218-219 **Line-Number of ONERR Error** \$DA.DB

220-221 **ONERR Error Address** \$DC.DD

222 **ONERR Error Code** \$DE

DOS 3.3 and PRODOS APPLESOFT
1: Language Not Available¹ 0 ?Next Without For
2 or 3: Range Error 16 ?Syntax Error (FP)
3: No Device Connected² 22 ?Return Without Gosub
4: Write-Protected 42 ?Out of Data
5: End of Data 53 ?Illegal Quantity
6: File or Path Not Found 69 ?Overflow
7: Volume Mismatch¹ 77 ?Out of Memory
8: I/O Error 90 ?Undefined Statement
9: Disk Full 107 ?Bad Subscript
10: File Locked 120 ?Redim'd Array
11: Syntax Error¹ or Invalid Option² 133 ?Division by Zero
12: No Buffers Available 163 ?Type Mismatch

13: File Type Mismatch 176 ?String Too Long
14: Program Too Large 191: ?Formula Too Complex
15: Not Direct Command 224: ?Undef'd Function
17: Directory Full² 254: ?Re-Enter
18: File Not Open² 255: (control-C interrupt)
19: Duplicate File Name²
20: File Busy² ¹DOS 3.3 only
21: File(s) Still Open² ²ProDOS only

224-225 **X of Last HPLLOT** (0-279) \$E0.E1

226 **Y of Last HPLLOT** (0-191) \$E2

228 **HCOLOR Code** \$E4

0=0, 42=1, 65=2, 127=3, 128=4, 170=5, 213=6, 255=7

230 **Hi-Res Plotting Page** \$E6

POKE 230,32 selects Page 1. POKE 230,96 selects Page 3.
POKE 230,64 selects Page 2.

231 **SCALE** \$E7

Note: SCALE=0 is equivalent to a SCALE of 256.

232-233 **Shape Table Start Address** \$E8.E9

234 **Hi-Res Collision-Check** \$EA
Example: XDRAW a shape. If PEEK(234)=0 then the shape started at a non-black hi-res point.

241 **SPEED** \$F1

Note: PEEK(241) is 256 minus the current SPEED.

243 **FLASH Mask** \$F3

249 **ROT** \$F9

Display Switches

DECIMAL (with negative equivalent) HEX

49232 (-16304) **Graphics** \$C050

49233 (-16303) **Text** \$C051

49234 (-16302) **Full-Graphics** \$C052

49235 (-16301) **Split-Screen** \$C053

49236 (-16300) **Page One** \$C054

49237 (-16299) **Page Two** \$C055

49238 (-16298) **Lo-Res** \$C056

49239 (-16297) **Hi-Res** \$C057

Note: Activate display switches by Poking each location.
Example: POKE 49232,0 switches to Graphics display

Keyboard, etc.

DECIMAL (with negative equivalent) HEX

49152 (-16384) **Read Keyboard** \$C000

49168 (-16368) **Clear Keyboard** \$C010

Example: 10 KEY=PEEK(49152); IF KEY<128 THEN 10
20 POKE 49168, 0
30 PRINT "KEY: "; CHR\$(KEY-128)

49200 (-16336) **Click Speaker** \$C030

Example: FOR A=1 TO 99 BUZZ=PEEK(49200) NEXT

49249 (-16287) **Button #0** \$C061

Paddle-0 Button or Open (left) Apple key *

49250 (-16286) **Button #1** \$C062

Paddle-1 Button or Closed (right) Apple key *

49251 (-16285) **Button #2** \$C063

*Example: If PEEK(49249+P) is greater than 127, then Paddle Button #P is being pressed—or it's not connected

DOS 3.3 Pokes

(assume DOS loaded in main memory)

POKE 40193, PEEK(40193)-N: CALL 42964

Moves DOS buffers down N*256 bytes.

POKE 44452, N+1: POKE 44605, N

Allows N file names before catalog pause.

POKE 44460,88; POKE 44461,252
Clears screen before catalog.
POKE 44505,234; POKE 44506,234
Exposes deleted file names in catalog.
POKE 44596, 234; POKE 44597, 234; POKE 44598, 234 Cancels catalog pause.
POKE 49107,234; POKE 49108,234; POKE 49109, 234 Prevents language card reload.
POKE 49384,0 Stops drive motor.
POKE 49385,0 Starts drive motor.

Notes

Apple's main memory consists of 65,536 bytes, numbered zero to 65535. Every byte has a value in the range 0-255

- You may Peek (look at) the value in byte number-B with the command— **PRINT PEEK(B)**
- You can usually Poke a new value-V into byte-B with the command— **POKE B,V**

Values higher than 255 must be stored in two bytes.

- To look at the value in consecutive bytes B1-B2— **PRINT PEEK(B1)+PEEK(B2)*256**
- To Poke a new value V (0-65535) into bytes B1-B2— **POKE B1, V-INT(V/256)*256** and **POKE B2, INT(V/256)**

Note: Since almost any memory location can be Peeked or Poked, program listings can reveal thousands of Peeks and Pokes not listed on this chart. Pokes are often used to write machine-language routines that may be activated with the CALL command—the possibilities are infinite.

Let A=PEEK(84435) and B=PEEK(84448).
If A=6 and B=0 then Apple IIc.
If A=6 and (B>223 AND B<240) then Apple IIe.
If A<>6 then Apple II or IIx.

Page-3 DOS Vectors

DECIMAL	HEX
976-978 Re-enter-DOS Vector	\$3D0.3D2
1010-1012 Reset Vector	\$3F2.3F4
Example: POKE 1012, 0 makes Reset boot. (POKE 1012,56 to restore normal Reset function.)	
1013-1015 Ampersand Vector	\$3F5.3F7
Examples: POKE 1014, 165; POKE 1015, 214 makes "&" LIST. POKE 1014, 110; POKE 1015, 165 makes "&" CATALOG. POKE 1014, 18; POKE 1015, 217 makes "&" RUN.	
1016-1018 Control-Y Vector	\$3F8.3FA

DOS 3.3 Locations

DECIMAL	HEX
42350 Catalog-Routine	\$A56E
Example: CALL 42350 catalogs a disk.	
40514 Greeting Program Run-Flag	\$9E42
POKE 40514,52 and INIT a disk. When booted, DOS will attempt to BRUN the greeting program. POKE 40514,20 for EXEC	
43140-43271 Commands	\$A884.A907
43378-43582 Error Messages	\$A972.AA3E
43616-43617 Last Blood Length	\$AA60.AA61
43634-43635 Last Blood Start	\$AA72.AA73
43624 Drive-Number	\$AA68
Example: POKE 43624, D changes disk input/output to Drive D.	
43626 Slot-Number	\$AA6A
Example: POKE 43626, S changes disk input/output to Slot S	

43698 **Control-D Command Character** \$AAB2 || 44033 **Catalog Track Number** | \$AC01 |
45991-45998 **File-Type Codes**	\$B3A7.B3AE
45999-46010 **Disk Volume Heading**	\$B3AF.B3BA
46017 **Disk Volume Number**	\$B3C1

ProDOS™ Locations

DECIMAL	HEX
48944 Slot/Drive Value	\$BF30
If PEEK(48944) is greater than 127 then Drive 2, otherwise Drive 1.	
47313-47422 Commands	\$B8D1.B93E
48840-48841 Last Blood Length	\$BEC8.BEC9
48825-48826 Last Blood Start	\$BEB9.BEBA

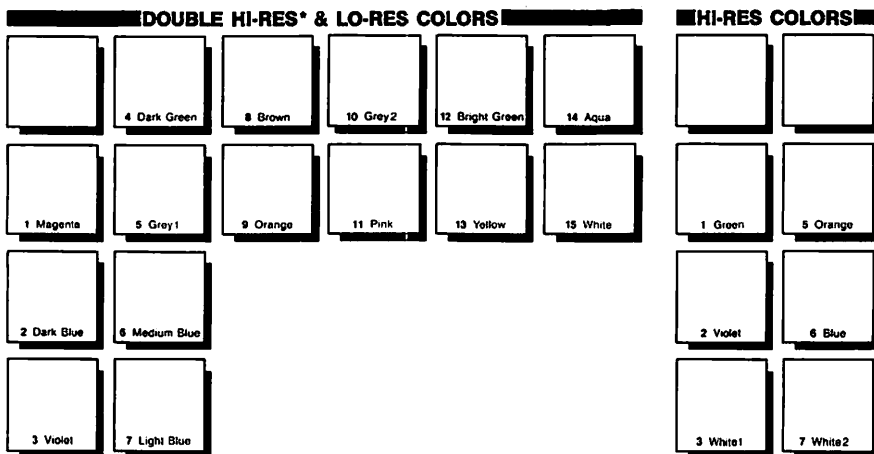
Useful Calls

DECIMAL (add 65536 for positive equivalent)	HEX
CALL-25153 Reconnect DOS 3.3	\$9DBF
CALL-3100 Reveal hi-res page 1	\$F3E4
CALL-3086 Clear hi-res screen to black	\$F3F2
CALL-3082 Clear hi-res to last color Hplotted	\$F3F6
Example: HGR2: HCOLOR=5: HPLLOT 0,0: CALL-3082	
CALL-2613 Hi-res coordinates to Zero-Page	\$F5CB
Example: The X and Y starting coordinates of the next shape table DRAW or XDRAW may be determined with a CALL-2613. Then X=PEEK(224)+PEEK(225)*256 and Y=PEEK(226).	
CALL-1438 Pseudo-Reset	\$FA62
CALL-1370 Boot	\$FAA6
CALL-1321 Display all registers	\$FAD7
CALL-1184 Clear screen and print "Apple..."	\$FB60
CALL-1036 Move cursor right	\$FBF4
CALL-1008 Move cursor left	\$FC10
CALL-998 Move cursor up	\$FC1A
CALL-958 Clear text from cursor to bottom	\$FC42
CALL-922 Move cursor down	\$FC66
CALL-868 Clear text-line from cursor to right	\$FC9C
CALL-756 Wait for any keypress	\$FD0C
CALL-678 Wait for a Return keypress	\$FD5A
CALL-657 Better Input: commas/colons o.k.	\$FD6F
10 PRINT "NAME (LAST, FIRST)";: CALL -657 20 AS="": FOR X=512 TO 767: IF PEEK(X)<>141 THEN AS=AS+CHR\$(PEEK(X)-128): NEXT X	
CALL-468 Memory move	\$FE2C
A Basic memory move: OS & OE are the Old-location Start & End, and NS is the New Start. GOSUB 5000 to execute the move— 5000 N=OS: LOC=60: GOSUB 5020: N=OE: LOC=62: GOSUB 5020: N=NS: LOC=66: GOSUB 5020: 5010 POKE 768, 160: POKE 769, 0: POKE 770, 76: POKE 771, 44: POKE 772, 254: CALL 768: RETURN 5020 POKE LOC, N-INT(N/256)*256: POKE LOC+1, INT(N/256): RETURN	
CALL-415 Disassembler	\$FE61
Note: Poke start address at locations 58-69 before Call.	
CALL-211 Ring bell and print "ERR"	\$FF2D
CALL-198 Ring bell	\$FF3A
CALL-151 Enter monitor	\$FF69
CALL-144 Scan input buffer	\$FF70
This example uses CALL -144 to execute a machine language routine from Basic (will not work in a subroutine): 100 AS="300: A9 C1 20 ED FD 18 89 01 C9 DB D0 F6 60 300G D823G" 110 FOR X=1 TO LEN(AS): POKE 511+X, ASC(MID\$(AS,X,1))+128: NEXT 120 POKE 72, 0: CALL -144	



Beagle Bros
Micro Software Inc.

APPLE® COLORS



(Colors may vary. Try adjusting your monitor.)

(Colors may vary. Try adjusting your monitor.)

To get on a really good mailing list, write
BEAGLE BROS INC.
3590 Old Town Avenue, Suite 102C
San Diego, California 92110

COPYRIGHT © 1984, BERT KERSEY, BEAGLE BROS INC.

*16-color, 560x192 Double Hi-Res
graphics may be created on 128K
Apples with Mark Simonson's
BEAGLE GRAPHICS*, now available at
your local Apple software store.

*APPLE is a Registered Trade Mark of Apple Computer, Inc.

ASCII VALUES

Low	High	Low	High	Low	High	Low	High	Mouse Characters	Main Memory Map
0 \$00 ctrl-@ 128 \$80	32 \$20 sp 160 \$40	64 \$40 @ 192 \$C0	96 \$60 * 224 \$E0	128 \$80 (256 \$F0	160 \$A0 P 288 \$D0	192 \$80 p 320 \$F0	224 \$60 P 352 \$D0	256 \$80 P 384 \$D0	4096 \$0000
1 \$01 ctrl-A 129 \$81	33 \$21 ! 161 \$41	65 \$41 A 193 \$C1	97 \$61 # 225 \$E1	129 \$81 (257 \$F1	161 \$A1 Q 289 \$D1	193 \$81 q 321 \$F1	225 \$61 P 353 \$D1	257 \$80 P 385 \$D0	4100 \$0000
2 \$02 ctrl-B 130 \$82	34 \$22 " 162 \$42	66 \$42 B 194 \$C2	98 \$62 \$ 226 \$E2	130 \$82 (258 \$F2	162 \$A1 R 290 \$D2	194 \$80 r 322 \$F2	226 \$62 P 354 \$D2	258 \$80 P 386 \$D0	4104 \$0000
3 \$03 ctrl-C 131 \$83	35 \$23 # 163 \$43	67 \$43 C 195 \$C3	99 \$63 c 227 \$E3	131 \$83 (259 \$F3	163 \$A1 S 291 \$D3	195 \$80 s 323 \$F3	227 \$63 P 355 \$D3	259 \$80 P 387 \$D0	4108 \$0000
4 \$04 ctrl-D 132 \$84	36 \$24 \$ 164 \$44	68 \$44 D 196 \$C4	100 \$64 d 228 \$E4	132 \$84 (260 \$F4	164 \$A1 T 292 \$D4	196 \$80 t 324 \$F4	228 \$64 P 356 \$D4	260 \$80 P 388 \$D0	4112 \$0000
5 \$05 ctrl-E 133 \$85	37 \$25 % 165 \$45	69 \$45 E 197 \$C5	101 \$65 e 229 \$E5	133 \$85 (261 \$F5	165 \$A1 U 293 \$D5	197 \$80 u 325 \$F5	229 \$65 P 357 \$D5	261 \$80 P 389 \$D0	4116 \$0000
6 \$06 ctrl-F 134 \$86	38 \$26 & 166 \$46	70 \$46 F 198 \$C6	102 \$66 f 230 \$E6	134 \$86 (262 \$F6	166 \$A1 V 294 \$D6	198 \$80 v 326 \$F6	230 \$66 P 358 \$D6	262 \$80 P 390 \$D0	4120 \$0000
7 \$07 ctrl-G 135 \$87	39 \$27 ' 167 \$47	71 \$47 G 199 \$C7	103 \$67 g 231 \$E7	135 \$87 (263 \$F7	167 \$A1 W 295 \$D7	199 \$80 w 327 \$F7	231 \$67 P 359 \$D7	263 \$80 P 391 \$D0	4124 \$0000
8 \$08 ctrl-H 136 \$88	40 \$28 (168 \$48	72 \$48 H 200 \$C8	104 \$68 h 232 \$E8	136 \$88 (264 \$F8	168 \$A1 X 296 \$D8	200 \$80 x 328 \$F8	232 \$68 P 360 \$D8	264 \$80 P 392 \$D0	4128 \$0000
9 \$09 ctrl-I 137 \$89	41 \$29) 169 \$49	73 \$49 I 201 \$C9	105 \$68 i 233 \$E9	137 \$89 (265 \$F9	169 \$A1 Y 297 \$D9	201 \$80 y 329 \$F9	233 \$69 P 361 \$D9	265 \$80 P 393 \$D0	4132 \$0000
10 \$0A ctrl-J 138 \$8A	42 \$2A * 170 \$AA	74 \$4A J 202 \$CA	106 \$6A j 234 \$EA	138 \$8A (266 \$FA	170 \$A1 Z 298 \$DA	202 \$80 z 330 \$FA	234 \$6A P 362 \$DA	266 \$80 P 394 \$D0	4136 \$0000
11 \$0B ctrl-K 139 \$8B	43 \$2B + 171 \$AB	75 \$4B K 203 \$CB	107 \$6B k 235 \$EB	139 \$8B (267 \$FB	171 \$A1 [299 \$DB	203 \$80 [331 \$FB	235 \$6B P 363 \$DB	267 \$80 P 395 \$D0	4140 \$0000
12 \$0C ctrl-L 140 \$8C	44 \$2C , 172 \$AC	76 \$4C L 204 \$CC	108 \$6C l 236 \$EC	140 \$8C (268 \$FC	172 \$A1 \ 300 \$DC	204 \$80 \ 332 \$FC	236 \$6C P 364 \$DC	268 \$80 P 396 \$D0	4144 \$0000
13 \$0D ctrl-M 141 \$8D	45 \$2D - 173 \$AD	77 \$4D M 205 \$CD	109 \$6D m 237 \$ED	141 \$8D (269 \$FD	173 \$A1 ^ 301 \$DD	205 \$80 ^ 333 \$FD	237 \$6D P 365 \$DD	269 \$80 P 397 \$D0	4148 \$0000
14 \$0E ctrl-N 142 \$8E	46 \$2E . 174 \$AE	78 \$4E N 206 \$CE	110 \$6E n 238 \$EE	142 \$8E (270 \$FE	174 \$A1 _ 302 \$DE	206 \$80 _ 334 \$FE	238 \$6E P 366 \$DE	270 \$80 P 398 \$D0	4152 \$0000
15 \$0F ctrl-O 143 \$8F	47 \$2F / 175 \$AF	79 \$4F O 207 \$CF	111 \$6F o 239 \$EF	143 \$8F (271 \$FF	175 \$A1 ` 303 \$E	207 \$80 ` 335 \$E	239 \$6F P 367 \$E	271 \$80 P 399 \$D0	4156 \$0000
16 \$10 ctrl-P 144 \$90	48 \$30 0 176 \$B0	80 \$50 P 208 \$D0	112 \$70 p 240 \$F0	144 \$90 (272 \$FF	176 \$A1 [304 \$F	208 \$80 [336 \$F	240 \$70 P 368 \$F	272 \$80 P 400 \$D0	4160 \$0000
17 \$11 ctrl-Q 145 \$91	49 \$31 ! 177 \$B1	81 \$51 Q 209 \$D1	113 \$71 q 241 \$F1	145 \$91 (273 \$FF	177 \$A1 \ 305 \$F	209 \$80 \ 337 \$F	241 \$71 P 369 \$F	273 \$80 P 401 \$D0	4164 \$0000
18 \$12 ctrl-R 146 \$92	50 \$32 " 178 \$B2	82 \$52 R 210 \$D2	114 \$72 r 242 \$F2	146 \$92 (274 \$FF	178 \$A1 ^ 306 \$F	210 \$80 ^ 338 \$F	242 \$72 P 370 \$F	274 \$80 P 402 \$D0	4168 \$0000
19 \$13 ctrl-S 147 \$93	51 \$33 # 179 \$B3	83 \$53 S 211 \$D3	115 \$73 s 243 \$F3	147 \$93 (275 \$FF	179 \$A1 _ 307 \$F	211 \$80 _ 339 \$F	243 \$73 P 371 \$F	275 \$80 P 403 \$D0	4172 \$0000
20 \$14 ctrl-T 148 \$94	52 \$34 \$ 180 \$B4	84 \$54 T 212 \$D4	116 \$74 t 244 \$F4	148 \$94 (276 \$FF	180 \$A1 [308 \$F	212 \$80 [340 \$F	244 \$74 P 372 \$F	276 \$80 P 404 \$D0	4176 \$0000
21 \$15 ctrl-U 149 \$95	53 \$35 % 181 \$B5	85 \$55 U 213 \$D5	117 \$75 u 245 \$F5	149 \$95 (277 \$FF	181 \$A1 \ 309 \$F	213 \$80 \ 341 \$F	245 \$75 P 373 \$F	277 \$80 P 405 \$D0	4180 \$0000
22 \$16 ctrl-V 150 \$96	54 \$36 & 182 \$B6	86 \$56 V 214 \$D6	118 \$76 v 246 \$F6	150 \$96 (278 \$FF	182 \$A1 ^ 310 \$F	214 \$80 ^ 342 \$F	246 \$76 P 374 \$F	278 \$80 P 406 \$D0	4184 \$0000
23 \$17 ctrl-W 151 \$97	55 \$37 ' 183 \$B7	87 \$57 W 215 \$D7	119 \$77 w 247 \$F7	151 \$97 (279 \$FF	183 \$A1 _ 311 \$F	215 \$80 _ 343 \$F	247 \$77 P 375 \$F	279 \$80 P 407 \$D0	4188 \$0000
24 \$18 ctrl-X 152 \$98	56 \$38 (184 \$B8	88 \$58 X 216 \$D8	120 \$78 x 248 \$F8	152 \$98 (280 \$FF	184 \$A1 [312 \$F	216 \$80 [344 \$F	248 \$78 P 376 \$F	280 \$80 P 408 \$D0	4192 \$0000
25 \$19 ctrl-Y 153 \$99	57 \$39) 185 \$B9	89 \$59 Y 217 \$D9	121 \$79 y 249 \$F9	153 \$99 (281 \$FF	185 \$A1 \ 313 \$F	217 \$80 \ 345 \$F	249 \$79 P 377 \$F	281 \$80 P 409 \$D0	4196 \$0000
26 \$1A ctrl-Z 154 \$9A	58 \$3A * 186 \$BA	90 \$5A Z 218 \$DA	122 \$7A z 250 \$FA	154 \$9A (282 \$FF	186 \$A1 ^ 314 \$F	218 \$80 ^ 346 \$F	250 \$7A P 378 \$F	282 \$80 P 410 \$D0	4200 \$0000
27 \$1B ctrl-[155 \$9B	59 \$3B + 187 \$BB	91 \$5B [219 \$DB	123 \$7B { 251 \$FB	155 \$9B (283 \$FF	187 \$A1 _ 315 \$F	219 \$80 _ 347 \$F	251 \$7B P 379 \$F	283 \$80 P 411 \$D0	4204 \$0000
28 \$1C ctrl-\ 156 \$9C	60 \$3C , 188 \$BC	92 \$5C \ 220 \$DC	124 \$7C { 252 \$FC	156 \$9C (284 \$FF	188 \$A1 ^ 316 \$F	220 \$80 ^ 348 \$F	252 \$7C P 380 \$F	284 \$80 P 412 \$D0	4208 \$0000
29 \$1D ctrl-] 157 \$9D	61 \$3D - 189 \$BD	93 \$5D] 221 \$DD	125 \$7D } 253 \$FD	157 \$9D (285 \$FF	189 \$A1 _ 317 \$F	221 \$80 _ 349 \$F	253 \$7D P 381 \$F	285 \$80 P 413 \$D0	4212 \$0000
30 \$1E ctrl-^ 158 \$9E	62 \$3E . 190 \$BE	94 \$5E ^ 222 \$DE	126 \$7E ~ 254 \$FE	158 \$9E (286 \$FF	190 \$A1 [318 \$F	222 \$80 [350 \$F	254 \$7E P 382 \$F	286 \$80 P 414 \$D0	4216 \$0000
31 \$1F ctrl-_ 159 \$9F	63 \$3F / 191 \$BF	95 \$5F _ 223 \$DF	127 \$7F ~ 255 \$FF	159 \$9F (287 \$FF	191 \$A1 \ 319 \$F	223 \$80 \ 351 \$F	255 \$7F P 383 \$F	287 \$80 P 415 \$D0	4220 \$0000

(Delete)

(Light area designates memory normally available to AppleSoft programs.)

INDEX

- Apple Almanac, The*, 209
Apple Mechanic, 165
Applesoft, 21, 189
 backups, 44
 BASIC vs. Integer, 131–132
 characters, hidden, 48
 CONTROL, 43, 181–182
 cursor, inverse, 87
 DATA statement, improper, 37
 delay loops, 57
 error messages, 50, 51, 92
 error silencer, 50
 Global Program Line Editor, 38, 40–42, 155, 177
 graphics program, 118–119
 hex conversion, 44
 line numbers, 59–60
 memory locations, 62
 MOD program, 47–48
 numbers, 55–56, 65, 69, 95, 96
 parsing, 53, 74–75, 201
 program deletion, 47
 program line deletion, 40
 question mark, 44–45, 65
 quotes, 35–36, 65, 86
 slot search, 53–54
 text window size, 43
 tips, 35–67
 tokens, 26–28, 49, 51–52, 80. *See also specific tokens*
 user options, 43
 vocabulary, 46
- Backscroll, 199
Backslash, 15–16
Bank Street Writer, 160
BASIC
 addresses, 2–3, 28
 binary conversion, 136–137
 Integer, 47, 59, 86, 127–132
Beneath Apple DOS, 29
Binary files, 160
Bracket
 left, 15–16
 right, 16
Byte Zap, 29
Bytes, 29, 30–31, 33, 71, 93, 135, 199–200
- CALL, 44, 141–142, 143, 198, 199

- Cassette player. *See* Tape
- Catalog, 148–152, 169–170, 175–187
- binary files, 160
 - DOS 3.3, 175, 176–179, 180–181
 - file naming, 154–156, 158, 171, 172, 176, 180, 182–184, 186–187
 - formatting, 185–186
 - hyphen command, 170
 - name omission, 179–180
 - ProDOS, 175, 181
 - VERIFY file, 169
- Characters
- chart, 204–205
 - flashing, 128–129
 - hidden, 48
 - inverse 19, 85–86, 88–89, 193, 202–203
 - lowercase equivalents, 90, 155–156
 - See also* Keyboard
- Circles, 144
- Cleaning, 194–195
- Code breaking, 102–105
- Colon, 200
- Colors, 109–111, 115, 117–118, 193
- Columns, number of, 203, 212
- Commands
- hyphen, 170
 - machine language, 137–141
 - multiple, 159
 - printer, 214
 - See also specific commands*
- Commas, avoiding, 158
- Computer types, 188–192, 201
- CONTROL, 43, 82, 179, 180, 181–182
- Cursor
- eliminating, 45–46
 - inverse, 87
- DATA statement
- improper, 37
 - item count, 56
- Data storage
- Byte Zap, 29
 - components, 29–30
- Decimals, 29, 44, 55–56, 65, 142, 199–200
- DEL, 64
- Disk
- booting, 162–165, 171, 200
 - cataloging, 169–170
 - DOS type, 152–155
 - Extra K*, 173
 - inserting, 4–5, 157
 - labeling, 4, 153
 - puzzle program, 205–206
 - RAM, 172–173
 - removing, 166
 - sleeves, 4–5, 7, 8
 - storage, 7, 8, 12, 167, 173
 - strength, 166
 - System Utilities, 152–154
 - two-sided, 8, 168–169
 - See also* DOS Boss; System Master, DOS 3.3
- Disk drive
- head cleaning, 194
 - in-use light, 167
 - labeling, 1
 - multiple, 166
 - placement, 3, 14
 - speed, 196–197
 - squeaky, 195
- Disk Operating System. *See* DOS
- DOS (Disk Operating System), 144–174
- changing, 146–152, 162, 167
 - destroying, 34
 - disconnection, 171
 - disk type, 152–155
 - DOS Boss*, 146–147, 148, 149, 165
 - error messages, 160–161
 - POKES, 17, 34, 147–152
 - ProDOS, 146, 153, 154–155, 160, 170–174, 175, 184, 190
 - program protection, 98–100
 - 3.3 System Master, 152–153, 165
 - writing, 30
- DOS Boss*, 146–147, 148, 149, 165

Drive. *See* Disk drive
Dvorak keyboard, 20

Editing

cursor location, 19
Global Program Line Editor, 38,
40–42, 155, 177
text window, 43
word breaks, 40–41

Electricity. *See* Power, supply

END, 202

Error

messages, 50, 51, 92, 151–152,
160–161, 201, 202
silencer, 50
tokens, 51–52

Extra K, 173

File. *See* Catalog

FLASH, 128–129, 156

FORTH, 48

Games and puzzles, programs for,
205–208, 212

Garbage removal, 25–26, 79

GET, 60–62

Global Program Line Editor (GPLE),
38, 40–42, 155, 177

GOSUB, 200, 201

GOTO, 67, 201

GPLE. *See* *Global Program Line Editor*

Graphics, 109–126, 213

BRUN, 143
circles, 114
colors, 109–111, 115, 117–118
shape tables, 122–125

Hardware maintenance, 193–197

HELLO, deleting, 158

Hex, 29, 31, 44

codes, 140–141
converter, 142, 199–200
equivalents, 32

IF, 68–69

INIT, inverse, 159

INPUT, 57–59

Integer BASIC, 47, 59, 86, 127–132,
188

FLASH, 128–129

mini-assembler, 134–135

printing characters, 129–130

speed, 127–128

variables as line numbers, 131

vs. Applesoft, 131–32

Keyboard

brackets, 15–16

character names, 16–17

directional, 18–19

Dvorak, 20

inverse lowercase, 19

K and D, marking, 13, 16

keyless characters, 15–16

numbers, 15

protection, 9–10

QWERTY, 20

RESET, 10–11, 17–18

Key-Cat, 165

Labeling

disk, 4, 153

disk drive, 1

mouse, 2

paddles, 1–2

“permanent,” 3–4

slot, 1

Languages, 48, 57

binary to BASIC, 136–137

commands, 137–141

entering program, 133–134

See also Applesoft; BASIC; Integer
BASIC

Line locator, 80

LIST, 73–82, 136, 200

alignment, 75

break, 81

page advance, 82

split, 78

See also Catalog; REM statements

- Memory, 21–34
 - addresses, formula for finding, 91
 - bytes, 32–33
 - clearing, 72
 - conserving, 70–72, 93
 - Extra K*, 173
 - locations as negative numbers, 62
 - moving, 21–22, 23, 157
 - RAM, 33, 172–173, 188–189
 - ROM, 33, 188–189
 - screen filler, 22
- Mini-assembler, 134–135
- Minuses, 200
- MOD, 47–48
- Monitor
 - mounting, 9, 13
 - Reset to, 18
 - type, 5
- Mouse, 2
- NEXT, 201
- Nibble*, 154
- Number systems. *See* Decimals; Hex
- ONERR, 50–52, 201
- Paddle
 - graphics setting, 112
 - labeling, 1–2
 - replacing, 2, 3
- Parsing, 53, 74–75, 201
- PEEK, 34, 66, 80, 88–89, 90, 198
- Plug. *See* Power, supply
- POKE, 34, 53, 66, 90, 198, 201, 213
 - DOS, 17, 34, 147–152
 - garbage removal, 25–26
 - inverse characters, 19, 88–89
 - invisible catalog, 178
 - program deletion, 47
 - program protection, 100–102
 - Reset programming, 17
 - tokens, 28
- POP, 62–63
- Power
 - failure, 166
 - supply, 11, 191
- Prefixes, 172
- Printer, 203
 - commands, 214
 - connecting, 201
 - stand, 6
- Program list, stand for, 6–7
- ProDOS, 146, 153, 154–155, 160, 170–174, 175, 184, 190
- Programs
 - binary to BASIC, 136–137
 - Byte Zap*, 29
 - character chart, 204–205
 - circles, 114
 - code breaking, 102–105
 - colors, 117–118
 - cursor, ending without, 45–46
 - deleting lines, 40
 - deletion, 47, 64
 - description, 23–25
 - games and puzzles, 205–208, 212
 - garbage, 25–26, 79
 - graphics, 111–114, 115, 116, 118–126
 - HELLO, 158
 - Integer, 127–130
 - inverse characters, 19, 85–86
 - machine language, 133–134, 141–142
 - Master Create, 162, 165
 - memory move, 21–22, 157
 - MOD, 47–48
 - printing, 47
 - protecting, 10–11, 54, 97–105
 - question mark deletion, 44–45
 - Reset key, 17–18
 - screen filler, 22
 - size, 30
 - sounds, 106–108
 - speed, 69–70
 - token viewer, 28, 49
 - two-line, 216–231
 - See also* Applesoft; DOS; Integer BASIC

- Punctuation. *See specific punctuation marks*
- Puzzles. *See Games and puzzles*
- Question mark, 44–45, 65
- Quotation marks, 35–36, 65, 74–75, 86, 88, 201
- QWERTY keyboard, 20
- Rack, disk, 7
- Random Access Memory (RAM), 33, 172–173, 188–189
- READ, 83–84
- Read Only Memory (ROM), 33, 188–189
- REM statements, 38–40, 71, 73–76, 77, 78, 200, 201
- RESET
- disk access during, 159
 - monitor, return to, 18
 - programming, 17–18
 - protector, 10–11
- ROM. *See Read Only Memory*
- SAVE, 54
- Screen clearing, 200
- Sectors, 30
- Shape tables, 122–125
- Slot, 1, 53–54
- Software
- storage, 12
 - See also Applesoft; BASIC; Integer BASIC*
- Sounds, 106–108
- beep, 107, 143, 151–152
 - chirp, 108
 - click, 106–107, 159
 - metronome, 106
 - moo, 108
- Space
- avoiding, 158
 - erasers, 2
- Speaker
- switch, 12–13
 - volume, 13
- Stands
- printer, 6
 - program list, 6–7
- STOP, 45
- Storage
- disk, 7, 8, 12, 167, 173
 - software, 12
 - See also Data storage*
- System Master, DOS 3.3, 152–153, 165
- Master Create program, 162, 165
- Systems Utilities, 152–154
- Tape, 158–159, 188
- Text
- column position, 89–90
 - help page, 84–85
 - inverse titles, 86–87
 - saving, 92
 - screen format, 89
 - tips, 83–96
 - vertical columns, 83
 - window, 43
- Tokens, 26–28, 49, 51–52, 80
- See also specific tokens*
- Tracks, 29–30
- Underscore, 15–16
- User options, 43
- VERIFY, 169
- Word breaks, 40–41
- Write-protect tabs, 168
- Writing. *See Editing; Text*

ABOUT THE AUTHORS

BERT KERSEY is a graphic artist who bought an Apple II as a toy in 1979. The following year, he founded Beagle Bros, which is now the leading Apple software utility publisher. Bert is the author/co-author of many Apple software products, including *Alpha Plot*, *Apple Mechanic*, *Beagle Bag*, *DOS Boss*, *Pro-Byter*, *Shape Mechanic* and *Utility City*. He was the founder of the *DOSTalk* column in *Softalk Magazine* and co-authored *The DOSTalk Scrapbook* (Tab Books). Some of Bert's best work can be found buried in Beagle Brothers' famous ads, tip books, and catalogs.

BILL SANDERS bought his first Apple in 1980 for simulation games. That lasted until he discovered programming. Under the stuffy title William B. Sanders, Ph.D., Bill has written about 25 books, roughly half of them on computers. *The Elementary Apple*, *The Apple Almanac* and *The Apple Macintosh Primer* are the best known among Apple users. As founder and editor of *The Sandy Apple Press*, Bill introduced "bad taste" to computer programming, a feat many thought impossible. Bill is currently a professor at San Diego State University.

TIPS, TRICKS, AND PROGRAMMING SAVVY

Programming tips and tricks are timesavers that make a programmer's life much easier. **The Big Tip Book for the Apple II Series** contains some of the most popular and useful shortcuts ever designed for all Apple II series computers—tricks and tips which have been developed and used by other users and programmers. The programming strategies in this book have been collected with wit and wisdom by **The Beagle Brothers**, well-known Apple software developers.

While most of the book covers Applesoft programming, this book also provides tips for dealing with hardware, disks and DOS, machine language, and every other facet of the Apple II computers. You'll also discover the useful ***two-liners*** presented in this book, which are the most compact source codes available anywhere for the Apple II computers.

Newcomers and old-timers alike will appreciate the breadth of coverage presented in this book, which includes:

- The Keyboard—Locating and using the missing keys on old Apple machines
- Memory Structure—Moving memory locations for efficiency
- Applesoft Tips—Applying others' programming experiences to tasks
- Protection—Save-protecting your programs
- Graphics—Exploring uncharted colors
- Machine Language—Making sense out of machine code
- Disks and DOS—Getting control of your computer
- Two-Liners—Programming at its best
- All these and much, much more