UNPROTECTED

Backups may be made using standard copying procedures.

COMPATIBLE

Any version of Apple II® DOS 3.3 *and* ProDOS™

· D CODE ·

APPLESOFT® PROGRAM COMPACTOR AND DE-BUGGER
by ALAN BIRD

APPLESOFT COMPACTOR

D CODE squeezes every wasted byte out of your Applesoft programs, to save memory space and increase efficiency. Optionally combine program lines, cut variable names to one or two characters, and remove REM statements.

All Goto's, Gosub's and If-Then's are automatically considered. The number of bytes-saved is printed on the screen.

D CODE also finds program lines that are *unused,* and lets you eliminate them to save even more wasted space.

PROGRAM PROOFREADER

Uncover errors *before* they happen! If you commit a typo while entering a program line, the error is reported on the spot *before* your program is Run.

Quickly scan existing programs too, for hidden errors and potential crashes.

PROGRAM COMPARER

Compare any two files to see if they are the same. Differences are reported so you will know what has been changed.

SUPER-TRACE FEATURES

Stop an Applesoft program anytime you want, and ask to see the most recent line numbers and program statements (1 to 10,000 of them) that have been executed. Great for finding out what makes programs tick (or crash!).

An all-new TRACE function neatly prints program statements and variable values in a sealed-off window at the bottom of the screen. Optionally step through a program line-by-line as you watch your variable values change!

DE-BUGGING BREAKPOINTS

Pre-set breakpoints before you RUN, so a program stops if a specified condition becomes true, or if a certain statement is executed a certain number of times.

Find all occurrences of any string in your programs—in *under 2 seconds!*

GPLE COMPATIBLE

D CODE's de-bugging features are fully-compatible with Beagle's *Global Program Line Editor* and *Double-Take.*

INCLUDES FREE PEEKS & POKES CHART

Registration Card

Mailing this card assures you of being on our list for upcoming issues of the *Beagle Bros Bulletin* as well as any important Update Notices regarding your software. The postage is free, so fill it out now, and MAIL IT TODAY, even if you've registered before.

PRODUCT NAME ____________________
(If you received this product FROM BEAGLE BROS BY MAIL, you are already registered—no need to mail this card.)

YOUR NAME ____________________

ADDRESS ____________________

CITY ____________________

STATE & ZIP ____________________

—optional—

What other Apple-related products would you like to see us produce? ____________________

____

Is lack of copy-protection a factor you consider when buying software? (Y/N): ______

Beagle Bros will continue producing low-priced unprotected software as long as our customers support us by not giving copies of our products away. Are you with us on this? (Y/N): ______

Other Comments: ____________________

____

IMPORTANT! Before you start:

DO NOT BOOT THE D CODE DISK if you want *DOS 3.3* D CODE.
Boot it only if you want *ProDOS* D CODE (see page 4).

Beagle Bros
MICRO SOFTWARE

D CODE

APPLESOFT PROGRAM COMPACTER AND DE-BUGGER

One disk, two versions: DOS 3.3 and ProDOS™
Compatible with any model of Apple II
(ProDOS version requires minimum 64K)

ISBN 0-917085-07-8

Published by
Beagle Bros Micro Software, Inc.
3990 Old Town Avenue
San Diego, California 92110

Table of Contents

About the Disk 2
Summary of D CODE Functions . . . 3
The D CODE Disk Catalog 4
COMPACT Program Packer 6
COMPARE File Comparer 17
D.BUG Program De-Bugger 20
  Fast Finder 22
  Program Checker 24
  Easy Lister 25
  Window Tracer 26
  Dump Tracer 32
  Breakpoints 34
Index 40

About the Disk

D CODE IS (as they say) POWERFUL

D CODE is a collection of three sophisticated machine language programs designed to help the Applesoft programmer work more efficiently. COMPACT makes your programs take up less space. COMPARE analyzes differences between programs. D.BUG adds new commands to your keyboard vocabulary, and makes finding those elusive bugs easier and—who knows?—maybe even fun!

D CODE IS DOS 3.3 AND ProDOS™ COMPATIBLE

All D CODE programs come in two versions—one set for working under DOS 3.3 and another set for ProDOS—both on the same side of the D CODE disk. DON'T BOOT THE D CODE DISK if you want the DOS 3.3 version of D CODE (see page 4 for more info).

BACK UP YOUR DISK

Like all Beagle Bros disks, D CODE is not copy protected, just copy*right*ed. Before you get too deeply involved, make a copy of the D CODE disk, using the copy program that came with your Apple (like DOS 3.3's *COPYA* or ProDOS's *FILER* or the Apple IIc *System Utilities* disk).

At Beagle Bros, we avoid copy-protection so our programs will be more friendly to work with, and therefore more valuable to you, the purchaser. Please don't give copies of our software away. Every illegal copy you see is a vote AGAINST friendly software and FOR copy protection and higher prices. You support us, and we'll support you.

BACK UP YOUR PROGRAMS

Please SAVE your original programs on disk and LOCK them before using it first, under a new name, just in case it doesn't survive the trip. This tip applies mostly to programs that have been altered with COMPACT.

WELCOME, BEGINNERS

D CODE is for the semi-experienced Applesoft programmer. If terms like "DOS", "BRUN" and "program line" don't make sense to you, you may want to do some more reading in your Apple manuals before moving on. Actually, just remembering to periodically make backups of your programs will rescue you from any serious problems. (Also see "Beginner's DOS Note" on page 5.)

D CODE Summary

To COMPACT an Applesoft Program (page 6)

1. Load your program from your disk.
2. Insert the D CODE disk, and type "BRUN COMPACT".
3. Select the features you want.
4. Type "C".

(To use COMPACT again while it's still loaded, type "&".)

To COMPARE Files (page 17)

1. Insert the D CODE disk, and type "BRUN COMPARE".
2. Type the name of two same-type (Applesoft, Text, etc.) files you want to compare.

(To use COMPARE again while it's still loaded, type "&".)

De-Bugging Functions (D.BUG) (page 21)

Insert the D CODE disk, and type "BRUN D.BUG" to install the following new commands (single-letter abbreviations in parentheses):

BREAK lets you set up program breakpoints (page 34).

CHECK (C) proofreads a program (page 24).

DUMP (D) lists the last statements used before a program crash (page 32).

FIND (F) finds all occurrences of a word in a program (page 22).

L lists a program (page 25).

NOTRACE (N) turns off all TRACE functions (page 26).

TRACE (T) turns on the trace and function (page 26).

VARIABLES (V:) sets up variables and expressions to be TRACEd (page 26).

WINDOW (W) lets you pre-adjust the size of the TRACE window (page 26).

SIZE (S) lets you set the buffer size for DUMP (page 32).

ZAP (Z) clears the DUMP buffer (page 32).

The D CODE Catalog

The D CODE disk has two sets of programs on it—one for Apple's older DOS 3.3 and another for the newer ProDOS. Both disk catalogs are nearly identical, with the ProDOS version containing three extra files—PRODOS, BASIC.SYSTEM and FP. Here is a rundown of the main files on the disk:

STARTUP is the greeting program that displays the main disk menu when you boot the disk. Since you can't boot the disk in DOS 3.3 (see next page), you will need to type "RUN STARTUP" to see this program.

TITLE is the machine-language program that does the crazy text title routine at the beginning of STARTUP.

COMPACT (page 6) packs Applesoft programs to the smallest possible size by removing Rem statements, shortening variable names and combining program lines.

LINE.SPLITTER (page 15) lets you chop an Applesoft program line in two, in case COMPACT (above) left you with a line that is too long to edit.

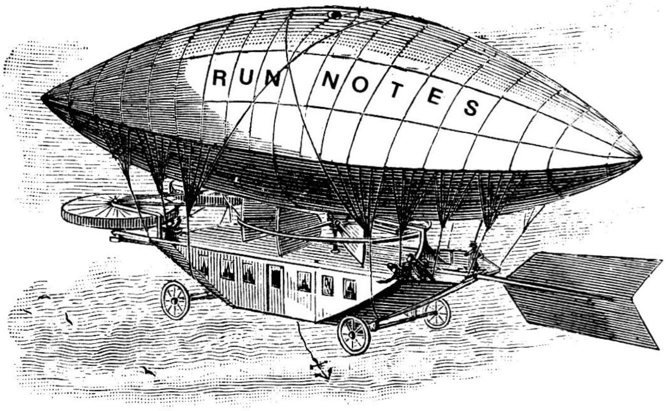
COMPARE (page 17) lets you compare two Applesoft, Binary, or Text files to see if they are identical.

D.BUG (page 20) gives you 11 new commands that proofread programs, trace program lines and variables as they are executed, find words and let you set up breakpoints for de-bugging your programs.

D.BUG.DEMO demonstrates the capabilities of D.BUG.

FP (ProDOS version of D CODE only) will remove the D CODE program (and any Applesoft program) currently in memory. Its function is basically the same as DOS 3.3's FP command. Type "-FP" to execute.

NOTES contains any changes or revelations that have occurred since this manual was printed.



Getting Started

ProDOS D CODE

To use D CODE under ProDOS, you can boot the D CODE disk by inserting it in your main drive and turning on your Apple's power switch. Or, with ProDOS booted, type "PR#6" from the keyboard. Or, if you're using a newer Apple, press CONTROL/OPEN-APPLE/RESET.

Soon, the D CODE menu will appear—just type the number corresponding to the program you want to run or load:

(1) COMPACT
(2) COMPARE
(3) D.BUG
(4) D.BUG DEMO
(5) NOTES (CHANGES TO MANUAL)

If you already have ProDOS booted, you can skip the menu and simply type a *hyphen* followed by the name of the D CODE program you wish to see or use—STARTUP, COMPACT, COMPARE, etc.

DOS 3.3 D CODE

If you are going to use D CODE with DOS 3.3, don't boot the D CODE disk. Instead, boot the DOS 3.3 System Master disk or whatever DOS 3.3 disk you normally would boot. Now insert the D CODE disk, type "RUN STARTUP", and pick the program you want to use from the menu. Or you can type "BRUN" followed by the name of the utility you wish to use, like COMPACT, COMPARE or D.BUG.

BEGINNER'S DOS NOTE

Which Disk Operating System (DOS) you use is up to you; you'll probably want to use the one you are used to. If you don't know which you are using, type "CAT". If you get a disk catalog, you're using ProDOS. If you get a "?Syntax Error" message, you're using DOS 3.3 (or you haven't booted, and therefore aren't using any DOS).

AND WE REPEAT (in case you've been asleep)

Boot the D CODE disk ONLY if you are going to be using ProDOS. To use DOS 3.3 D CODE, boot a DOS 3.3 disk first.

COMPACT

D CODE's COMPACT program allows you to reduce the amount of memory occupied by an Applesoft program, by optionally removing Rem statements, packing as many statements as possible into single program lines, and shortening variable names.

COMPACT-PROGRAM ADVANTAGES

1. A compacted program will occupy less space in memory. If, for example, a program won't quite fit in memory under one of the graphics pages, compacting it might just squeeze it in. Or you may need more room for variables, more subroutines, etc.

2. A compacted program will usually take up less space on disk. A DOS 3.3 disk sector is 256 bytes long; a ProDOS block is 512 bytes. Sometimes an entire sector or block can be saved by eliminating just a couple of bytes.

3. A compacted program will tend to run slightly faster (nothing that will knock your socks off, however).

4. It just plain FEELS GOOD to make a program as small as possible. This feeling, of course, depends on your own personal set of values (and how content your childhood was).

COMPACT-PROGRAM DISADVANTAGES

1. Compacted programs tend to be more difficult for humans to read. More statements per program line is one reason. Abbreviated variable names are another (bigger) reason.

2. Without Rem statements, a program can be difficult to decipher, especially a couple of months down the road.

3. After compacting a program, you may have some program lines that, due to their length, are uneditable, even with GPLE (Beagle Bros' *Global Program Line Editor*), and especially with ancient cursor-tracing methods. The LINE.SPLITTER program (page 15) is a solution that pretty much scratches this problem off the list, however.

PLAY IT SAFE

Because of the disadvantages above, you should always LOCK your *un*compacted program on disk, so you won't accidentally SAVE the compacted version on top of it. Then Save your compacted version on disk under a new name. Your expanded version with its Rem statements and descriptive variable names might come in handy some day. In fact, now that you have the ability to quickly compact your code, you may want to use *more* Remarks and *longer* variable names to help in "pre-release" de-bugging.

How to Use Compact

1. Lock the program you want compacted (type "LOCK PROGRAM-NAME") on disk so you don't accidentally lose it.

2. LOAD the program you want to compact (type "LOAD PROGRAM-NAME").

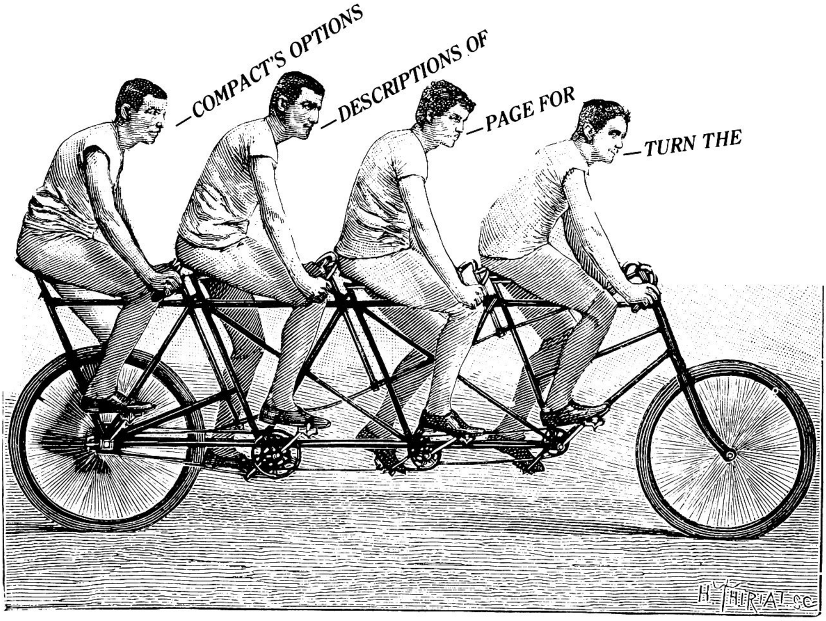
3. BRUN COMPACT (just type "BRUN COMPACT" with the D CODE disk in your drive). If this has recently been done, you can probably just type "&".

D.BUG Note: If you want D.BUG and COMPACT both in memory, you must BRUN D.BUG *first.* Otherwise COMPACT will disappear.

Now you will see COMPACT's menu:

```
(1) REMOVE REMS ............... YES
(2) CONCATENATE LINES.......... YES
(3) SHORTEN VARIABLE NAMES .... YES
(4) RENAME VARIABLES........... YES
(5) COMPACT PART OF PROGRAM .. NO
(6) VARIABLE TABLE TO PRINTER .. NO
(C) COMPACT
(Q) QUIT
```

You have six YES/NO options before compacting a program (the top four always start out as YES). You can toggle each option from YES to NO and back by pressing the number key indicated on the left. When you are ready, press "C" to compact the program. It won't take long.



COMPACT Menu Options

Here is an explanation of each COMPACT menu option:

1. REMOVE REMS

YES means that when you compact your program, every Remark statement will be deleted. If you use a lot of Rems, this option will save more program space than any other, because every character and space in a Remark takes up an entire byte of memory.

2. CONCATENATE LINES (computerese for "Connect Lines".)

YES means that when you compact your program, as many statements as possible will be packed into single program lines, thus eliminating old line numbers. Sample program:

```
    10 TEXT: HOME
    20 GOSUB 50
    30 PRINT "DOGFOOD"
    40 END
    50 PRINT CHR$(7)
    60 RETURN
```

When these lines are concatenated, they look like:

```
    10 TEXT : HOME : GOSUB 50: PRINT "DOGFOOD": END
    50 PRINT CHR$(7): RETURN
```

Every line number that is removed saves you a big four bytes of space. In this example, we saved 16 bytes.

Technical Note: Applesoft needs an "overhead" of 5 bytes per line—2 for the line number, 2 for a pointer to the next line, and 1 for a zero that ends the line. You lose one of the 5 bytes saved for the extra colon that separates the connected statements, leaving you with 4 saved for every line number eliminated.

ULTRA-LONG-LINE PROBLEMS

With option 2 set at YES, COMPACT will often create a program line that works perfectly, but is too long to edit. Applesoft allows program lines of about 250 bytes (each Applesoft word like "PRINT" takes up one byte). Editing, however, has to consider each *character* in the listing (now "PRINT" takes up *five* characters, plus two more for spaces on each end). Even GPLE's "Pack" feature (removes spaces from program lines) won't always let you edit an ultra-long line.

WATCH OUT that you don't inadvertently chop the end off of a program line when you attempt to edit it. If you are cursor-tracing a line and you hear beeping (or if you try to edit with GPLE and code is missing), type CONTROL-X immediately, and BRUN LINE.SPLITTER (page 15).

3. SHORTEN VARIABLE NAMES

When this option is set to YES, all variable names longer than two characters will be shortened to two characters. For example:

Old	*New*
NAMES$	NA$
NUMBER	NU
TABLE(NUM)	TA(NU)
WXYZ%	WX%

In Applesoft, it doesn't matter how long your variable names are, only the first two characters count (COMPACT leaves *only* those two characters if you select YES for this option). The variables APPLE, APPALOOSA and AP are the same. So are MOUSE$, MOUNTEVEREST$ and MO$. The advantage to long variable names is that they are more descriptive. The disadvantage is that they take up a lot of room—one byte per character.

Note: OPTION 3 simply chops the end off of long variable names. OPTION 4 actually renames variables without regard to their former names.

4. RENAME VARIABLES

When this option is selected, COMPACT will change as many two-or-more character variable names into one-letter names as possible. The multiple-character variables that are used *most often* will be converted to single letter names until all 26 letters have been used (for each variable type. Remember, A$, A% and A may all be in the same program). See the sample variable conversion table on page 13.

Note: A (4) YES sets (3) to YES and (5) to NO. A (4) NO sets (6) to NO.

5. COMPACT PART OF PROGRAM

Use this option if you only want a portion of a program compacted. After typing "C" to start compacting, you will be asked for the start and end line numbers for compacting. You may default to the beginning or the end of the program by simply hitting RETURN as an answer to either question.

A (5) YES sets (4) to NO.

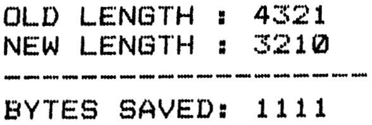
6. VARIABLE TABLE TO PRINTER

When set to YES, this option will print the renamed variables on your slot 1 printer (if option 4 is set to YES). See page 12 for details.

COMPACT options (continued)

C. COMPACT

Just type "C" to compact your program. After a brief pause (depending on the length of your program), a variable conversion table will be displayed if you selected option 4 (sample on page 13). When compacting is completely finished, you will see the former length of the program, the new length, and the amount of memory that was saved, all in bytes.



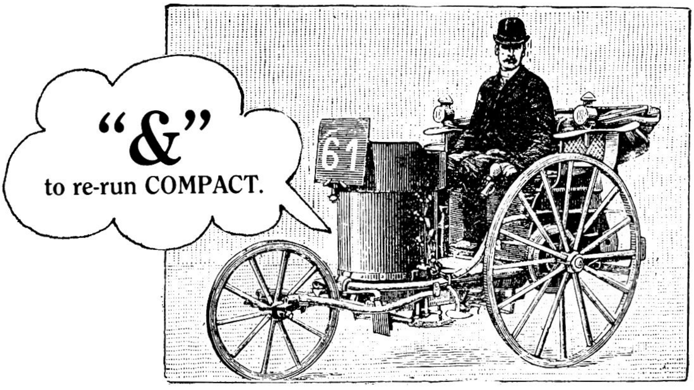
```
OLD LENGTH : 4321
NEW LENGTH : 3210
------------------------
BYTES SAVED: 1111
```

Now that a compacted program has replaced your original program in memory, SAVE it before you RUN it. (Since you've Locked the original uncompacted program on disk, you won't accidentally Save the compacted version under the same name, will you?) You could name your compacted version with a suffix like ".C". Now you will have the original program (like "TESTPROGRAM") and the compacted program (like "TESTPROGRAM.C"), both on the same disk.

If you decide to ignore our advice and Save your compacted version with the same name as your original, there is no way the number of disk sectors will be smaller, unless you Delete the original version from the disk first. (DOS 3.3 only. ProDOS has eliminated this minor bug.)

"&" TO RE-RUN COMPACT

After COMPACT has been used and exited, you will usually be able to bring it up again by typing "&" (return). If this doesn't work, just type "BRUN COMPACT" again.



EXTRA BYTES AT THE END OF A PROGRAM

COMPACT assumes that any extra bytes that it finds imbedded beyond the end of a program is relocatable code that the program uses. If extra bytes are found, you will be asked if you wish to keep them. Answering "Y" will move the code to the new program end. "N" will delete the extra bytes.

If you don't think you have anything beyond the end of the program, and COMPACT asks you about it anyway, just answer "N" and that will be the end of that.

UNUSED STATEMENTS

When programs have undergone heavy revision, statements often remain that can't possibly be executed—your program just won't encounter them. COMPACT will report the line numbers that contain these potentially useless statements. It's up to you to delete them after COMPACT is finished.

UNUSED STATEMENT EXAMPLE:

```
100 PRINT "HELLO" : GOTO 120 : PRINT "GOOD-BYE"
```

Here, "GOOD-BYE" will never be printed because of the GOTO statement immediately before it. You could delete ": PRINT "GOOD-BYE" " and save 11 more bytes. In this case, you would NOT want to delete the entire line.

ANOTHER EXAMPLE:

```
90 PRINT "DOGFOOD": GOTO 110
100 PRINT " IS FOR DOGS."
110 PRINT " IS YUMMY WITH HOT SAUCE."
```

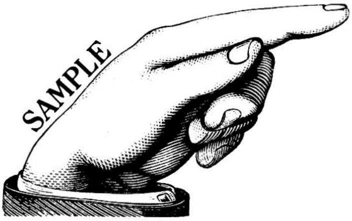
Assuming no other statement in this program goes to line 100, COMPACT will tell you about it. You could delete line 100 (the *entire line* this time) and save about 20 bytes.

INTENTIONAL UNUSED STATEMENTS

We like to use the following save-a-program trick:

```
1 GOTO 10
2 PRINT CHR$(4);"SAVE PROGRAM NAME": END
10 REM PROGRAM STARTS HERE
```

The trick here is that we can simply type "RUN2" anytime we want to Save the program without having to remember its exact name. COMPACT will report line 2 as unused, even though we want to keep it. No problem; just don't delete line 2.



THE VARIABLE CONVERSION TABLE

When RENAME VARIABLES (option 4) is set to YES, a variable conversion table will be displayed on the screen during compaction. This chart lists the name of every variable in the program, it's new name (if it was changed), and the number of times it appears in the program. (Note: Only the first two characters of the variable will appear under the OLD column heading, even though that variable may have had a longer name.).

One-character variables are listed first, unchanged, in the order of their appearance in the program.

Next come the multiple-character variables. COMPACT will shorten as many of these as possible to one character, with the ones that appear most often in the program changed first.

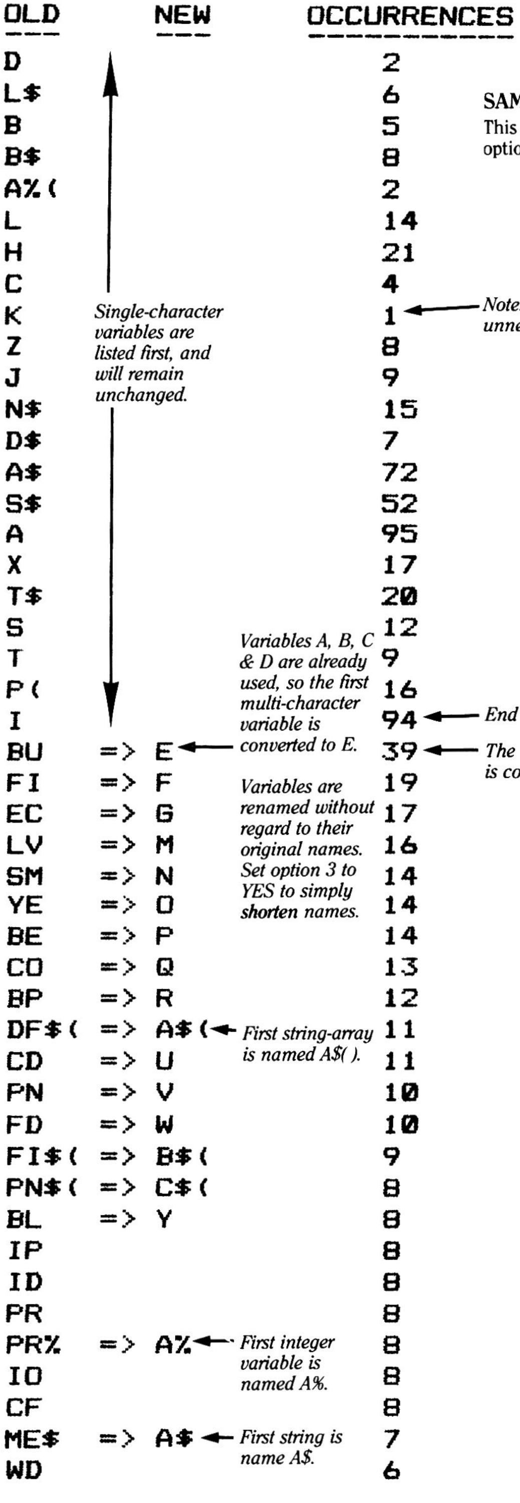
If no name appears in the NEW column, it's because there are no more single characters left for that particular variable type. This will only occur in very large and/or complex programs. Usually real or string variables will be the first to run out of the 26 available single-character names.

LOOK FOR "LONE" VARIABLES

Watch the conversion chart for variables that appear only once or twice in a program. This could indicate that the variable name was misspelled or was once part of a program segment that was removed. You might be able to save additional space (or uncover a potential bug) if you look at the lone variable (use the FIND command in D.BUG, page 23).

SAMPLE VARIABLE CONVERSION TABLE

This table prints on the screen when COMPACT option 4 (Rename Variables) is set to YES.



```
OLD          NEW        OCCURRENCES
---          ---        -----------
D                            2
L$                           6
B                            5
B$                           8
A%(                          2
L                            14
H                            21
C                            4
K                            1
Z                            8
J                            9
N$                           15
D$                           7
A$                           72
S$                           52
A                            95
X                            17
T$                           20
S                            12
T                            9
P(                           16
I                            94
BU     =>  E                 39
FI     =>  F                 19
EC     =>  G                 17
LV     =>  M                 16
SM     =>  N                 14
YE     =>  O                 14
BE     =>  P                 14
CO     =>  Q                 13
BP     =>  R                 12
DF$(   =>  A$(               11
CD     =>  U                 11
PN     =>  V                 10
FD     =>  W                 10
FI$(   =>  B$(               9
PN$(   =>  C$(               8
BL     =>  Y                 8
IP                           8
ID                           8
PR                           8
PR%    =>  A%                8
IO                           8
CF                           8
ME$    =>  A$                7
WD                           6
```

Single-character variables are listed first, and will remain unchanged.

Note: Only 1 occurrence may indicate an unnecessary (or misspelled) variable name.

Variables A, B, C & D are already used, so the first multi-character variable is converted to E.

End of old single-character variable names

The most-often used multi-character variable name is converted first.

Variables are renamed without regard to their original names. Set option 3 to YES to simply shorten names.

First string-array is named A$( ).

First integer variable is named A%.

First string is name A$.

COMPACT TIPS

1. Before using COMPACT, use D.BUG's CHECK command (page 22) to catch any syntax errors in your program.

2. Now that you don't have to worry about Rems eating your program space, use them to more-thoroughly document the development versions of your programs. Use longer variable names too—like COUNTER instead of CTR. Just remember that Applesoft only looks at the first two characters.

3. To look for unused program lines without compacting a program, set all six menu options to NO. (Some minor changes *might* be made to your program.)

4. To save even more space, **RE-NUMBER YOUR PROGRAM BY 1's**, using the Renumber feature from Beagle's *Double-Take* disk (ProDOS or DOS 3.3).

COMPACT ERROR MESSAGES

NO APPLESOFT PROGRAM IN MEMORY

This means what it says. Load your program first, then BRUN COMPACT (or type "&" if COMPACT is already loaded).

MEMORY OVERFLOW. PROGRAM TOO LARGE

This sometimes happens when you have other machine language program(s) in memory in addition to COMPACT. You may need to re-boot or take other measures to clear memory.

RELOAD YOUR BASIC PROGRAM

This message will occasionally come up when you're going to compact a very long Applesoft program. Just re-load the program as instructed, and type "&" to re-run COMPACT.

In case you care: When you first BRUN COMPACT, it's code Bloads at $4000 (16384 decimal), and then relocates itself just under HIMEM. If your loaded program extends above $4000, it will be partially wiped out by COMPACT and you'll be told to re-load.

There's a COMPACT example on page 16.

LINE.SPLITTER

LINE.SPLITTER simply chops a program line into two parts, in case it is too long to edit (see page 8). With your program loaded, simply type "BRUN LINE.SPLITTER" and, when asked, type the number of the line you want to split. LINE.SPLITTER will split the line as near the middle as possible, taking into consideration any IF statements.

The second section of the split line will be numbered one line number higher than the first section.

LINE.SPLITTER ERROR MESSAGES

LINE DOES NOT EXIST

Oops, try again.

RENUMBER FOLLOWING LINE

If the higher number is already taken, the split will be cancelled and you will have to renumber that part of your program to make room. (Use the RENUMBER option from our Double-Take disk, or do it "by hand" by cursor-tracing or by using GPLE's edit feature.)

LINE CAN'T BE SPLIT

The line has only one statement, or its first statement contains an IF.

SAMPLE BEFORE:

```
120  DIM SC(263),FI(255):I = 1:J =
     1:K = 1:ML = 16384:M0 = 1740
     8:MM = 17408:PR = 17409:A =
     1:B = 2:C = 3:D = 4:F0 = 128
     :FF = 255:TF = 256:LOC = 192
     00:Q = - 16384:QQ = Q + 16:
     G$ = CHR$ (7):D$ = CHR$ (4
     ):RT = 16
130  PRINT "MEAT LOAF"
```

SAMPLE AFTER:

```
120  DIM SC(263),FI(255):I = 1:J =
     1:K = 1:ML = 16384:M0 = 1740
     8:MM = 17408:PR = 17409:A =
     1:B = 2:C = 3
121 D = 4:F0 = 128:FF = 255:TF =
     256:LOC = 19200:Q = - 16384
     :QQ = Q + 16:G$ = CHR$ (7):
     D$ = CHR$ (4):RT = 16
130  PRINT "MEAT LOAF"
```

COMPACT Example:

Before:

```
5   ONERR   GOTO 50
8   TEXT
9   HOME
10   READ NUMBER
20 ALPHA$ =   CHR$ (NUMBER)
25   PRINT ALPHA$;
30   GOTO 10
50   END
99   REM DATA
100   DATA 68: REM D
101   DATA  46 : REM .
102   DATA   66: REM B
103   DATA   85: REM U
104   DATA    71: REM  G
105   DATA    32: REM   SPACE
106   DATA   70: REM F
107   DATA   73: REM I
108   DATA   78: REM N
109   DATA  68: REM D
110   DATA   83: REM S
111   DATA  32: REM SPACE
112   DATA  66: REM B
113   DATA   85: REM U
114   DATA    71: REM G
115   DATA   83: REM S
```

Notice how COMPACT removes leading spaces in DATA statements. Trailing spaces not affected.

Note: The READ A$ command ignores leading spaces. You should use quote marks if you want to keep the spaces. (Example: DATA "   TRY", "   THIS")

```
OLD LENGTH :  310
NEW LENGTH :  100
-----------------
BYTES SAVED:  210
```

After:

```
5  ONERR   GOTO 50
8  TEXT : HOME
10  READ A:A$ =  CHR$ (A): PRINT
     A$;: GOTO 10
50  END : DATA 68,46,66,85,71,32
     ,70,73,78,68,83,32,66,85,71,
     83
```

COMPARE

You've probably noticed how you tend to collect different versions of the same program, saved to different disks under the same or different names. D CODE's COMPARE program will take two Applesoft programs and tell you exactly what lines are unique, different, or the same. You can also use COMPARE to check binary and text files to see if they are identical.

JUST BRUN COMPARE AND TYPE TWO FILE NAMES

Select COMPARE From the D CODE startup menu, or simply type:

```
BRUN COMPARE
```

Unfortunately, COMPACT and COMPARE cannot live in the same Apple at the same time—irreconcilable differences. COMPARE and D.BUG get along quite well, however. So do COMPACT and D.BUG.

When the COMPARE screen appears, enter the names of the two files you want to compare. RETURN (with no name) signifies the Applesoft program currently in memory. Under DOS 3.3, type the file name and any DOS parameters that go with it (for example—MYPROGRAM,S6,D2). Under ProDOS, COMPARE will assume the current prefix, unless the full pathname is specified (for example—/MYDISK/MY.SUBDIR/MYPROGRAM).

(continued)

**Programs may look the same, but don't be so sure.
Use COMPARE to find subtle differences.**

COMPARING APPLESOFT FILES

When you're comparing two Applesoft programs, COMPARE will ask you if you want occurrences of identical lines printed. You will usually want to answer "N", since "Y" will often print a ton of meaningless line numbers.

During comparison, a 1, 2, D or S will appear next to the line numbers as they are shown on the screen:

1 means that this line is unique to Program #1 (the first one you selected), and doesn't exist in Program #2.

2 means the line exists in Program #2 and not in #1.

D means both programs have duplicate line numbers, but the contents of the lines are different.

S means the lines are the same. (This will appear only if you answered "Y" to the DISPLAY SAME LINES? option.)

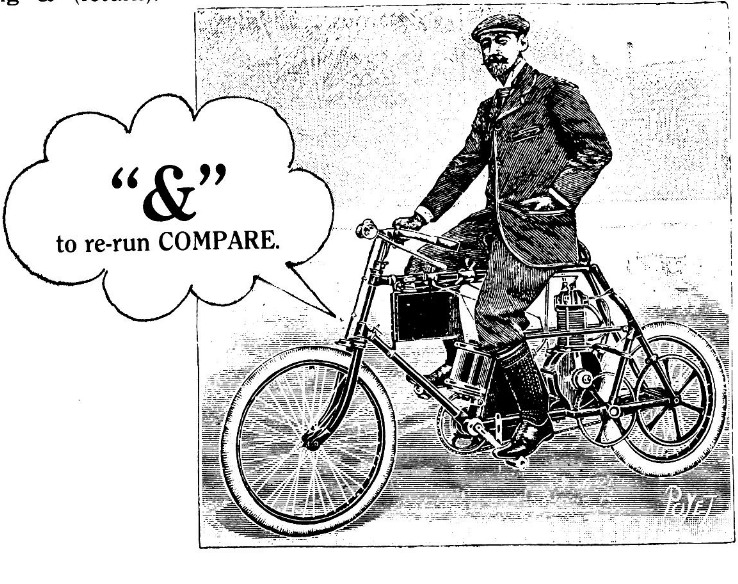
COMPARING BINARY AND TEXT FILES

Since there are no line numbers in binary and text files, COMPARE will just tell you if the files are identical or not.

Binary File Note: If you're using DOS 3.3, COMPARE will also display the starting address and length of both files. (If you're in ProDOS, just type "CATALOG" and take a look there.)

"&" TO RE-COMPARE

Once it's loaded, you may usually re-enter COMPARE and use it again by typing "&" (return).



(19 is blank)

D.BUG

NEW COMMANDS/NEW DE-BUGGING POWER

D.BUG is a powerful machine language utility that will help you de-bug and develop your Applesoft programs. Once D.BUG is loaded into memory, you will have eleven brand new commands to work with (see next page). Before you tackle any serious programming, play around with each of D.BUG's commands and features, and get a feel for what they'll do for you.

All of D.BUG's commands may be typed directly or used inside your programs; just treat them like normal Applesoft commands. Of course you must load (BRUN) D.BUG itself before the new commands will work.

Check out the D.BUG.DEMO program to get a better idea of how D.BUG will make life easier for you. Select the demo from the STARTUP menu, or type "RUN D.BUG.DEMO".

LOADING D.BUG

To load and activate D.BUG's commands, just type:

```
BRUN D.BUG
```

Or select D.BUG from the STARTUP program's menu—see page 4.

If you want D.BUG to co-exist in memory with either COMPACT or COMPARE, D.BUG must be loaded first. Other programs like GPLE, Double-Take, ProntoDOS, and so on, should be loaded *before* D.BUG.

D.BUG may be loaded from within a program in the usual way:

```
10 PRINT: PRINT CHR$(4);"BRUN D.BUG"
```

REMOVING D.BUG

To disable D.BUG and free up the approximately 5½K of memory it occupies, type "FP" (DOS 3.3) or "-FP" (ProDOS—*FP* is a memory-clearing file in the ProDOS D CODE catalog.). This will "erase" D.BUG as well as any Applesoft program in memory. If you're in the habit of typing "FP" instead of "NEW", change your habit now, or you'll lose D.BUG every time.

SINGLE-CHARACTER ABBREVIATIONS

Each D.BUG command may be abbreviated by typing only its *first character* (or characters). For example, the new CHECK command can be abbreviated as C, CH, CHE, or CHEC. To function properly, some of the new commands may or must be followed by other characters or words.

D.BUG FUNCTIONS AND COMMANDS

Fast Finder (page 22)

New Command: **FIND (F)**
Function: Quickly searches through an Applesoft program for occurrences of a specified character or word.

Program Checker (page 24)

New Command: **CHECK (C)**
Function: Quickly proofreads Applesoft programs for syntax and undefined statement errors. In addition, everything you type from the keyboard is proofread automatically (no command required).

Easy Lister (page 25)

New Command: **L**
Function: Saves you three keystrokes every time you list a program, by letting you type "L" instead of "LIST".

Window Tracer (page 26)

New Commands: **TRACE (T), NOTRACE (N), VARIABLES (V) and WINDOW (W)**
Function: Lets you watch Applesoft program line numbers and statements "live" as they are executed (and optionally watch variable and expression values) in an adjustable text window at the bottom of the screen.

Dump Tracer (page 32)

New Commands: **DUMP (D), SIZE (S) and ZAP (Z)**
Function: After a program stops for any reason, you can see the line numbers and statements that were most-recently executed.

Breakpoints (page 34)

New Command: **BREAK (B)**
Function: Lets you set up breakpoints so your program will automatically stop when a variable becomes a certain value, or when other specified conditions are true.

FAST FINDER (BRUN D.BUG to load)

New Command: FIND (F)

With any Applesoft program (and D.BUG) in memory, type "F" (return). You will be asked what you want to "SEARCH FOR:". After you respond, all line numbers containing your character or word will be reported. If your word appears in a line, say three times, that line number will appear three times. The total number of occurrences will be reported after the search ends.

FIND scans your entire program twice, first looking for your string in PRINT, REM and DATA statements, and then in the form of Applesoft words, or *tokens*. When you tell FIND to search for "READ", for example, it reports one set of line numbers for occurrences like PRINT "READ YOUR MAIL." and then another set of numbers for occurrences like READ A$.

Drawbacks: While FIND will successfully find all occurrences of a variable like X, it will *also* throw in all occurrences of variables XX, YX, XY and MAX; strings like "FOX" and "Xebec", and so on. (Note: You can use GPLE's slower global search routine to locate variables only.)

FIND won't find parts of Applesoft keywords, like the GO in GOTO or the TURN in RETURN. For example, it won't find the "HO" in line 10, but it will find it in lines 20 (twice) and 30 (three times).

```
10 TEXT: HOME: NORMAL
20 HO=3: SHO=3: HBO=7
30 PRINT "MELANIE'S OKLAHOMA HOME WAS HOT."
```

WILDC@RD C#ARACTERS

Using "@" in a search word will match any *single character* in a string. For example, SEARCH FOR: D@G will find all occurrences of the words "DOG", "DIGGER", "DAGWOOD", "D3P0G", and so on.

Using "#" in a search word will match *any number of characters.* For example, "D#G" will find all occurrences of the words "DOG", "DOODLING", "D=1: FOR Z=1 TO 10: PRINT G", and "DEAFENING".

FIND AND LIST (FIND L or FL)

Typing "FINDL" or "FL" works like "F" (above), but each occurrence will be Listed, and the search word will be highlighted in inverse.

You will FIND some examples on the following page.

FIND Examples

F
SEARCH FOR: X
(You type "FIND" (return) *or* "F" (return); *"SEARCH FOR:"* appears on the screen; you type "X" (return).)
This will print the line number of every program line that contains the variable X or the character X.

FL
SEARCH FOR: AB
This will find and list every occurrence of AB. For example, the variables ABC, CAB and XYZ79ABN and the strings ABBEY, RABBIT and CRAB.

FL
SEARCH FOR: POKE 4933@,0
This will find and list every POKE of 0 into memory locations 49330 through 49339, including POKE 49330,0; POKE 49331,0; POKE 49332,0; etc.

FL
SEARCH FOR: IF # THEN 100
This will find and list all occurrences of "IF" followed by anything, and ending with "THEN 100", including such things as:

```
IF A THEN 100
IF VAL(X$)=127 THEN 100
IF PEEK(222)=255 OR C$="GOODBYE" THEN 100
```

PROGRAM CHECKER (BRUN D.BUG to load)

New Command: **CHECK (C)**

With any Applesoft program (and D.BUG) in memory, type "C" (return), and your program will quickly be proofread for two specific things:

?SYNTAX ERRORS: This includes misspelled commands (like "PTINT"), improperly punctuated Applesoft statements (like "INPUT A/B") and type-mismatch errors (like A="CAT" and A$=CAT).

?UNDEFINED STATEMENT ERRORS: For example, a "GOTO 100" statement when there is no line 100 in your program.

Sorry: Misspelled words inside quote marks (including DOS commands) and in REM and DATA statements will be ignored. CHECK will also not find ?Illegal Quantity errors and the like. The COMPACT program will find program statements or lines that can't possibly be executed (not really an error; see page 11).

D.BUG does not *cancel* improper statements, it just tells you about them. Maybe your "error" was intentional—like a GOTO 100 when you hadn't typed in line 100 yet. After entering a program line that is improper, you should immediately edit, re-enter or delete the line. And speaking of editing, D.BUG is totally compatible with GPLE (Beagle Bros' *Global Program Line Editor*—see page 39).

If you *want* to type a statement that contains an error, and you don't want to see and hear D.BUG's warning, precede the statement (and line number, if any) with a *slash* ("/"). The slash turns off D.BUG's proofread function for that line only.

ERROR MESSAGES

No Errors: This means that the program in memory when you typed "C" is free of syntax and undefined statement errors.

⟨?⟩ An inverse "?" in a listed program line or statement means a syntax error exists nearby.

⟨#⟩ An inverse "#" in a listed program line or statement means an undefined statement error exists nearby.

"LIVE" SYNTAX CHECKING

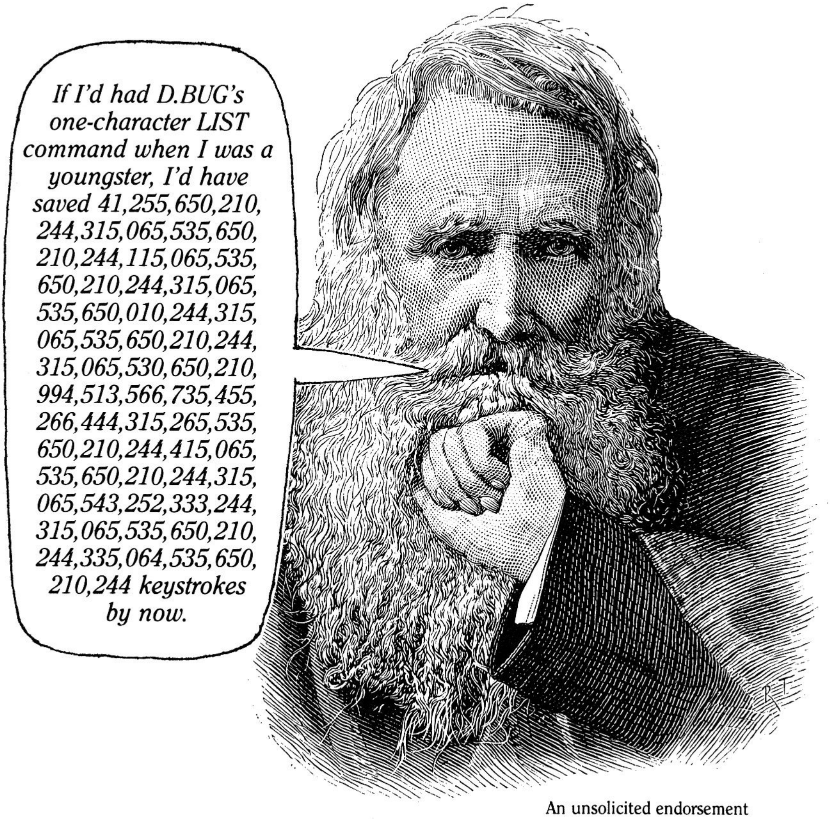
With D.BUG loaded, every time you type *anything,* it will automatically be checked for syntax and undefined statement errors. Improper statements will be listed and flagged with an inverse "?" or "#".

EASY LISTER (BRUN D.BUG to load)

New Command: **L**

With D.BUG loaded, you may simply type "L" (return) to LIST the program in memory. All Applesoft syntax is in effect; therefore you can use commands like L10-100, L-100 and L100-.

GPLE Note: GPLE let's you define *anything* to be typed by ESC plus one other key. GPLE's two-keystroke "ESC-L" List command won't let you specify a line range (a disadvantage), but it does let you add *other* commands in front of LIST, such as TEXT and NORMAL (an advantage).


If I'd had D.BUG's one-character LIST command when I was a youngster, I'd have saved 41,255,650,210, 244,315,065,535,650, 210,244,115,065,535, 650,210,244,315,065, 535,650,010,244,315, 065,535,650,210,244, 315,065,530,650,210, 994,513,566,735,455, 266,444,315,265,535, 650,210,244,415,065, 535,650,210,244,315, 065,543,252,333,244, 315,065,535,650,210, 244,335,064,535,650, 210,244 keystrokes by now.

An unsolicited endorsement

WINDOW TRACER (BRUN D.BUG to load)

New Commands: **TRACE (T)**
NOTRACE (N)
VARIABLES (V)
WINDOW (W)

Note: D.BUG's TRACE and NOTRACE replace Applesoft's versions of the same command.

TRACE (T) (more details on pages 27-28)

Typing "T" before you Run a program activates an adjustable-size "trace window" at the bottom of the text screen that displays line numbers and statements (and optional variable values) as they are being executed. You have the ability to slow program execution down or even execute one program line at a time.

Hardware Note: Most Apple II+ 80-column hardware (non-Apple brand) does not support windows. D.BUG's TRACE will still work, but 80-column screen layouts will scroll improperly. 40-column TRACE will work just fine.

NOTRACE (N)

Typing "N" disables TRACE and related commands.

VARIABLES (V:) (more details on page 30)

With TRACE active, this command lets you specify which variable and expression values will be displayed in the trace window. For example, you could type "V: X,Y,Z$" after typing "T" (for TRACE) to display values for those variables when the program is Run. Typing "V:" with no variables will turn off variable display.

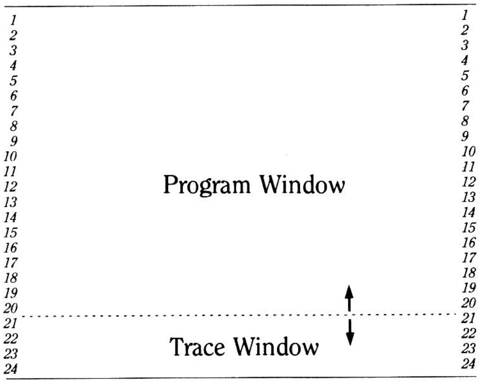
WINDOW (W) (more details on page 29)

With TRACE active before a program is Run, typing "W" lets you pre-adjust the size of the trace window at the bottom of the screen. Press Return when finished adjusting.

The trace window may also be adjusted during a TRACE (see page 28).

TRACE Command Details

When you type "TRACE" or "T", the 40 or 80-column screen will be divided into two *windows* that can function independently. Normally the trace window is the bottom three lines of the screen; the top twenty lines remain for your program output (a line of hyphens takes one more line):



Limitations: Because TRACE alters the screen's normal boundaries, your program's screen formatting may be slightly off (nothing as drastic as Applesoft's old TRACE, though). Graphics program tracing is limited to the four text lines at the bottom of the screen. You can't see the trace window at all when you're viewing page 2.

If TRACE quits tracing and you haven't pressed a key, check your program; it might have a NOTRACE command in it.

CHANGING TRACE SPEED AND WINDOW SIZE

During a TRACE, you may use any of the commands on the next page to control the program's speed or shut TRACE off completely. Any of the WINDOW adjustment commands on page 29 may be made while a program is tracing (no need to press "W" first, however).

EXPERIMENT PLEASE

As we mentioned earlier, IT PAYS TO EXPERIMENT! To fully get the hang (pardon the expression) of all these commands, you need to play around with them—A few keypresses are worth 65,536 words. The following program is a good one to use to test TRACE's commands and functions:

```
10 FOR I=1 TO 100
20 PRINT I
30 NEXT
40 A=A+1: GOTO 10
```

Type this program in, then type "T" and "V: I,A". Now type "RUN" and try the speed-control keys (next page) and the window commands (page 29).

TRACE CONTROL KEYS

The following keys may be used while your program is running (except while waiting for a GET or INPUT statement):

Space Bar: When you press the space bar, your program will execute just one statement and wait for you to press the space bar again. If the trace window is active, you will see each command appear *before* it is actually executed. If you want, you can exit your program (with the usual control-C) before a command is performed.

Return: After you're finished single-stepping with the space bar, the RETURN key will restore normal-speed program execution.

Left Arrow: Pressing the Left-Arrow (Backspace) key during a trace will slow down program execution to one of eight different speeds.

Right-Arrow: Pressing the Right-Arrow key during a trace will speed a program up (assuming it has been slowed down).

Button#1 or Closed-Apple key: Pressing paddle/joystick Button #1 or the Closed-Apple key (IIe/IIc only) will shut off the TRACE function to make your program execute more quickly (normal speed).
Apple II+ Note: You may also shut off TRACE by moving the trace window down off the screen with ctrl-J (next page).

Button#0 or Open-Apple key: Pressing paddle/joystick Button #0 or the Open-Apple key (IIe/IIc only) will restore the TRACE function to its full vim and vigor, windows and all.

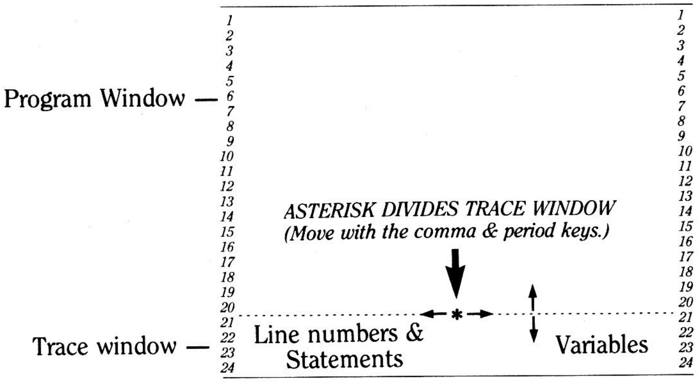
Control-C: Pressing CONTROL-C stops a program, as usual.

Also... You may use the Window Size keys (next page) during a TRACE.

WINDOW Command Details

Pressing "W" (after you have turned on TRACE with "T") will let you make size adjustments to the trace window at the bottom of the screen. If your program is *already running* and TRACE is active, you don't need to press "W" to use the Size Adjustment commands below.

The trace window occupies the full width of the screen in either 40 or 80 columns. An asterisk marks the division between the statement window and the variable window (see next page for variable tracing notes).



WINDOW SIZE ADJUSTMENTS

Up-Arrow or Control-K: Pressing the Up-Arrow (IIe/IIc only) or ctrl-K (any Apple) will enlarge the trace window upward, thus reducing the size of the program window. The trace window may fill all but the top three lines of the screen.

Down-Arrow or Control-J: Pressing the Down-Arrow (IIe/IIc only) or ctrl-J (any Apple) will reduce the size of the trace window downward, thus enlarging the size of the program window. You may even reduce the trace window all the way off the screen to allow your program to execute normally.

>: Pressing the comma key (shifted or not) will enlarge the size of the variable window from right to left 20 spaces at a time, thus reducing the space for traced line numbers and statements.

<: Pressing the period key (shifted or not) will reduce the size of the variable window from left to right 20 spaces, thus enlarging the space for traced line numbers and statements.

VARIABLE Command Details

Before a program is Run (and after turning on TRACE), typing "V:" followed by one or more variables and expressions will make those variables' values appear in the right section of the trace window (see WINDOW, previous page). Due to screen space limitations, only the first five characters of the variable or expression itself will be visible. If the variable value (or string) is longer than 12 characters, it will also be truncated.

Typing "V:" (note the colon) alone will cancel all variable tracing. So, of course, will a NOTRACE command.

EXAMPLES:

```
T
V: X
RUN
```

"T" turns on TRACE so that line numbers and statements will be printed during execution. "V: X" says trace variable X too. "RUN" starts program execution. You may now use <, >, and the Up/Down arrows (or ctrl-K/J) to adjust the size and arrangement of the trace window.

```
T
V: X, X+50, CHR$(N)
RUN
```

This command would trace the variable X, the expression X+50 and the character whose ASCII value is N.

```
T
V:
RUN
```

This command would trace line numbers and statements only. The "V:" cancels all variable tracing.

VARIABLE ERRORS

Some expressions, like CHR$(-65) or 123/0, are impossible to interpret, so they will produce an error message during a trace.

For example: V: X%, X%*256, CHR$(X%), NA$(X%)

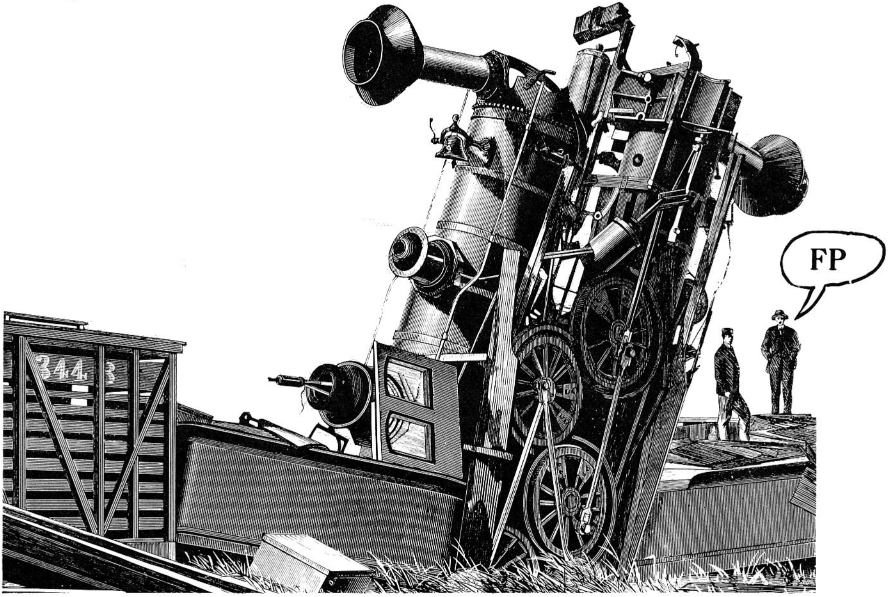
This VARIABLE command could produce the following output in the variable window. While X%=65, you might see:

```
    X%     = 65
    X%*25  = 16640
    CHR$(  = A
    NA$(X  = NOW IS THE TIME
```

While X%=-1, you might see:

```
    X%     = -1
    X%*25  = -256
    CHR$(  = ** ERROR **
    NA$(X  = ** ERROR **
```

Notice how the expressions are shortened— "CHR$(X%)" becomes "CHR$(", etc. Also notice that some of the expressions have produced appropriate error messages.



Type "FP" any time you want to destroy D.BUG and all of its commands.

DUMP TRACER (BRUN D.BUG to load)

New Commands: **DUMP (D)**
SIZE (S)
ZAP (Z)

DUMP (D) (more details on next page)

After a program "crashes" at an error or stops for any reason, typing "D" will *dump,* or list, in order used, the last line numbers and statements that were executed. In effect, you can print a history of your program's execution.

If you follow "D" with a number, only that number of statements will be dumped. Typing "D12", for example, would dump 12 statements. (Exceptions: See SIZE and ZAP.)

SIZE (S)

Typing "S" followed by a number (for example "S500") will adjust the size of the *dump buffer,* letting it hold, in this example, 500 statements. The higher the number, the less memory you have available for other uses (each statement occupies only 2 bytes).

SIZE will determine the number of statements dumped if you try to DUMP a number larger than the dump buffer.

The minimum buffer size is 5 (accomplished by simply typing "S"). The maximum is somewhere above 10,000 (100 is a more practical upper limit). The default, when you load D.BUG, is 50 statements (100 bytes).

The SIZE command will CLEAR all of your variables. You will need to RUN your program again before you can do another DUMP.

Technical Note: The dump buffer resides above HIMEM. When you change the buffer's size, HIMEM is adjusted accordingly.

ZAP (Z)

"Z" will ZAP (clear) the dump buffer.

IMPORTANT: You must ZAP the dump buffer if you want to type "RUN FILENAME" (a DOS command) followed immediately by a DUMP. Otherwise you'll be looking at meaningless garbage. Typing "RUN" alone (an Applesoft command) *automatically* accomplishes a ZAP (for example, if you typed "LOAD FILENAME" and then "RUN"). A ZAP is also accomplished by adding, changing or deleting a program line, or using the SIZE command.

DUMP Command Details

With your program (and D.BUG) loaded, type "RUN". When the program stops, for any reason, type "D" to trace the last program lines and statements executed. If you don't specify a number, the entire contents of the buffer will be dumped. If you want fewer or more statements dumped, include a number after the D—like D3 or D2500. You can only dump as many statements as the dump buffer will hold (see SIZE, previous page).

If you want to capture a DUMP on your printer, type "PR#1" before you type "D".

The trace buffer is automatically cleared when you make any changes to a program (add, delete or modify a line). Therefore a DUMP after a change will produce nothing. Always DUMP immediately after your program has been RUN and stopped.

If you don't want to clear the dump buffer when you RUN a program, type "RUN line#" instead of "RUN" ("line#" is your program's first line). Now when you DUMP, earlier program Runs will be dumped first, with each separate RUN separated by a horizontal line. Just type "RUN" if you don't want earlier executions shown.

DUMP CONTROL KEYS

The following keys may be used during a DUMP:

Left and Right-Arrows: The Arrow keys control DUMP direction. Pressing the Backspace key during a DUMP will start dumping backwards through the dump buffer—just in case you missed something. Inverse or flashing "R's" will indicate Reverse dumping. Don't attempt analysis of a program while dumping backwards (it's confusing). Re-establish normal dumping with the Right-Arrow.

Control-S: Pressing control-S during a DUMP will pause the display until another key is pressed. If you hit one of the direction keys during a pause, the listing will continue in whatever direction has been specified.

Space Bar: Pressing the space bar during a DUMP will step through the dump buffer one statement at a time. Any other key resumes normal speed.

Control-C: Pressing control-C exits a DUMP.

BREAKPOINTS (BRUN D.BUG to load)

New Commands: **BREAK IF...**
BREAK ON...
BREAK AT...
BREAK LIST
BREAK+
BREAK-

If you've ever been stumped by problems like "How did A$ get set equal to "ABC"?" or "When does that X variable get assigned the value 10000?" or "When does location 216 get Poked with a zero?", you're going to like the BREAK command.

Typing "BREAK:" (or "B:") followed by a number, 1-8, and some instructions, tells your program to stop IF a specific condition is true, or ON the occurrence of a certain command, or AT the execution of a certain line number.

BREAKPOINT DEMO

Type in this little program so you can test some of the different types of D.BUG breakpoints on the following page:

SAMPLE PROGRAM:

```
5 TEXT: HOME
10 X=INT(RND(1)*20)
20 PRINT X
30 IF X+2 THEN PRINT CHR$(7): GOTO 10
40 IF NOT INT(RND(1)*200) THEN 10
50 PRINT "END": END
```

SAMPLE BREAKPOINTS:

Now type in the following sample breakpoint commands:

```
B1: IF X=10
B2: ON GOTO
B8: AT 20,10
```

The number after the "B" indicates the number of the break statement. *Any* number, 1 through 8 is allowed.

BREAK Command Details

BREAK IF...

The first sample breakpoint, "B1: IF X=10" will cause a program, when RUN, to stop any time X becomes equal to 10. When you Run the program several times, it will stop with the following message:

```
BREAKPOINT 1: IF X=10
BREAK IN 20
```

The first line tells you which breakpoint caused the break. The second line tells you the *next statement executed* after X was set equal to 10.

Here are some other BREAK IF... examples (Remember, the number after the "B" could be any number 1-8.):

B1: IF X=50 AND Y>50 would break any time X=50 and Y is greater than 50

**B1: IF DF$="DOGFOOD"** would break any time DF$="DOGFOOD"

Note: The maximum size of a BREAK IF... or BREAK ON... statement is 32 bytes.

BREAK ON...

The second sample breakpoint (previous page), "B2: ON GOTO", will stop the program on the first statement that begins with a GOTO.

Here are some other BREAK ON... examples that could be used to test other programs:

B2: ON PRINT would break on any statement that started with PRINT

B2: ON PRINT " would break on a statement like PRINT "HELLO", but not on a statement like PRINT A$ or just plain PRINT.

B2: ON IF X=25 AND Y=30 would break on any statement that started with "IF X=25 AND Y=30".

B2: ON AND Y=30 would *never* break, because a statement cannot legally start with "AND".

B2: ON Y= would break the first time Y is assigned a value.

Note: The maximum size of a BREAK IF... or BREAK ON... statement is 32 bytes.

BREAK AT...

The third sample breakpoint (previous page), "B8: AT 20,10", tells the program to stop at line 20 the 10th time a line 20 statement is encountered. (Again, the number 8 was used only as an example; any number, 1-8, is legal.)

The "AT 20,10" actually means the 10th execution of *any* line 20 statement. So if line 20 contained *two* statements, the program would break on the *fifth* encounter with line 20. A "B1: AT 20" command (with no second number) would break the *first* time line 20 is encountered.

BREAK LIST (BLIST)

Typing "BLIST" will list the breakpoints currently in memory, like this:

```
+BREAKPOINT 1: IF X=10
+BREAKPOINT 2: ON GOTO
+BREAKPOINT 8: AT 20,10
```

Typing "B2LIST" would list only breakpoint 2:

```
+BREAKPOINT 2: ON GOTO
```

BREAK-

The plus sign (+) next to each listed breakpoint (above) indicates that that breakpoint is active. A minus sign (-) indicates a breakpoint that is inactive; it will be ignored. To de-activate a particular breakpoint, like number 2, you can type "B2-". To de-activate all breakpoints, type "B-".

BREAK+

To re-activate a particular breakpoint, like number 2, you can type "B2+". Typing "B+" will activate all breakpoints.

Note: Initially defining a breakpoint automatically activates it.

REMOVING BREAKPOINTS

You can erase a particular breakpoint by typing "B" followed by a number and a colon (like "B2:").

BREAKPOINT Notes

CONTINUING AFTER A BREAK

You can usually use the Applesoft command "CONT" if you wish to continue running your program after a break. *BUT,* if your program stopped due to a BREAK IF or BREAK ON condition, you will need to either turn off the breakpoint or change the condition that caused the break. Otherwise you've got another break coming immediately. For example, if your program stopped because A=5, and you try to continue with A still equal to 5, the program will immediately break again (because A still equals 5). You could use a direct keyboard command (don't change your program) to set A equal to some other value; then use CONT.

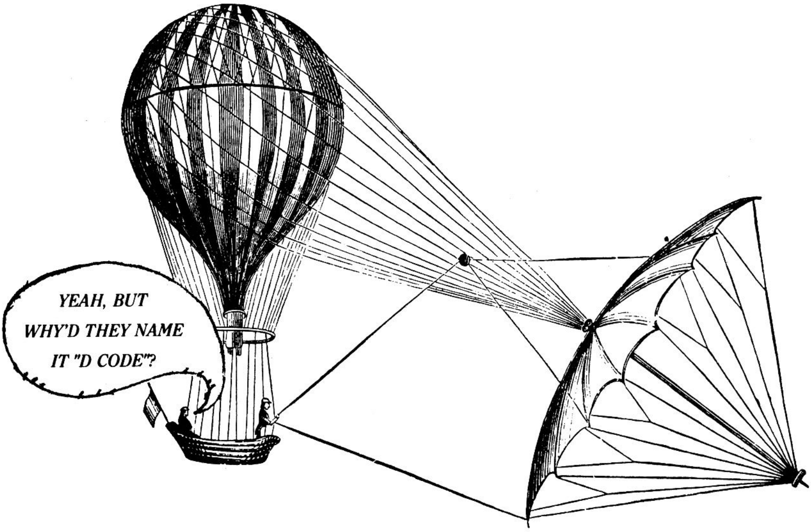
Note: If you type a direct keyboard command with an error in it, you disable the CONT command. Don't ask why; we don't know.

USING BREAK IN YOUR PROGRAMS

Just like all D.BUG commands, breakpoints can be defined, activated and deactivated from within your Applesoft programs.

PROGRAM SPEED

BREAK IF breakpoints will often slow program execution considerably, depending on how many breakpoints have been specified, and their complexity. However, BREAK ON and BREAK AT will not slow a program down much at all.



(38 is blank)

(Thinly-disguised advertisement)

GPLE

GLOBAL PROGRAM LINE EDITOR by Neil Konzen

$49.95, Compatible with any Apple II, DOS 3.3 and ProDOS—Includes Apple Tip Book #7

A "WORD PROCESSOR" FOR APPLESOFT PROGRAMS

GPLE is *The* classic Applesoft line editor for the Apple. It lets you edit your program lines fast without awkward cursor-tracing or clunky "escape editing" methods.

GPLE is installed in memory when you boot, remaining "invisible" to your programs and unaffected by even the most "destructive" commands, such as FP and INT. You may install GPLE in normal 48K memory or in the Language Card (built-in on all IIe's and IIc's).

INSERT AND DELETE

Now you can make almost instant changes to any Applesoft or Integer Basic program line. GPLE lets you jump the cursor to the change point in the line and insert or delete text. Other code in the line moves aside to make room (what you see is what you get). If you make a mistake, you can restore the line to its previous condition with a keystroke.

Control-characters are easy to insert and delete, appearing in inverse when being edited.

With GPLE, it is no longer necessary to trace the cursor to the end of the line you are editing. No matter where the cursor is, hit Return, and that line is entered into memory.

GLOBAL SEARCH & REPLACE

GPLE finds any word or variable in a program, letting you change that line, delete it, or just look at it. Here are some examples of GPLE's Global capabilities:

- Look at all lines containing a GOSUB.
- Edit or delete all lines with a REM.
- Locate all occurrences of the variable XX.
- Replace all X-variables with ABC's.
- Change all *Hello* strings to *Good-Bye's.*

DEFINABLE ESC FUNCTIONS

GPLE lets you define an ESC-keypress followed by any other key to perform any keyboard task. For example, *ESC 1* can catalog drive 1, *ESC L* can do a "HOME: LIST", *ESC N* could type an entire subroutine... *Anything* you want, whenever you want it.

A complete set of Escape functions is included with GPLE, pre-programmed and ready to use. Each function may be used as is, or deleted or changed whenever you like. After you create your own "Escape Table", you can save it on disk so it will be in memory the next time you load GPLE.

80-COLUMN COMPATIBILITY

All GPLE edit and global features support Apple 80-column cards *and* most 80-column cards on *any* Apple IIc, IIe, II+ or II.

Double-Take, ProntoDOS, DOS Boss, Flex Type, etc.,—and, of course, all of your Applesoft and Integer Basic programs—get along quite well with GPLE.

GPLE DOS MOVER

GPLE comes with its own "DOS Mover" program that lets you move DOS to the Language Card (built-in on all IIc's and IIe's) for an EXTRA 10,000 Bytes (10K) of programmable memory. GPLE itself may be located on the Language Card or in Main 48K memory.

PLUS APPLE TIP BOOK #7

Learn more about your Apple—GPLE comes with more tips and tricks from Beagle Bros, many involving GPLE. Hours of good reading and Apple experiments.

I CAN'T *BELIEVE* I PROGRAMMED ALL THOSE YEARS WITHOUT GPLE!

D CODE Index

Abbreviated Commands 20
Ampersand (&) 10, 18
Applesoft Files, Compare 18
Back-ups 2
Beginner Info 2, 4
Binary Files, Compare 18
BREAKPOINT (B) 34-37
Breakpoints 34-37
Cash, Free 45
Catalog......................4
CHECK (C) 24
COMPACT 3, 6-14
COMPARE 3, 17-18
Concatenate Lines 8
Continuing After Break.........37
D.BUG 3, 20-37
D.BUG.DEMO 20
DOS 3.3/ProDOS 2, 4, 5
DUMP (D) 32
Dump Tracer 32-33
Easy Lister 25
Eighty-Column Trace Problem ...26
Error Finder 24
Errors, Variable Tracing 31
Fast Finder............... 22-23
FIND (F) 22-23
FIND and LIST (FL)...........22
FP 4, 20
GPLE 22, 25, 39
Imbedded Machine Code 11
LINE.SPLITTER...............15
LIST (L) 25
NOTES.......................4
NOTRACE (N) 26
ProDOS/DOS 3.3 ....2, 4, 5
Program Checker 24
REM Removal 8
SIZE (S) 32
Slowing Down Programs 28
STARTUP 4, 5
Stepping Through a Program 28
Syntax Errors, Finding 24
Text Files, Compare 18
TITLE 4
TRACE (T) 26, 27-28
Undefined Statement Errors 24
Unused Statements 11
VARIABLES (V) 26, 30-31
Variables, Compacted 9, 12-13
Variables, Tracing 21, 30-31
Wildc@rd C#aracters 22
WINDOW (W) 21, 29
Window Tracer 26-31
ZAP (Z) 32

Disclaimer of All Warranties and Liabilities

Even though the software described in this manual has been tested and reviewed, neither Beagle Bros nor its software suppliers make any warranty or representation, either express or implied, with respect to this manual, the software and/or the diskette; their quality, performance, merchantability, or fitness for any particular purpose. As a result, the diskette, software and manual are sold "as is," and you, the purchaser, are assuming the entire risk as to their quality and performance. In no event will Beagle Bros or its software suppliers be liable for direct, indirect, incidental, or consequential damages resulting from any defect in the diskette, software, or manual, even if they have been advised of the possibility of such damages. In particular, they shall have no liability for any programs or data stored in or used with Beagle Bros products, including the costs of recovering or reproducing these programs or data. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

ProDOS™

This product includes software, ProDOS™, licensed from Apple Computer, Inc. Apple Computer, Inc. makes no warranties, either express or implied, regarding the enclosed computer software package, its merchantability or its fitness for any particular purpose. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

Rush—Software Order

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 200751 SAN DIEGO, CALIFORNIA

POSTAGE WILL BE PAID BY ADDRESSEE:

BEAGLE BROS INC.
Attn: Minnie Assembler, Order Dept.
3990 Old Town Ave., Suite 102C
San Diego, California 92110

D CODE
PROOF OF PURCHASE
(Save for disk or manual updates.)

More Beagle Bros Apple Software

(WHAT'S NEW? Check our ads in *A+*, *Call-A.P.P.L.E.*, *inCider*, *Nibble* and other Apple® magazines.)

■ *GRAPHICS* ■

□ ALPHA PLOT (II+, IIe, IIc)† $39.50
Normal hi-res (6 colors, 280x192 pixels) drawing and typing on both hi-res pages. Compress pictures to 1/3 disk space.

□ APPLE MECHANIC (II+, IIe, IIc)† 29.50
Create hi-res shapes for animation with Applesoft's DRAW & XDRAW commands. Put fancy hi-res type in your programs. List & learn demo programs teach you hi-res programming.

□ APPLE MECHANIC TYPEFACES† 20.00
26 new editable fonts to be used with Apple Mechanic.

□ BEAGLE GRAPHICS (IIc or 128K IIe)★ 59.95
Double hi-res drawing (16 colors, 560x192 pixels) and typing in many typestyles (all editable). Color fill, cut & paste, 200+ color mixes. 33 new commands for using double-res in your programs. Convert normal hi-res pictures and programs to double hi-res, compress pix to 1/3 disk space...

□ FLEX TYPE (II+, IIe, IIc)† 29.50
Variable-width text (wide, normal, condensed) controllable with normal Applesoft commands. No 80-column card reqd.

□ FRAME-UP (II+, IIe, IIc)† 29.50
Make Apple "slide shows". Keyboard controlled or unattended, using your existing hi-res, lo-res and text screens.

□ TRIPLE-DUMP (II+, IIe, IIc)★ 39.95
Transfer any image, including double hi-res, to your dot matrix printer. Make Giant (8" high characters) Banners too.

■ *ALL-PURPOSE* ■

□ DISKQUIK (IIc or 128K IIe)† $29.50
Acts like half a disk drive in slot 3. Silent and fast as a hard disk. Load/save files in memory with normal commands.

□ FATCAT (II+, IIe, IIc)★ 34.95
Reads all of your DOS 3.3 and ProDOS file names into one or more Master Catalogs for sorting, searching and printing. Alphabetize file names on disks. Compare any two files.

□ PRONTO-DOS (II+, IIe, IIc)† 29.50
Triples the speed of loading and saving. New TYPE command displays text file contents. Move DOS for extra 10K.

■ *PROGRAMMING* ■

□ BEAGLE BASIC (IIe, 64K II+)† $34.95
Puts Applesoft in RAM so you can change it and add enhancements—new commands like if-then-ELSE, SWAP variables, GOTO/GOSUB-a-variable, TONE, HSCRN, etc.

□ D CODE (II+, IIe, IIc)★ 39.95
Compact Applesoft programs and reveal unused code. Autoproofread Applesoft programs, even as you type. Trace any number of program statements *after* stopping a program...

□ DOS BOSS (II+, IIe, IIc)† 24.00
Reword DOS 3.3 commands. Change "Catalog" to "Cat", "Syntax Error" to "Oops" or *anything.* Includes many meaty tips for altering DOS, including program "save-protection".

□ DOUBLE-TAKE (II+, IIe, IIc)★ 34.95
2-way scroll for Listings & Catalogs. Better List-format, fast variable+line number display, better renumber/append, auto line-numbering, instant hex/dec converter and more.

□ GPLE (II+, IIe, IIc)★ 49.95
Edit Applesoft without cursor-tracing. Features insert & delete and fast search & replace. Make all keys be "function keys" to type anything you like (ESC-1 catalogs disk, etc.). Move DOS 3.3 out of main memory to add 10K of space.

□ SILICON SALAD (II+, IIe, IIc)† 24.95
Over 100 utilities and tricks— hi-res program splitter, DOS killer, disk scanner, hi-res text imprinter, 2-track catalog...

□ TIP DISK #1 (II+, IIe, IIc)† 20.00
100 tips on disk from Tip Books 1-4. Fascinating Apple programming techniques. Includes Apple Command Chart.

□ UTILITY CITY (II+, IIe, IIc)† 29.50
21 utilities— List-formatter puts each statement on a new line, multi-column catalogs, invisible/trick file names, etc.

■ *GAMES* ■

□ BEAGLE BAG (II+, IIe, IIc)† $29.50
12 games on one disk. Voted to 1983's MOST POPULAR list in *Softalk* poll. The best Apple game bargain on the market.

□ I. O. SILVER (II+, IIe, IIc)† $29.95
Two games in one—a great strategy game and a fast action arcade game. Superb unlocked machine language graphics.

† Supports DOS 3.3 only
★ Supports both DOS 3.3 and ProDOS™

(Subject to change—See our current ads ~~or~~ catalog.)
"APPLE" is a Registered Trade Mark of Apple Computer, Inc.

D CODE™, Copyright © 1984, Alan Bird
ISBN 0-917085-07-8

Published by BEAGLE BROS MICRO SOFTWARE, INC.
3990 Old Town Avenue, San Diego, California 92110