



THE BANTAM COMMODORE USER'S LIBRARY

THE BIG TIP BOOK

FOR THE COMMODORE 64/64C/128

JOHN ANNALORO AND BERT KERSEY
THE BEAGLE BROTHERS



**THE
BIG TIP BOOK
FOR THE
COMMODORE**

**THE
BIG TIP BOOK
FOR THE
COMMODORE**

John Annalaro and Bert Kersey



BANTAM BOOKS

TORONTO • NEW YORK • LONDON • SYDNEY • AUCKLAND

Dedicated to R.A., B.A., ?A., and me.

THE BIG TIP BOOK FOR THE COMMODORE
A Bantam Book / June 1987

All rights reserved.

Copyright © 1987 by John Annaloro and Bert Kersey.

Cover design copyright © 1987 by Bantam Books, Inc.

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 0-553-34411-0

Published simultaneously in the United States and Canada

Bantam Books are published by Bantam Books, Inc. Its trademark, consisting of the words "Bantam Books" and the portrayal 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
B 0 9 8 7 6 5 4 3 2

CONTENTS

Preface	vi
1. Getting Started	1
2. For Beginners Only: Inside Tips	11
3. Pointers for Beginners	27
4. Converting Programs to the 128	37
5. Basic Tricks	43
6. Screen and Text Graphics	53
7. Memory and Speed	61
8. Useful Applications	71
9. Protection	77
10. Advanced Programming Tricks	87
11. Machine Language	97
12. Disks and Drives	107
13. Audio and Video Communications	123
14. Printers and Printing Tips	131
Appendix A 64 and 128 Basic Commands	141
Appendix B Abbreviations for Basic Keywords	157
Appendix C Commodore Character Codes	163
Appendix D Escape Codes for the 128	169
Index	171

PREFACE

I like programming. We like programming. And working in the same office with a group of professional programmers, it is natural to learn some great tricks. A few of the tips in this book have come from their answers to questions asked. Some have come from snooping and eavesdropping. Many of the tips have come from watching someone show off, in a manner of professional rivalry and one-upmanship, that says “I can do more with the Commodore than you.” And sometimes they can.

Most of the tips have come from my experience of writing software for the computer marketplace. I hope you find them to be educational and enjoyable. Knowing that 5,000 people will be using a program is a major motivation to write some showy code. Further, knowing that magazine reviewers will be criticizing or praising our work in the computer trade press has had us losing sleep over how to impress the world.

Once, we tried keep a library to store all the algorithms that could be filed—especially for the fun of keeping up with our colleagues in the business. So we have made notes, kept a clipping file from magazines, collected these tips, experimented, and improvised to create some programming tricks. We’ve looked in books, taken apart other software packages, and generally turned diskettes inside-out to put together a library of interesting Commodore tips to use, ponder, or play with. So here it is.

Some of the tips are very useful. They can be a valuable help in lots

of ways. Some are ingenious ways to do little things. A few are just weird—only to be used on days when you might need to amuse yourself. And some are apparently useless, like, for instance, this mini-musical word processor for the Commodore 128.

```
10 TRAP 20: GET A$: PRINT A$;: PLAY A$: GOTO 10
20 RESUME 10
```

This lets you type onto the screen of the computer, but also “plays” the musical notes that correspond to the letters, A, B, C, D, E, F, and G when they are pressed. It is perfect for someone needing to write a letter to Ozzy Osborne.

If you have an “older” Commodore 64 (rather than a 128), all is not lost. The mini-word processor program listed below turns your old 64 into an even older-style typewriter. Every time you press the RETURN key a sweet little bell sounds.

```
10 GET A$: PRINT A$: IF A$ <> CHR$(13) THEN 10
30 F = 54272
40 FOR X = 0 TO 24 : POKE F+X,0 : NEXT
50 POKE F+1,125
60 POKE F+5,11
70 POKE F+15,250
80 POKE F+24,15
90 POKE F+4,21
100 FOR T = 1 TO 999: NEXT: PRINT
110 GOTO 10
```

These two little gems are here at the beginning of the book in case you want to make notes. Lots and lots of notes. Better yet, by changing the programs just a little, both can be modified for use with your printer. Now you can really transform hundreds of dollars in computer equipment into the bygone era of the typewriter. There’s nothing like being on the trailing edge of technology.

First, in either program, add these lines:

```
1 OPEN 7:4
5 GET A$: IF A$ = " " THEN 5
```

Then, for the 128, change line 10 to read:

```

10 TRAP 20: PRINT A$: PLAY A$: PRINT#7,A$:
   GOT0 5

```

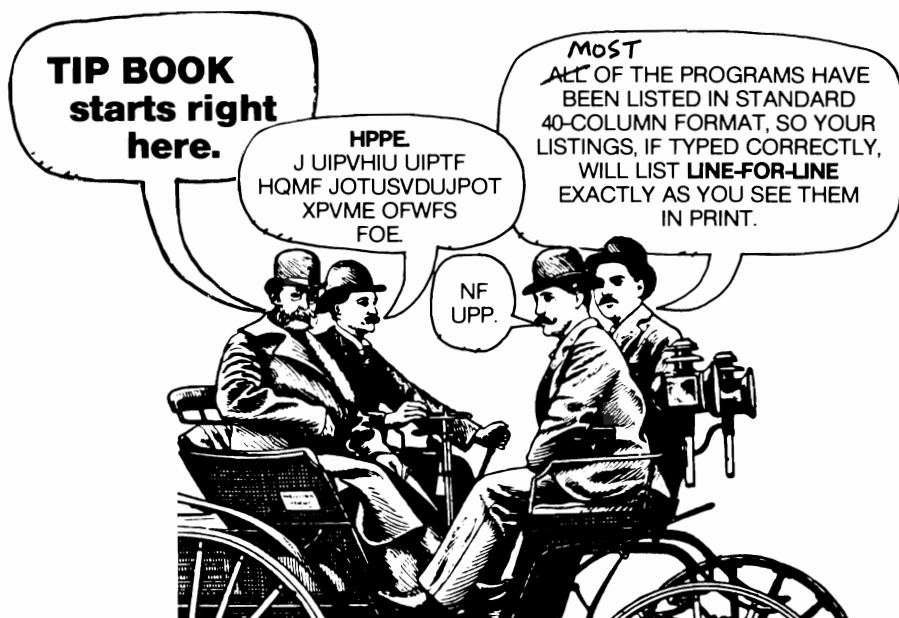
Or for the 64, change line 10 to:

```

10 PRINT A$: PRINT#7,A$: IF A$ <> CHR$(13)
   THEN 5

```

Then change the last line of the program to return to line 5, rather than line 10. Thats all; the wonderful weird old world of word processing is all yours!



MORE ON TAKING NOTES

Remember all those little write-protect tabs that come with a package of diskettes? Ever wonder what to do with the hundreds of extra ones? Well, here's Beagle Bros. Big Tip No. 1. Use those little devils to mark the important pages in this book. Fold one over each page-to-remember, and let the outer edge stick out about one-fourth inch. . . .

CHAPTER 1

GETTING STARTED

BURNING IN

Here's a known Beagle fact. With any complex electronic component, if a new piece of equipment is going to fail it will do so early in its operational life. So a common practice is to "burn in" a new computer (or TV set, disk drive, etc.). That means to leave it powered up round-the-clock after you first get it home, to test its reliability.

It doesn't need to be doing anything. Just leave it turned on for about 72 hours straight. Check it occasionally to see if it still works. If it doesn't, at least you can get your money back because it broke while it's covered by a warranty.

The chances of something going wrong are small, since most of the electronic equipment you buy has already been burned in and tested at the factory. But you'll have a much greater feeling of confidence in your computer purchase if you do it (again) yourself.

HOOKING UP EQUIPMENT

The disk drive and monitor connection sockets are right next to each other on the back of the computer, and look a lot alike. It's a good idea to label them so time's not wasted jamming a five-pin DIN monitor connector into the disk drive's serial port. On the 64, the monitor hookup is on the right. When the 128 was designed, some industrial designer was paid a lot of money to switch the sockets around—now it's on the left. Use the plastic embossed tape-type labels so you can tell at a glance which cable goes into which slot.

NEVER DO THIS

We don't know why you would want to do this, but some people have found that it is possible to plug the 64's round power supply input into the round monitor or disk drive socket. Since this could do major damage, don't try it. And to be sure it doesn't accidentally happen, use a piece of colored tape to mark the incoming cables and the corresponding correct sockets. That way if Uncle Louie begins his career in computer science with your equipment. . . .

CABLE TIE-UPS

Twist ties or rubber bands can be used to keep your computer desk from looking like an explosion in a licorice factory. Curl the unused lengths of monitor, disk drive, and printer cables, then tie them up. For those who want to spend money on such bondage devices, Radio Shack sells "official" cable ties that do the same thing and cost more.

DISK DRIVE PLACEMENT

Use a radio to find out at which side of your TV set or monitor to place your disk drives. Turn the radio on and place it first on the left and then on the right side of the set. The side that causes the most interference on the screen and/or radio *isn't* the place for your disk drives.

WHERE SHOULD THE DISKS GO?

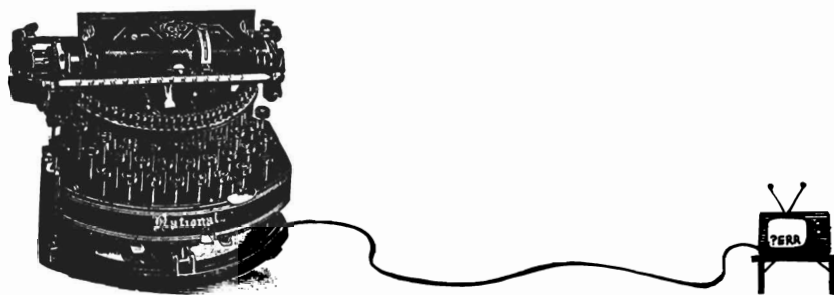
If you worry that strong magnetic fields could corrupt your valuable data disks and expensive software, check the area around the desk library with a compass. If there is danger lurking, the compass needle will point it out.

NO-TEAR SLEEVES!

Next to creating high-resolution graphics shapes from scratch in 64 mode, 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 polling and research by an independent testing lab shows that no one knows why, although dust prevention is undoubtedly the answer.

CONTROL PORTS

There are two joy-stick control ports on the right side of your computer. If your game hot-stick won't work, plug it into the other port. Usually, it's no. 2; that's the "one" to use.



MONITOR SELECTION

Experts say (and we 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), and amber is a workplace-eye-safety requirement in some European countries. If you do a lot of word processing an easy-eye monitor may be well worth the investment.

The only thing worse than a black and white monitor is a color TV . . . unless you are watching cartoons.

EYE-SAVING TIP

If your eyes tire from staring at the monitor, give some thought to having your vision checked. If that’s the problem, glasses may be less expensive than buying an endless string of monitors in search of a “good” one.

GLARE

Locate your monitor or TV set away from a window, in a location that won't get the distraction of direct sunlight's glare on the screen.

And, on nice days, locate your monitor away from the window, so you won't be tempted to glare outside at the distracting direct sunlight.

MONITOR MANAGEMENT

Monitors seldom need replacement but frequently need color adjustments. Type in this color bar program and save it for when you need to make many minor monitor management modifications.

```
10 FOR X = 1 TO 11
20 READ A
30 FOR Y = 1 TO 80
40 PRINT CHR$(A); "{RVS-ON} {SPACE}";
50 NEXT Y: NEXT X: END
60 DATA 5,28,30,31,144,156,158,159,150,149,153
```

DON'T DRINK AND COMPUTE

While the 64 and the 128 provide great party entertainment, beware of whatever liquid spilling into your keyboard. If it does get wet in there, immediately turn off the power, disconnect the machine, open it up, and let Mr. Commodore dry himself out for at least a day.

CLEAN ME

There's no excuse for a dirty computer, just because you can't give the machine a bath. A faintly damp cloth can clean the case and a stiff paint

brush should be used to dust between the keys. It's also a good idea to cover the computer when it's not in use.

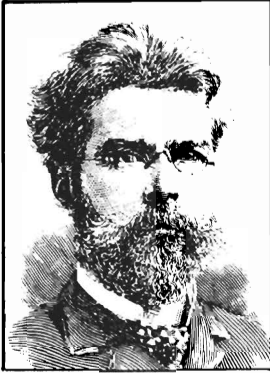
POWER STRIPPING

A multiple outlet box with its own power switch (and fuse, or even a surge suppressor) is a good way to turn on all your equipment at once. It's a lot easier than feeling around the back of all your peripherals to turn on the computer, monitor, disk drive, printer, and modem—one at a time.

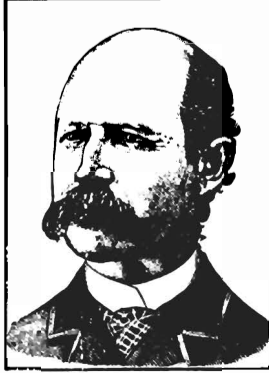
POWER SUPPLIES

Also use a multi-outlet power strip with its own power switch to completely turn off the computer's power supply box. Even when Mr. Commodore is turned off, the box will draw power if it is plugged in—staying unnecessarily warm. Eventually this can lead to the power supply's failure. We're careful now, here at this Beagle Bros. Branch Office, because we have three dead power supplies.

●The Beagle Bros. Staff●



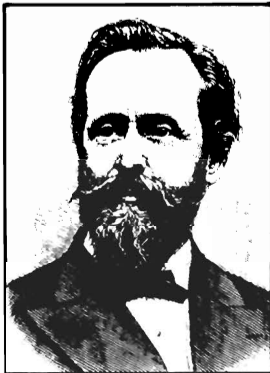
Al Gorithm



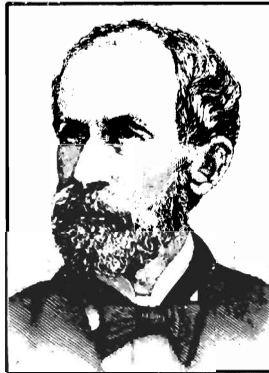
Len Adollar



Flo Chart



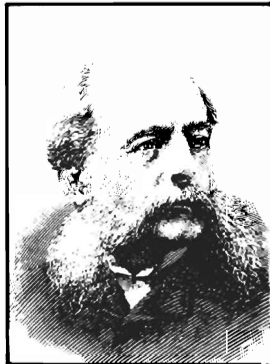
Tex Window



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.

RACE NOTES FROM BEAGLE HEADQUARTERS

Our records show that Peter Lake, from somewhere-out-there, called the Beagle Building to report on his customized computer environment. It seems Peter is a professional writer and uses his computer as a word processor all day, every day. So he mounted his system on a drawing board tilted at about 15 degrees. To level his monitor he built a wedge-shaped mount. He painted his computer case black to reduce glare (we still have not noticed glaring from the case). He glued sequins to his K and D keys so he can feel his fingers' position on the keys without looking. (Peter, the 128 already has these little bumps.) Now for the best part: he sits in a special race-car seat, headrest and all, leans back, and types away.

We asked Peter if he wears his seat belt; he said, "Only when I type fast."

PAGE MARKERS

We all have reference books and magazines with bits and "gnybbles" of information that we go back to, time and time again. A great way to mark the important pages in a magazine (or in this book, as mentioned before) is with the little write-protect tabs that come with disks; fold one over and let it stick out. After all, in every package of diskettes there are at least ten extra tabs wishing for a useful application.

TIME TO GO

If you don't have a clock around and you need to know when it's time to stop what you're doing, ask the computer. While the Commodore doesn't wear a watch, it does have an internal timekeeper that tells how long it's

been since the machine was turned on. The running time displays in hours, minutes, and seconds (HHMMSS). In the direct mode, just type:

```
PRINT TI$
```

Better yet, set TI\$ to the current time when you first sit down. Then you'll always have the correct time.

CHAPTER 2

FOR BEGINNERS ONLY: INSIDE TIPS

BASIC

In the programming language BASIC, every instruction you give the computer has a precise spelling use and definition. When these instructions and rules are all added up, the end result is a method of getting the computer to do as it's told.

Each instruction given the computer must be expressed in exact terms, and the way these commands are phrased is called *syntax*. If you make a mistake, the Commodore will stop what it's doing and just sit there, displaying the following error message on the screen:

?SYNTAX ERROR

It's then up to you to go back to find the mistake and correct the phrasing or spelling of your commands.

The Commodore 64 and 128 understand only a small set of *keywords*, and a smaller number of letters called *variables*. The Commodore will always know what the keywords mean, and their use remains the same in every program. If, for instance, line 10 of your program tells the computer to directly GOTO line 600, Mr. Commodore will go directly there—skipping the rest of the program instructions. No matter what the computer

was told in previous instructions, whenever it encounters the GOTO statement it will immediately do as directed. It understands the keyword GOTO yet will not allow you to substitute other words with similar meanings in English for BASIC commands. You must use GOTO, you cannot use JUMPTO, SKIPTO, or YO—GO OVER THERE.

Without a doubt, the most frequently used command in BASIC is the PRINT statement. A string of letters enclosed in quotation marks following the word PRINT can be written on the screen whenever and wherever you desire. In a program, of course, the command must have a line number:

```
10 PRINT "MY NAME IS AXELROD"  
20 PRINT "BUSINESS IS THE CORNERSTONE OF  
    COMMERCE"
```

In the *program mode*, nothing happens until your program is RUN. Type the word RUN and press the RETURN key, and the monitor will display the exact string of letters enclosed within, but not including, the quotation marks.

When you entered the RUN command, there was no line number preceding the RUN instruction. The absence of a line number causes the computer to process immediately the instruction on the screen. For example, enter:

```
PRINT "I LOVE MY COMMODORE COMPUTER"
```

Then press the RETURN key *without* giving the command a line number. Immediately Mr. Commodore will display the message enclosed within the quotation marks. This is called the computer's *direct mode*.

THE VALUE OF A QUOTE

Any letters, numbers, graphic symbols, or punctuation marks following the PRINT statement will be reproduced on the screen exactly as you typed them, if they are in quotation marks. If they are not enclosed in quotation marks, the computer always assumes that the information rep-

resents a mathematical equation, and proceeds to try to do your arithmetic. Numbers will be processed as figures and letters will read as a representation of an unspecified numeric amount. To see how it works, enter this mathematical equation in the direct mode:

```
PRINT "20 + 19"
```

Press the RETURN key and the computer will display:

```
20 + 19
```

Now enter the same PRINT statement without the quotation marks. As soon as you press the RETURN key Mr. Commodore replies with the answer to your math problem:

```
39
```

PUT IT WHERE?

The 64 and the 128 must be told both *how* to write information to the screen, and *where* to put it. Within a PRINT statement, if you press the SHIFT-CLEAR/HOME key inside the quotation marks, the computer will clear the screen—then move the cursor to the HOME position of the upper lefthand corner of the monitor. The notation for this sequence: PRINT "[CLR HOME]" is used a lot throughout this book. To make many of the programming tricks work, printing must be done on a clear screen or in a specific location.

A NEW PROGRAM

Enter the word NEW and press the RETURN key. NEW clears any old programs or statements with line numbers that may be sitting in the computer's memory. Now you have a clean start and can write a fresh program.

A common practice is to begin program line numbering at 10, and increase the increments in 10s as well. This gives you plenty of room to add more commands in the middle of a program, should you choose to do so later.

If you don't start a fresh program with the NEW command, any leftover statements will be intermingled with the program you are beginning to write; and most likely, nothing will work right. Too bad.

If you have never, never, entered a program into the computer before, try this. It will first clear the screen, then it will print your name in the upper lefthand corner:

```
10 PRINT "{CLR HOME}"
20 PRINT " (YOUR NAME) "
30 PRINT "IS A ROCKET SCIENTIST"
40 PRINT "AND JUST WON THE NOBEL PRIZE"
```

Enter the word RUN, and if you have no syntax errors, the program will work just fine. If you have made a mistake, it is necessary to go back and change something in the program. But where is the program? (It's not on the screen anymore.) It is currently being stored in Mr. Commodore's memory. To see it again you'll have to enter the BASIC keyword LIST and press the RETURN key.

MAKING CHANGES IN A PROGRAM

Changes can be made in an existing program by LISTing it on the screen, moving the cursor to the right spot, and simply writing over any existing information. When you're finished, press the RETURN key.

Or, on any blank portion of the screen, you can just retype the line, changing it anyway you like. The Commodore will automatically sequence the information into its proper place.

GET THAT LINE OUTTA HERE

To remove a line from the program, simply enter the line number and press the RETURN key. Any line in the program can be removed in this manner. It can be done any time, even if the program is not displayed on the screen.

PRINT FORMATTING

BASIC gives you the option of printing stuff line by line, or printing one line immediately at the tail end of the last. To make the computer connect the PRINT statements, each string of letters enclosed in quotation marks must be followed by a semicolon (;). This program uses semicolons to format the printed output:

```
10 PRINT "{CLR HOME}"
20 PRINT " (YOUR NAME) ";
30 PRINT "IS VICE PRESIDENT OF THE BANK";
40 PRINT "A CHAMPION ATHLETE";
50 PRINT "AND IS DOING WELL";
60 PRINT "IN LIFE"
```

When this program is run, Mr. Commodore will display each item adjacent to the last letter printed. Some additional editing will need to be performed, however, to correct sentence punctuation and spacing—because this doesn't look quite right:

```
CHARLIE JONESIS VICE PRESIDENT OF THE BA
NKA CHAMPION ATHLETEAND IS DOING WELL.IN
LIFE
```

THE PRINT ZONES

A comma (,) following a string of text tells the computer to “tab” over into the next “print zone” to begin writing the data to the display. The 40-column display of the 64 and 128 is four zones wide. This method of formatting program output is especially useful when working with numeric data that needs to be displayed in columns. . . .

```
NEW
10 PRINT "AGE",
20 PRINT "HEIGHT",
30 PRINT "WEIGHT"
40 PRINT 22,
50 PRINT "5-11",
60 PRINT 155
70 PRINT 19,
80 PRINT "5-4",
90 PRINT 120
RUN
```

And the screen will display:

AGE	HEIGHT	WEIGHT
22	5-11	155
19	5-4	120

Note that the numeric data did not need to be enclosed in quotation marks, but the heights that required a dash separating feet from inches to be printed did. If those values didn’t use quotes, the computer would display the *result* of lines 50 and 80, rather than the information exactly as written. The first height would be -6 (or 5 minus 11) and the second would be 1 because the computer assumed you wanted to print the result of 5 minus 4. Try it and see.

CRUNCHING PROGRAMS

It's possible, and often preferred, to combine multiple instructions on a single line. The first three lines of the program above can be crunched into one single line:

```
10 PRINT "AGE", "HEIGHT", "WEIGHT"
```

Although it looks more confusing, it's the economical way to write long programs. In the same way you can combine PRINT statements, multiple commands can be entered on a single line if you separate each statement with a colon (:). . . .

```
10 PRINT "{CLR HOME}"
20 PRINT "HELLO"
30 END
```

Is the same as:

```
10 PRINT "{CLR HOME}": PRINT "HELLO": END
```

MORE SPACE SAVINGS

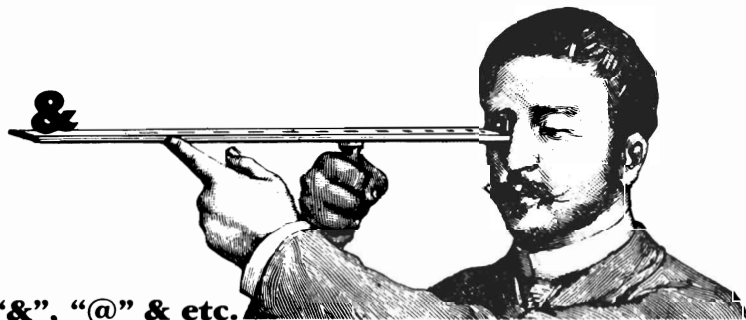
While we are into saving space, there is a not-very-secret abbreviation for the PRINT statement, and using it can save a lot of time. The question mark (?), when not enclosed in quotation marks, is always interpreted as the keyword PRINT. These two statements do the same thing:

```
10 PRINT "RULE 1": PRINT "DON'T GET MAD-GET EVEN!"
10 ?"RULE 1": ?"DON'T GET MAD-GET EVEN!"
```

MORE NOVICE TIPS

Shortcuts and abbreviations for keywords have no effect on how a program runs, but they do make life easier for the programmer. Other tips on how to keep things simple are:

1. Move to a tropical island.
2. Use the NEW command before writing a new program.
3. Clear the screen using the CLR HOME key before LISTing or RUNning a program.
4. Number your program by fives or tens to leave room to add more statements in between.
5. Remember, Mr. Commodore thinks everything he sees is a number and will try to do math with it, unless the letters, numbers, or symbols are enclosed in quotation marks. And data to be printed as-is *must* be in quotes.



“&”, “@” & etc.

All right, lookit, 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 *little pointy thing*.

BRIEF REM-ARKS

My sister never had much of an effect. She hung around a lot, REMinding me of things I might otherwise forget, and REMinding others of things I never wanted them to know in the first place. She would tell where I hid the B-B gun, and how she heard me making plans to sell my father's coin collection for Arnold Johnson's stuffed moose head. Her REMarks weren't always appreciated. When she starts talking I ignore everything thereafter.

There is a statement in BASIC that is quite similar, because it has no effect on the way things go. The REM statement is used in the body of a program to add REMarks or notes that may be helpful to you as the programmer. Everything on a program line after the REM command is ignored by the computer. Nobody even knows it's there, unless they LIST the program. The REM statements may be used for an introduction to the program, and in commercial software and utility programs, REMs are right up there at the beginning REMinding the world who wrote the program and what it does.

```
10 REM *****
20 REM * (C) 1987, A. JOHNSON *
30 REM * RARE COIN CLASSIFICATION PROGRAM*

40 REM *****
50 PRINT "{CLR HOME}"
60 PRINT "ATTENTION SPORTSMEN"
70 PRINT "MOOSE SEASON IS OPEN"
```

MORE REMS

REMs can also be used at the end of a line to explain the function of the statements within a program, or what certain variables stand for. REM is a very useful tool, and like Arnold Johnson, has little or no effect.

```

10 REM A PROGRAM TO DEMONSTRATE QUADRATIC FORM
20 PRINT X↑2+(5*X*Y)+Y↑2: REM A HOMOGENEOUS
  POLYNOMIAL
30 REM OF THE SECOND DEGREE
40 END

```

LET

If you tried to run the program above, the computer gave you the sad answer of zero. It did the math, but since X and Y had not been given a value, it multiplied and added zeros by zeros. In the program, X and Y are *variables*, or letters used to represent a numeric value. Unfortunately, nowhere in the program did we tell the computer what X and Y were equal to. Your computer did its best, but came up with goose eggs.

We can change this by using the LET statement. Add these lines to the quadratic program above and RUN the program to get a real answer of 141.

```

12 LET X = 4
14 LET Y = 5

```

ASSIGNED NUMERIC VARIABLES

Arithmetic calculations can be written into LET statements. Each of the following statements is an acceptable use of the LET keyword, and assigns a numeric value to the letter variables.

```

LET S  = 5+4
LET A  = 46*3
LET B  = 99/20
LET C  = A+B: LET D = B-A: LET E = C+D
LET D  = C-B+2
LET G  = G/(B+3)
LET QU = X↑2+(5*X*Y)+Y↑2: REM OUR QUADRATIC

```

LETLESS LETS

Forget about using the keyword LET. Nobody who is anybody uses LET. The command works whether you use the word LET or not—so let's just shorten the statements above to:

```
12 X = 4: Y = 5
```

STRING VARIABLES

Surprise. Letters, entire sentences, and graphic symbols can be assigned to a teenie-weenie variable. Mr. Commodore has just three rules for *string variables* that contain alphabetic information:

1. The variable name must be followed by a dollar sign.
2. The information must be enclosed in quotation marks.
3. If you don't follow string variable rules 1 and 2, a little man inside the computer will display the famous message:

```
?SYNTAX ERROR
```



GOOD STRINGS

These are good uses of string variable expressions. They have been tested at our factory, and certified by qualified inspectors.

```
P$ = "RATE OF PAY"  
E$ = "EAT AT JOE'S"  
BB$ = "BEAGLE BROS."  
HI$ = "HI MOM, SEND MONEY"  
A$ = E$ + HI$
```

MATH ATTACK-YAK-YAK

Many times, mathematical factoring is called for. If you were writing a payroll program for a small glue factory, it would be necessary to *multiply* an employee's rate of pay by the number of hours worked, *add* any benefit payments the company makes, then *subtract* the proper amount for taxes. The worker would then *divide* the amount on the check by the number

of hours worked to find out what he really made per hour—his net actual salary.

In writing the glue factory program to calculate payrolls, you'll have to use the first four symbols below before the check gets stuck in the envelope. As you look at the list, you'll see the 64 and 128 can do some real sticky math:

SYMBOL	MEANING
+	Add
–	Subtract
*	Multiply
/	Divide
n	Pi (the numeric constant 3.141592654)
↑	Exponentiation ($3 \uparrow 3 = 3 * 3 * 3$)
EXP(X)	Calculate the constant e (2.71828183) raised to the power of the number given.
SQR(X)	Returns the positive SQuaRe root of the argument given within the parentheses.
INT(X)	Gives the INTeger value of the variable. The INT function takes all the decimal digits off a number.
SIN(X)	Returns the SINE of a number, which must be expressed in radians. The value of COS(X) is equal to SIN($X = 3.14159265 / 2$).
COS(X)	This mathematical function calculates the COSine of the number, where the number is an angle given in radians.
TAN(X)	Returns the TANgent of the variable “X”, which must be expressed in radians. If the TAN function overflows, the error message ? D I V I S I O N B Y Z E R O is displayed.

SYMBOL	MEANING
ATN(X)	This mathematical function returns the ArcTanGent of the number. The result is always the angle in radians, whose TANgent is the number given. It is always in the range of $-\pi/2$ to $+\pi/2$
ABS(X)	Returns the ABSolute value of a variable, which is its value without any signs. The ABSolute value of a negative number is that number multiplied by -1 .
LOG(X)	Returns the natural LOGarithm (LOG to the base of e) of the argument. If the value of the argument is 0, or negative, the BASIC error message ? I L L E G A L Q U A N T - I T Y will be displayed.
RND(1)	Provides a RaNDom real number, greater than or equal to 0 and less than 1. RND(0) returns the most recently generated RaNDom number. Each negative argument generates a particular RaNDom number that is the same every time RND is used with that argument, and subsequent RNDs with positive arguments will always follow a particular, repeatable sequence. Every time RND is used with a positive number, a new RaNDom number from 0 to 1 is generated, unless it is part of a sequence of RaNDom numbers initiated by a negative argument.
SGN(X)	Returns -1 if the number is negative, 0 if the number equals 0, and $+1$ if the argument is positive.

A FIRST LOOK AT INTEGERS

Real numbers have a decimal point and decimal places (such as 6.932 or even 7.0), while *integers* do not. INTegers have had their decimal digits cut off. Now, you too, can ritualistically cut off decimal digits in your own home with the INT() command.

INT() is one of the most helpful statements on the 64 because it

is used to control the rounding of numbers. (On the 128 you can use the PRINT USING command as well.) This is essential when factoring dollars and cents because a number line \$34.99995505E-03 might be confusing.

How would you get such a number anyway? Try this problem:

Arnold Johnson worked 29 hours at a taxidermy shop. At the end of the week he was paid \$401.00. How much did Arnold earn per hour?

Answer: \$13.8275862 (per hour)

The answer can be rounded back with the INT() function. But if you try:

```
M = INT(13.8275862): PRINT M
```

the Commodore will give you 13. With a little more work, we can get Arnold what he deserves, including his 83 cents. Try the two rounding routines below when you need a trick to format a number:

```
10 REM LET Y = FIGURE TO BE ROUNDED
20 REM 2 DECIMAL DIGITS FOR MONEY AMOUNTS
30 Y = INT(Y*100+.5)/100: PRINT Y
```

or

```
10 REM LET Y = FIGURE TO BE ROUNDED
20 REM MAKE D = NUMBER OF DECIMAL DIGITS DESIRED
30 D = 4: Y = INT(Y*10^D+.5)/10^D: PRINT Y
```


CHAPTER 3

POINTERS FOR BEGINNERS

THE KEYBOARD

Most of the keys you see on the computer have been handed down from the ancient age of the typewriter. The understandable ones do the same things as typewriter keys. Getting uppercase or lowercase letters is a little different. On the 64 and 128, you switch between the two modes by pressing the COMMODORE LOGO key in the lower lefthand keyboard corner *and* the SHIFT key at the same time. This will toggle between the two text formats.

In uppercase/graphics mode, pressing a SHIFTEd letter key will print the graphic character on the front right of the keycap. In the lowercase/uppercase mode, pressing a SHIFTEd letter will display only capital letters. In either mode, holding down the COMMODORE LOGO key and pressing a letter will display the other special graphic symbol on the front left of the keycap.

The RUN/STOP and RETURN keys are on opposite sides of the keyboard so that they don't get bumped accidentally. Why? Because unless you have disabled those keys with special POKEs into memory, pressing the two will immediately stop any program that's running. It also clears the screen, but it doesn't hurt anything or clear the program out of memory. Think of these keys as "panic buttons." Don't, under any circum-

stance, however, press them while you're saving a program to the disk drive—it will make a mess of a perfectly good file.

The cursor-moving keys, of course, move the cursor. In 64 mode, you'll need to press a SHIFTed arrow key to move the cursor to the left or up the screen.

On the 64, the function keys, F1 through F8, won't do anything until you have programmed them to go crazy. There is a special character code and symbol assigned to each one. The little routine below will illustrate one way function keys function:

```
10 GET A$: IF A$ = " " THEN 10
20 IF A$ = "{F1}" THEN PRINT "THIS IS EASY"
30 IF A$ = "{F3}" THEN GOTO 100
40 PRINT A$: GOTO 10
100 PRINT "END OF PROGRAM"
110 END
```

Line 10 gets a character from the keyboard and assigns it to the variable A\$. If no key is pressed, A\$ equals nothing, so we go back to the beginning and check it again, and again, and again until a key is pressed.

Line 20 asks if A\$ is F1, and if so, prints the message.

Line 30 asks if A\$ is F2, and if so, sends the program off to a line near the end of the program.

And line 40 prints anything else you enter, and then begins the process over again at line 10.

MORE ON THE FUNCTION KEYS

On the 128, the function keys really do make life easier for a programmer. Here's a list of the standard features:

- F1** Enters one of the graphics modes, once you supply the mode number and press the RETURN key. Then you can use fancy graphics commands like BOX or PAINT.

- F2** Automatically prints DLOAD" on the screen. Then you'll need to type only a program name and the closing quotes to get a program off the disk. It saves six keystrokes.
- F3** Gets a directory of the disk in the drive and lists it on the screen. This sure beats the 64's LOAD"\$",8...[LIST]...blow-your-program-away sequence.
- F4** Wipes the screen clear with the SCNCLR command.
- F5** Prints DSAVE" on the screen, so you don't have to type all the letters yourself when saving a program in memory.
- F6** RUNs the current program in memory.
- F7** Automatically gives a screen LISTing of the program you're working on.
- F8** The key to the door for the machine language monitor.

REDEFINING THE 128's FUNCTION KEYS

All of the function keys on the 128 can be reprogrammed by using the KEY command. It can be done in the direct mode, as below, or from within a program. Use the routine below to make official Beagle Bros. function keys.

```
10 FOR X = 1 TO 8
20 KEY (X), "BEAGLE BROS."
30 NEXT
```

JUST SO YOU KNOW....

There are a lot of references in this book to “direct mode” things to do, and “program mode” things to do. To review, direct mode allows you to type just the command, and press the RETURN key. The computer executes the commands right away. In program mode, you must give the line numbers, and nothing will happen until you RUN the program. Okay, got it?

SYMBOLIC REVIEW

Remember that the keys shown below do these special things. Using them in the regular way can do irregular things to a program.

+	Add
-	Subtract
*	Multiply
/	Divide
↑	Exponentiation
\$	String variable
%	Integer variable
?	PRINT command
:	End of statement

THE LIST TRICK LISTING

F7	(128) Lists entire program
LIST	Lists the entire program
LIST 50	Lists only the line number given
LIST 50–70	Gives lines from 50 to 70, inclusive
LIST –60	Lists from the beginning up to line 60
LIST 100–	Lists from line 100 to the end of program
LIST [CONTROL]	Slows the display speed when CONTROL is pressed
LIST [RUN/STOP]	Stops the listing display
10 LIST	Makes LIST part of a program; when the program is RUN code will display
LOAD"\$",8 [LIST]	Gives a directory of what's on the disk

HELP ME

If you're using a Commodore 128 to try these tips and tricks, you'll be able to use the HELP command to proofread programs and catch your typing errors. If a program stops running and an error message displays, press the HELP key at the top center of the 128's keyboard. This will show the problem line on the screen and highlight the area of the mistake.

SAVE ME

Save programs on a disk or they will go bye-bye when the power goes off. If you are writing a long program it is a good idea to save your work every 15 minutes or so. The common syntax is:

SAVE "PROGRAM NAME", Ⓢ

The disk will whirr and the red light will go on. This means the drive is writing your program file. If everything goes okay, the light will go off and you can continue. If something goes wrong, the red light will flash on and off, or equally as bad, the drive will stop with the light still on.

**THE DRIVE'S RED LIGHT**

Don't panic if the red light signals trouble, because nothing is lost, yet. A flashing light indicates a the drive has a problem doing what you asked:

you get another try. The typical cause is 1) you tried to SAVE a program with a name that already exists on the disk, 2) no disk is in the drive or the drive's door is not closed, or 3) you are trying to LOAD a program from the wrong disk. Reset the drive by turning its power switch off and then on, and try again.

When the light stays on, the drive indicates that a file that should be closed is still open. Failing to close the program right away can corrupt the disk files. Typing this disk command will avoid the horror:

```
OPEN 15,8,15: CLOSE 15
```

SAVE AND REPLACE BUGS

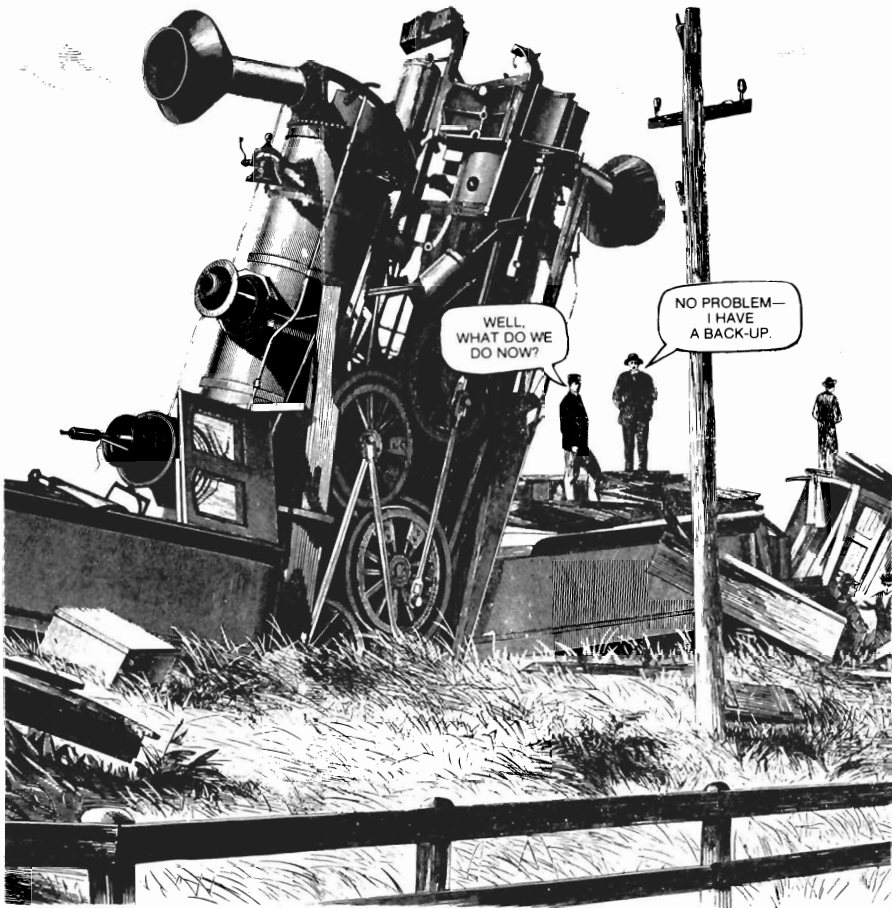
The Commodore SAVE@ command, which automatically saves one program and erases an old version with the same name, needs some bug swatting. Sometimes it also erases more than you might like, especially if the disk is nearly full. The safe way to save a file is to first erase the old version with the SCRATCH command; then save the new version. It's a few more keystrokes, but it is worth another five seconds to save five hours of programming.

```
OPEN 15,8,15,"S:PROGRAM NAME"  
CLOSE 15  
SAVE "PROGRAM NAME",8
```

SAVE AND VERIFY

The VERIFY command is used to make sure everything you sent to the disk arrived okay. It's a SAVE saver that can be squished right into the SAVE command line:

```
SAVE "PROGRAM NAME",8: VERIFY "PROGRAM NAME",8
```



EASY LOADING

If you're not using a 128 with its predefined F2 DLOAD key (or even if you are but want to try something different), here's a time saving trick for loading programs right off the directory.

1. List the directory of the disk.
2. Move the cursor up to the program wanted.
3. Type the word "LOAD" right over the file size numbers.
4. Move the cursor across the name, then type ,8:

AUTOMATIC LINE NUMBERING

Don't be a chump. Use the 128's automatic line numbering feature as often as possible. Just type AUTO in the direct mode. If you enter AUTO 10, the lines will be incremented by tens. If you type AUTO 1000000000, you're out of luck. The Commodore refuses to let you number your program lines in increments of a billion.

RENUMBERING

If you need to renumber all or part of a program, the 128 has as RE-NUMBER command built in. Like AUTO, it can be used only in the direct mode. RENUMBER won't let you increment your program lines by billions either.

MORE ON PROGRAM LINE RENUMBERING

Typing over a program line number with a new location will not erase the old statement. You'll just get a duplicate set of commands at each line number.

QUOTE MODE

Beagle Bros. quote of the month is, and we quote, "Quote mode is used to allow the entering of control characters, colors, and BASIC commands in thir shortest form—within quotation marks. Any time an odd number of quotation marks have been typed, many of the keys will print weird and funny graphic symbols instead of letters and numbers. The easy trip out of bizarre quote land is to press SHIFT-RETURN. Then you'll be able to cursor back up to the line and type like a civil being."

CHAPTER 4

CONVERTING PROGRAMS TO THE 128

THE BASIC MOVE

These tips are about how to write and modify programs; not hooking together an old Apple // with a new Commodore 128. We'll just assume it is usually a matter of making the basic move of keying the whole thing back in again.



Programs may look the same, but don't be so sure.

EASY TRANSLATIONS?

The measure of how easy it is to translate BASIC programs from the 64 or any other computer to the Commodore 128 is how little rewriting must be done. Certainly there should be no need to rethink the underlying logic of the program; especially at the level of how the algorithms influence the data. Look over the code and no matter how big or small, consider any program “convert-able” if the *order* of the programming commands does not need to be changed.

As it is, almost any simple program written for the 64 will run on the 128. All of the old commands are available and compatible with the new BASIC version 7.0. It doesn’t work the other way around, however, because there is lots of 7.0 syntax that will bring a 64 or other computer to a quick halt. PEEK and POKE addresses are different and won’t correspond. Same with SYS and USR commands. So use the list below to evaluate the chances for a quick and easy translation:

1. Is the vocabulary simple? Forget programs with commands that seem strange, or extensions to BASIC made with special software utilities. If the syntax of the commands feels familiar then it’s probably okay.
2. If there is frequent use of PEEK, POKE, references to memory-specific locations, or calls to built-in machine language routines, forget it. Programs like this are so specialized that it’s usually faster to rewrite the whole darn thing.
3. If the flow and logic of the program is understandable then take it on. It’s easy to reproduce things you can understand and impossible to figure out things you can’t figure out.
4. Is the program well documented? If the last programmer who worked on the code left notes and comments embedded in the program the job gets easier.
5. Is the programming structured? If so, good. If not, and the program is complex, good luck. It’s easy to rebuild a program with a modular design, but hard to unravel a mess of “spaghetti code.” (This is to remind you that unstructured, complex problems, have easy-to-understand *wrong* solutions.)

HOW TO MAKE PORTABLE PROGRAMS

Designing a program for portability comes at the expense of speed and memory. At Farmsoft, in Horsehaven, Iowa, they say portable programs are made to be plain, dependable work horses; not exotic thoroughbreds.

Use the tips below as programming guidelines for structuring code to write on a Commodore and possibly move to another computer or share with a friend.

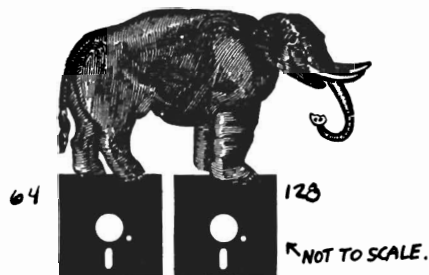
1. Put routines with special features in subroutines or modules that are separate from the main program.
2. GOTO and GOSUB should be followed by a single line number. GOTO 10 is okay, GOTO A is not.
3. Use as many constants as possible, and fewer variables. Like Tip 2, this trades speed for understandability.
4. Avoid PEEK, POKE, WAIT, SYS, FRE, and other memory-specific commands.
5. Embed lots of REM comment statements, including system limits, functions of the program parts, a table of variables, data structures, and the nature of error traps.
6. Limit IF . . . THEN statements to one logical operator and a simple line number after THEN. Complex operations on a true condition are acceptable to Commodore but may not be functional on other computers.
7. TAB(n) and SPC(n) commands are interpreted a little differently in every version of BASIC, as are some of the CHR\$ codes, so be careful.
8. Try not to do extensive string manipulations. If you must, place these functions in a separate module.
9. Leave plenty of space between line numbers for functions that may have to be broken into steps on the new version.
10. Limit yourself to a few commands or statements per line. Write only one complex function or compound evaluation per line.
11. Don't crunch the code by taking out all the blank spaces. It's easier to translate the first example below than the second:

```
100 IF F = 100 THEN PRINT "CORRECT"
100 IFF=100 THEN PRINT "CORRECT"
```

12. Avoid unprintable characters. Use the CHR\$ codes for control commands instead of the Commodore graphic symbols in quote mode. They will be easier to break out and translate.
13. Try to keep your list of commands and numeric functions to the following list—their syntax is common to almost every computer:

DATA	ON/GOTO	EXP
DIM	PRINT	INT
END	READ	LOG
FOR/NEXT/STEP	RESTORE	RND
GOSUB	RETURN	SGN
GOTO	AND/OR	SIN
IF/THEN	ABS	SQR
INPUT	ANT	TAN
NEXT	COS	

14. And, ahh-darn, avoid graphics; lo res, high res, or other res. A picture may be worth a thousand words—of additional programming.



JUST BETWEEN THE 64 AND 128

Watch your language when speaking with the list of reserved variables. For a while only TI, TI\$, and ST (time and status variables) were reserved. Then BASIC 4.0, DS, and D\$ (disk status variables) were added to the reserved stack. With the 128 we just reserved EL, ER, and ERR\$ (the error trapping variables).

BASIC 7.0 CAUTION

The commands below will save lots of programming on the 128. But, if used, they will add lots of conversion reprogramming on the 64. Avoid them!

```
BANK
BEGIN/BEND
BLOAD/BSAVE
BOX/CIRCLE/COLOR/GRAPHIC/PAINT/RCLR/RDOT/RGR/
SCALE/SCNCLR
BUMP/COLLISION/MOVSPR/RSPCOLOR/RSP0S/RSPRITE/
SPRCOLOR/SPRITE
CHAR
DCLEAR/DOPEN/DSAVE
DEC/HEX#
DO/LOOP/WHILE/UNTIL/EXIT
ELSE
ENVELOPE/FILTER/PLAY/SOUND/TEMPO/VOL
FAST/SLOW
FETCH/STASH/SWAP
GETKEY
GOTO
GSHAPE/SSHAPE
INSTR
JOY/PEN/POT
KEY
LOCATE
POINTER
```

PRINT USING/PUDEF
RECORD
RESUME/TRAP
RWINDOW/WINDOW
SLEEP
WIDTH
XOR

CHAPTER 5

BASIC TRICKS

PASSWORD TRICK #1

Password access is added to programs to keep the “wanna-useits” out of the picture. The sneakiness is in hiding the password in the program, so that nobody can find it by listing the software to the screen. This trick hides the password “BEAGLE BROS.” by printing CURSOR-DELETE characters immediately after listing the code word assigned to A\$:

```
10 A$="BEAGLE BROS.": REM"{SHIFT-RETURN}
```

After typing the quotation mark after the REM statement in line 10, press a SHIFTed RETURN key. Next, move the cursor back up to the position after the last quotation mark and press the INSERT key 15 times—then the DELETE key 15 times. A string of inverse letter Ts will be displayed on the screen. When you relist the program, BEAGLE BROS. will not show on the screen, although it's still there and still equal to A\$. Now continue on:

```
20 PRINT "{CLEAR HOME}"  
30 INPUT "PASSWORD"; X$  
40 IF X$ <> A$ THEN NEW  
50 REM BEGIN PROGRAM HERE
```

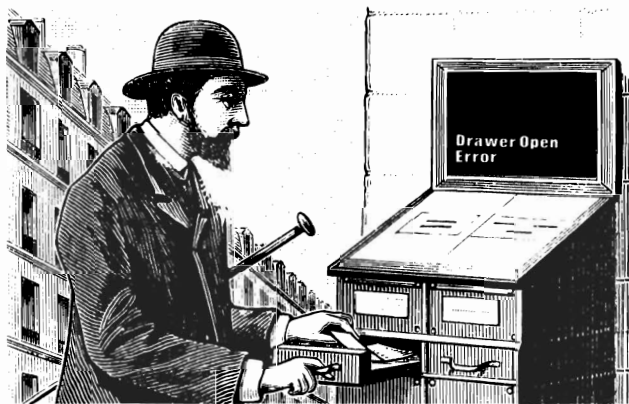
PASSWORD TRICK #2

This one uses a loader program and an identification number poked into a memory location.

```
10 REM LOADER PROGRAM
20 POKE 724,128: REM USE DELETE KEY TRICK #1 TO
  HIDE POKE
30 LOAD "MAIN PROGRAM".8
```

This main program first gives the user five attempts to get the password/identification number correct, then erases the whole disk. If you're not that mean, then change line 40 to print a nasty message, then NEW the program.

```
10 REM MAIN PROGRAM
20 PRINT "{CLR HOME}"
30 INPUT "ID NUMBER"; A
40 C = C + 1: IF C = 5 THEN OPEN 15.8,15,"S:*":
  CLOSE 15: NEW
50 IF A <> TO PEEK (724) THEN 20
60 REM BEGIN MAIN PROGRAM HERE
```



DISK LIBRARIAN

A single disk can be used to save a copy of the directories of all your program disks. You can make a library because the directory of any old disk can be saved just like any old program. It saves pulling every one of your disks out of its sleeve when you're hunting for an old obscure program.

Display a directory of each disk with the LOAD "\$",8 command, then list it to the screen. Last, insert the library disk into the drive and save the disk directory with a normal SAVE command syntax.

HEY! DON'T TOP OFF THAT DISKETTE

Beagle Bros. rule No. 65535 is never fill up your disks. Leave an amount of free space equal (at least) to the largest program stored there. That way, if future program modifications are needed, the new copy of the program can be saved *before* the old version is erased. "Safety first," we Beagles say.

AND SPEAKING OF SAFETY....

It's dumb not to periodically save a program in progress, at least after every 15 minutes of new work. So here's a good routine to add to the end of every program written. Every time a SAVE is needed, type GOTO 10000 in the direct mode.

```
9999  END
10000 OPEN 15,8,15,"S:PROGRAM NAME": CLOSE 15
10010 SAVE "PROGRAM NAME",8
```

HERE'S A BETTER WAY TO SAVE

For big programs under development, this built-in safety SAVE makes backup copies along the way. That way, if a new version introduces bugs that are hard to deal with, you can always go back to a previous pass.

Notice that the disk drive channel is open and closed every time, avoiding a lockup if a backup version doesn't exist yet. (Good thinking, right?)

This routine first erases the oldest backup version of the program on the disk (if there is one,) then renames the second copy to #3—the first backup copy becomes #2, and the previous “new” version becomes BACKUP #1. Last, the current version in memory is saved to the disk, any time you type GOTO 10000!

```

9999  END
10000 OPEN 15,8,15,"S:BACKUP #3":CLOSE 15
10010 OPEN 15,8,15,"R:BACKUP #2,BACKUP #3: CLOSE
      15
10020 OPEN 15,8,15,"R:BACKUP #1,BACKUP #2: CLOSE
      15
10030 OPEN 5,8,15,"R:PROGRAM NAME,BACKUP #1:
      CLOSE 15
10040 SAVE "PROGRAM NAME",8

```

ERASING PROGRAMS

The SCRATCH command erases a program from the disk directory. If, whoops, you erase a program by mistake, it's possible to get it back (oh boy!) if you do it right away—first thing—no delay—pronto—now.

LOAD "*",8 gets the last program accessed on the disk, usually even if the access was part of a SCRATCH routine. Once it's safely loaded back into memory, save it again and count your lucky stars.

MORE ERASE TRICKS

Here's a quick review of all the ways to get old junk out of the disk trunk.

```
OPEN 15,8,15,"N:DISKNAME": CLOSE 15
```

Erases the whole disk real fast, just don't use the disk ID number.

```
OPEN 15,8,15,"S:*": CLOSE 15
```

Deletes everything on the disk, sort of slowly, file by file.

```
OPEN 15,8,15,"S:A*": CLOSE 15
```

Deletes all files on the disk that start with the letter A.

```
OPEN 15,8,15,"S:AB*": CLOSE 15
```

Deletes all files on the disk that begin with the letters AB.

```
OPEN 15,8,15,"S:PROGRAM NAME": CLOSE 15
```

Erases only the specified program name from the directory.

```
OPEN 15,8,15,"S:PROGRAM 1, PROGRAM 2, PROGRAM  
3": CLOSE 15
```

Erases each of the three program names specified from the directory.
A fast way to get rid of more than one program with a single command.

RUN TIME TRICKS

If you're lazy, or have seen one too many syntax errors from RUNDY., use these tricks and you'll never have to type RUN again. . . .

1. Type any letter and then press SHIFT-RUN/STOP.
2. Press SHIFT-?-SPACE BAR (all three at the same time).
3. POKE 816,32 in 64 mode, then SHIFT-RUN/STOP.
4. On the 128, press F6.

In fact, the hot 64 tip is to add the POKE to the END statement line of a program under development. Then, for every one of the zillion test runs through the program you'll only have to press SHIFT-RUN/STOP:

```
9999 POKE 816,32: END
```

MORE EASY WAYS OUT OF WORK

Did-ya-know if-ya-need a string of program statements similar to one that already exists in a program, you can get the second line without typing it all in? Ha! The Commodore allows you to move the cursor up to the line to clone and type a new target line number right over the first one, and it doesn't alter the original. When the program is LISTed again, both lines are right there in their proper places, and the second one can be edited if needed.

MORE SPACE SAVINGS AND LESS WORK

We have heard from a reliable source (at the board level) that semicolons can be eliminated from many PRINT statements. Extensive testing has shown that all versions of the lines below produce the same laboratory results.

```
10 PRINT CHR$(66)CHR$(84)CHR$(77)CHR$(68)
10 PRINT CHR$(66) CHR$(84) CHR$(77) CHR$(68)
10 PRINT CHR$(66); CHR$(84); CHR$(77); CHR$(68)
```

or

```
20 PRINT A$B$C$
20 PRINT A$ B$ C$
20 PRINT A$; B$; C$
```

ONE MORE SPACE SAVER

When every last byte counts, or when one more character is one too many to type, eliminate the last quotation mark in your PRINT statements. The short lines below, again, have been scientifically proven to do the same darn thing.

shortest	10 ?"BEAGLE BULLETIN
shorty	10 PRINT "BEAGLE BULLETIN
short	10 PRINT "BEAGLE BULLETIN"

LONG LINES

Multiple statements can be put on the same line by separating each command with a colon: lines of up to 80 characters can be created this way.

But, if for some unknown reason you need to make a line longer than 80 columns, replace the fully spelled-out long form of the command or “keywords” with the shifted abbreviation tokens found in Appendix B. You might be able to gain up to ten or 15 extra character places this way. And it works fine. Once the line is listed, however, it can’t be edited, just retyped.

MORE THINGS TO ADD TO A PROGRAM UNDER DEVELOPMENT

In addition to doing a save-me and backup routine, it’s helpful to examine the directory every now and then to be sure everything is okay in disk land.

The bad news for some is the Commodore 64’s shortcoming—using LOAD"\$",8 trashes the current program stored in memory. Checking the

disk, therefore, means you'll have to load the program again to work on it more.

The good news is that we know how to get around this problem. Add these lines to the end of a program, then GOTO 20000 and LIST in the direct mode to see the disk directory. It's not real elegant, but it works real well. And viewing the directory won't turn your software into vaporware. . . .

```
19999 END
20000 PRINT"{CLR HOME} {23 CRSR DOWNS}"
20010 PRINT "POKE 46,PEEK(44)-1: POKE 44,8
           {HOME}"
20020 POKE 44,PEEK(46)+1
20030 LOAD "§",8
```

This will produce a clear screen with a POKE statement on the bottom, and a SYNTAX ERROR that, for once, we don't care about. Move the cursor back up to the top of the screen and type LIST. The directory will be displayed. Then move the cursor over the POKE statement line and press RETURN to get your old program back.

P.S. If you're in 128 mode, don't go to all this trouble. Pressing F3 does this better and with less effort.

RANDOM NOTES

Random numbers are used for all sorts of things. As a reminder (or maybe for a random thought), here's the usual format for making some semi-random numbers. The INTeger command is used to get rid of decimal places. The values can be changed to suit your needs and protect the innocent.

```
PRINT INT(RND(0)*16)
```

Prints a number between 0 and 15. (For people who think like computers and need to start counting at zero.)

```
PRINT INT(RND(0)*16)+1
```

Prints a random number between 1 and 16. (For computers who need to be like humans and start counting at one.)

```
PRINT INT(RND(0)*(200-150)+150)
```

Generates a random number between 150 and 199. Of course, the high and low values can be changed to get any range.

CHAPTER 6

SCREEN AND TEXT GRAPHICS

64 AND 128 GRAPHICS MODES

Beagle Bros.' testing agencies have determined that Commodore 64s and 128s have three (count them) modes of graphics output. The first is text or character graphics. These are the "regular" things that can be printed from the keyboard using normal PRINT statements and the symbols on the front faces of the keycaps. Each text character is made from a little eight-dot rectangular matrix, and the video output from character graphics will be adequately displayed on a color composite monitor or even a TV set (unless, of course, you choose to make 80-column text graphics on the 128. Eighty-column text on a TV set is beyond even the eyesight of an owl with a telescope).

This chapter is mostly about text graphics, screen placement, and cursor control.

The second type of graphics is high-resolution "bit-mapped" graphics. In other words, your programming must control every little bit: the teenie-weenie pixels (picture elements) on the screen. Pixel-by-pixel programming is done by means of the VIC (Video Interface Controller) chip, and its sister 8563 chip on the 128 (to help format its bit-mapped, 640 × 200 visual output). To really see these good resolution pictures at their best, you need a real RGB monitor.

Sprites are the small movable object maps (or blocks) that can be directed around the screen. Their movement can be individually controlled by BASIC and machine language commands on the 64, or better yet, the new sprite movement commands built into the 128. Standard sprites are 24×21 pixels square; multicolor sprites give up size for color data bits, measure 12×22 pixels. The Commodore *Programmer's Reference Guide* contains lots of good information on how to map and control sprites. It's recommended reading.



JELLY ROLL SCROLLS

This delicious message mover is good for getting your tasty point across the table. It's also a good way to learn how to do horizontal scrolling.

```

10 A$ = "THE JELLY ROLL"
20 C = 1: B$ = "    ": REM 20 SPACES
30 A$ = B$+B$+A$
40 PRINT "{CLR-HOME}" ; MID$(A$,C,40);
50 C = C+1
60 IF C > LEN(A$) THEN C = 1
70 GOTO 40

```

VERTICAL SCROLL CONTROL

When many records need to be displayed, it is possible to have more data than can be shown on Commodore's 25-line screen. This algorithm will keep the bottom records from scrolling the rest off the top.

```
10 PRINT "{CLR-HOME}"
20 FOR X = 1 TO 200
30 C = C+1: IF C = 23 THEN GOSUB 60
40 PRINT "SEE RECORD NO."X
50 NEXT: END
60 C = 0: PRINT "MORE"
70 GET A$: IF A$ = " " THEN 70
80 RETURN
```

WHERE'S THE CURSOR?

Another way to control scrolling is to locate the vertical line position of the cursor, and stop the printing if the line is near the bottom of the screen. With the 64 use:

PRINT PEEK (214)

To find the vertical line number.

POKE 214,1

To move the cursor to the top of screen.
If you change the 1 to a different value,
the cursor will go to that vertical line
number instead of to the home position.

CURSOR COMMUNICATION

Yet another way to place your cursor to a specific spot on the screen is to use a subroutine like this:

```
100 POKE 781,Y: POKE 782,X: POKE 783,0: SYS
    65520: RETURN
```

X is the column coordinate, Y is the row designation. Now in your 64 program you can type:

```
10 X = 12: Y = 15: GOSUB 100: PRINT "HI MEM.  
HOW'S SID?"
```

Of course this can be done on the 128 using the built-in CHAR function. Also, it will work on a bit-mapped screen as well as on a text screen. Why didn't they think of doing this before we did?

VTAB FOR THE COMMODORE

Because all old-dog Apple programmers can't learn new tricks, and anyone converting Applesoft programs to Commodore BASIC can sometimes get stuck because there is no normal Commodore equivalent for the VTAB command, we give you this trick. There will be a VTAB, however, if these lines are added to the beginning of the program:

```
10 DIM VTAB$(25)  
20 ROW$ = "{HOME}"  
30 VTAB$(0) = ROW$  
40 FOR X = 1 TO 25  
50 ROW$ = ROW$ + "{DOWN}"  
60 VTAB$(X) = ROW$  
70 NEXT
```

Then, anywhere following in the program you can type commands similar to Commodore's PRINTTAB (which replaces Applesoft's HTAB) in a fashion like this:

```
PRINT VTAB$(10); "HI MOM, I'M ON TV!"
```

A-MAZING SUBROUTINES

If you have a graphics printer, try this maze-making routine. Unfortunately there is no guaranteed solution to the puzzle it prints, or even starting and ending points. If you don't have a graphics printer just run line 20 and change PRINT#7 to a simple PRINT statement. This will get the program to print the maze on the screen, which is probably a better idea. We don't know a good application for this onscreen maze, and we don't recommend using a felt pen to draw on your monitor.

```
10 OPEN 7,4
20 PRINT#7, CHR$(161.5+RND(0));:GOTO 20
or
20 PRINT CHR$(161.5+RND(0));:GOTO 20
```

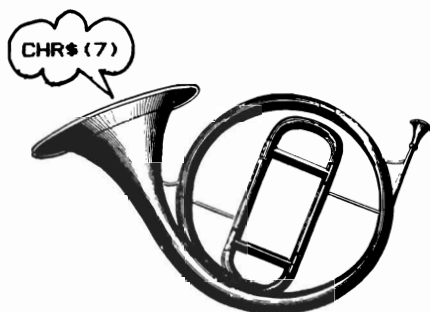
THE FRIED RICE MAZE

If you think the maze above is insane, try to watch while this maze is completed on your monitor screen—without going crazy. The Beagle Bros. cook thinks this looks like it's making a "diagonal maze" on the bottom of a frying pan out of little rice kernels ?!@??

```
10 PRINT "{CLEAR HOME}"
20 A$ = "{CRSR DOWN} {CRSR UP} {CRSR LEFT} {CRSR
  RIGHT}"
```

Reposition the cursor over the last quotation mark in line 20 and press the INSERT key four times. Then press the DELETE key four times. A reversed T character will print inside the quotation marks. Later this line will neither display properly nor allow editing. Now enter the remaining lines, and RUN like mad!

```
30 PRINT MID$(A$,RND(0)*8+1,1);
40 PRINT CHR$(109.5+RND(0));: GOTO 30
```



PRINTING COLORS AND BELLS

Yes, the color of items printed on the screen can be changed by including the *control code* for any one of the Commodore's 16 text colors in with the PRINT statement. After typing the opening quotation mark, press the CONTROL key or the COMMODORE LOGO key and a number from 1 to 8. This will place a graphic symbol, or *token*, in with the string to be printed.

In addition, on the 128, entering CONTROL-G within the quotes on a PRINT statement will make your bell ring.

Here's a chart you may find handy:

Key Combination	Result
CONTROL-1	black
CONTROL-2	white
CONTROL-3	red
CONTROL-4	cyan
CONTROL-5	purple
CONTROL-6	green
CONTROL-7	blue
CONTROL-8	yellow
COMMODORE LOGO-1	orange
COMMODORE LOGO-2	brown
COMMODORE LOGO-3	light red
COMMODORE LOGO-4	dark gray

Key Combination	Result
COMMODE LOGO-5	medium gray
COMMODE LOGO-6	light green
COMMODE LOGO-7	light blue
COMMODE LOGO-8	light gray
CONTROL-G	ding-dong

CHAPTER 7

MEMORY AND SPEED

FAST AND SLOW ON THE 128

The FAST command places the 128 into the 2-MHz operating mode, allowing you to run nongraphic programs in 80-column format at twice the normal speed. Unfortunately, disk input and output don't get faster. And it also turns off the 40-column screen, so there are no graphics and no switching between 40- and 80-column modes from within the program unless you issue the SLOW command first.

SPEEDING UP THE 1541 DISK DRIVE

You can speed up disk I/O operations considerably by making the screen go blank and sending a special command that puts the computer into the old "VIC" mode. Have a special "loader" program, or any program reading long data file from the disk, execute these statements:

```
10 POKE 53265,PEEK(53265) AND 239
20 OPEN 15,8,15,"UI-": CLOSE 15
30 LOAD "MAIN PROGRAM",8
```

When the file is loaded, reset the drive and turn the screen back on with:

```
100 POKE 53265,PEEK(53265) OR 16
110 OPEN 15,8,15,"UI+": CLOSE 15
120 REM THE REST OF THE PROGRAM
```

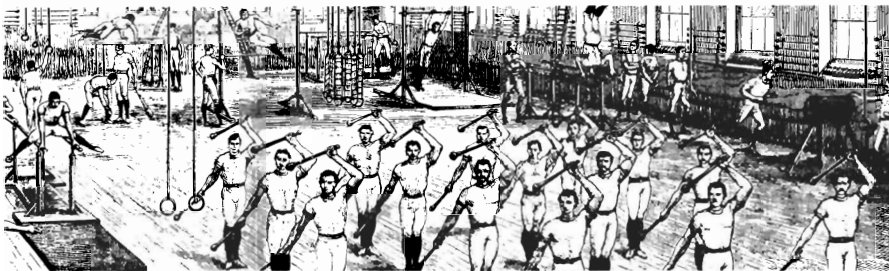
FAST SAVE AND BACKUP

A fast way to save a program and put two numbered backup copies on the disk is with this little algorithm. If you make this part of your program, type GOTO 10010 while in direct mode.

```
10010 FOR X = 1 TO 3
10020 P$ = "PROGRAM NAME"
10030 P$ = P$ + STR$(X)
10040 SAVE P$,8: NEXT
```

Or, in just direct mode, type:

```
FOR X=1 TO 3:P$="PROGRAM NAME":P$=P$+STR$(X):SAVE
P$,8:NEXT
```



BEAGLE BROS COMPUTER TRAINING FACILITY, SYNTAX, ARIZONA

MOST PROGRAMMERS WANT....

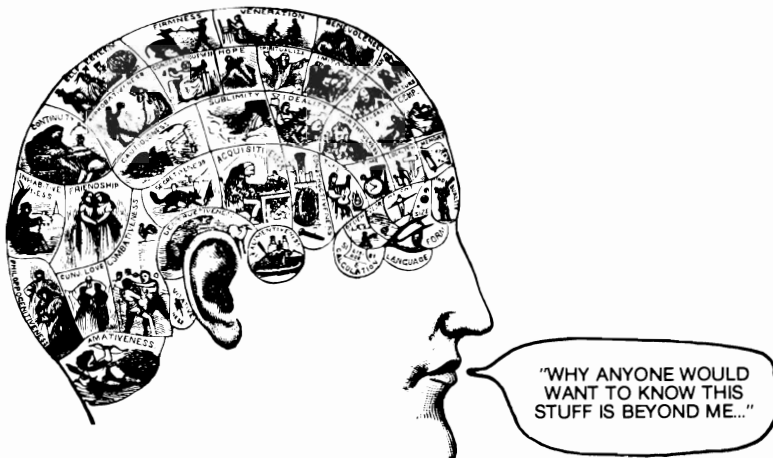
Most programmers want their programs to run faster and to occupy less memory. So most programmers should abide by these tips:

1. Plan ahead and try to program efficiently. Write down your major routines in English before you start coding. Draw pictures of your important screens. And structure the programming into clear modules.
2. Assign variables to frequently used constants. Assign the variables early in the program especially if you have a big number that is used many places in the code. Don't use:

```
FOR X = 1 TO 72
```

```
A = 1: B = 72: FOR X = A TO B
```

3. Place frequently used subroutines at the beginning of the program.
4. Use NEXT instead of NEXT X.



5. Use floating point variables for values that will be mathematically manipulated. Since all math is done in floating point, changing from an integer value takes time.
6. Use % to assign nonmathematic values integer status. They are then stored using less memory.

MORE SPEED AND MEMORY TRICKS

1. Renumber your program by ones starting at line 1. Line numbers are stored as strings, so the fewer digits in your GOTOs and GOSUBs, the less memory used.
2. Change all variable names to one character, two only if needed. It uses less memory.
3. DIM all arrays as small as possible.
4. Delete all REM statements in the final version of your program. If you need a copy of the software with comments, keep a separate working copy with embedded REM statements.
5. Use integer arrays if possible. Integer variables take up less room than real variables.
6. Get rid of extra command words. For instance, LET B = 72 can be expressed B = 72.
7. Don't use the END statement unless your program ends in the middle of the code.
8. Frequently assigned variables should be declared FIRST.
9. If you don't have to use arrays, don't.
10. Don't use TAB functions. They repeat the cursor-right command and are generally slow.
11. FOR . . . NEXT loops are faster than evaluated GOTO statement loop routines.
12. In a program with a stack of IF statements, place the decision with the highest probability at the top of the stack.
13. A forward GOTO—to a larger line number—is *much, much* faster than a backward GOTO.
14. Eliminate blank spaces between statements in the code. The two statements below do the same thing:

```
20 IF B = 72 THEN GOTO 100
20 IFB=72THEN100
```

15. Put as many statements on one line as possible, and separate the commands with colons. Up to 80 characters can be entered per line.

USING THE CLR COMMAND

The CLR command makes available all RAM memory that had been used but is no longer needed. Any BASIC program is untouched, but out go all variables, arrays, GOSUB addresses, FOR . . . NEXT loops, user-defined functions, and files. All their space is made available for new variables.

FORCED GARBAGE COLLECTION

Sometimes a program will seem to hang up for no apparent reason, then resume operation again. It is especially common in programs that assign lots of data to string variables. The computer stops to do “garbage collection.” This means that it has temporarily run out of memory and needs to check and then clear RAM that has stored temporary values. You can force the computer to do garbage collection before it runs out of memory by using the FRE command at a point in the program where speed is not important.

FRE by itself will return the value of bytes available for BASIC’s use. On the 64, the argument 0 should be used. On the 128, use the argument 0–5 to point to the specific memory bank to be evaluated. Examples:

```
PRINT FRE (0)
PRINT FRE (1)
IF FRE (0) < 100 THEN PRINT "WE'RE RUNNING OUT OF
    GAS, MOMMA"
```

128 DCLEAR

The statement below on the 128 closes all the files and clears open channels on the specified device number. It's a quick way to get things done.

DCLEAR D0 This statement clears every open file on drive 0,
device number 8.

BASIC ABBREVIATIONS

Appendix B in the back of the book lists a whole bunch of typing abbreviations for every single BASIC keyword in both the 64 and 128 systems. Even though you may type only two letters, the entire keyword will appear in its correct syntax when the program is relisted.

GREEN GO, RED WAIT

A good way to wait for a keypress is with the WAIT statement. It really causes all program execution to be suspended until a given memory address recognizes a specific bit pattern. WAIT is commonly used as part of a GETKEY subroutine like:

```
900 POKE 198,0: WAIT 198,1: GET A$: RETURN
    (64 mode)
900 POKE 208,0: WAIT 208,1: GET A$: RETURN
    (128 mode)
```

CHEAP AUTOMATIC LINE NUMBERING

CLEAR-HOME the screen and use this routine in the direct mode to quickly put program line numbers on the screen. Then just type in the statements and press RETURN.

```
FOR X = 0 TO 200 STEP 10: PRINT X: NEXT
```

EXOTIC AUTOMATIC LINE NUMBERING

Automatic line numbering is such a good idea that it was built into the 128. The AUTO (n) feature can be used; every time you press RETURN, the next line number is printed on the screen and the cursor appears two spaces to the right.

ABBREVIATING ZERO

The value of zero can be abbreviated by a decimal point. The Commodore will process the reading of a decimal point faster than the character "0". It can be used to assign variables such as A=0, but not as part of another number such as 9000. Try this in the direct mode.

```
PRINT . . . . . {RETURN}
```

REPEATING KEYS

POKE 650,128 in 64 mode makes all keypresses repeat the character pressed. This is especially fast and helpful when designing graphics using the graphics characters displayed on the keycaps; you can just hold the key down instead of pounding it many times in a row. When you're done, POKE 650,0. Back to normal.

DON'T STOP, DON'T STOP

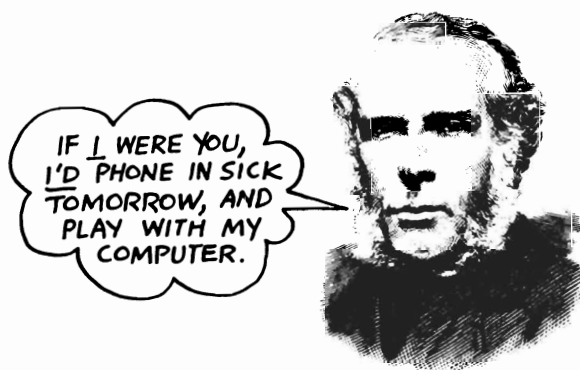
Another way to be sure that our program will keep going as fast as it can is to disable the STOP key. POKE 808,234 does both RUN/STOP and RESTORE. POKE 808,237 to bring the computer back from la-la land.

USING THE ON STATEMENT

It's a lot faster to execute the ON statement than evaluate a list of IF... THEN conditions. The two routines do the same thing. The second one requires less computer processing.

```
100 GET A$: A = VAL(A$)
110 IF A = 1 THEN 100
120 IF A = 2 THEN 200
130 IF A = 3 THEN 300
140 IF A = 4 THEN 400
150 IF A = 5 THEN 500
160 GOTO 100
```

```
100 GET A$: A = VAL(A$)
110 ON A GOTO 100, 100, 300, 400, 500: GOTO 10
```



DELAY LOOPS

If your program has many delay loops composed of empty FOR . . . NEXT counters, put them at the beginning of the program as a subroutine. This will save dumb, redundant code.

```
100 FOR X = 0 TO 4000: NEXT: RETURN  
110 FOR X = 0 TO 8000: NEXT: RETURN  
120 FOR X = 0 TO 16000: NEXT: RETURN
```

CURSORS ON SPEED

Make the 64's cursor go real fast with this POKE. Values near zero make the cursor go fast; near 255 makes it move extra slowly.

```
POKE 56341,0
```

MORE ON CURSORS

Try POKE 214,30, then hold down the cursor key. It sends the 64's cursor to its limit.

WHEN ALL ELSE FAILS

Use machine language subroutines to do the time-consuming functions.

AND DON'T DO THIS

You'll see the ocean if you POKE 0,0.

CHAPTER 8

USEFUL APPLICATIONS

SIMPLE SORTING

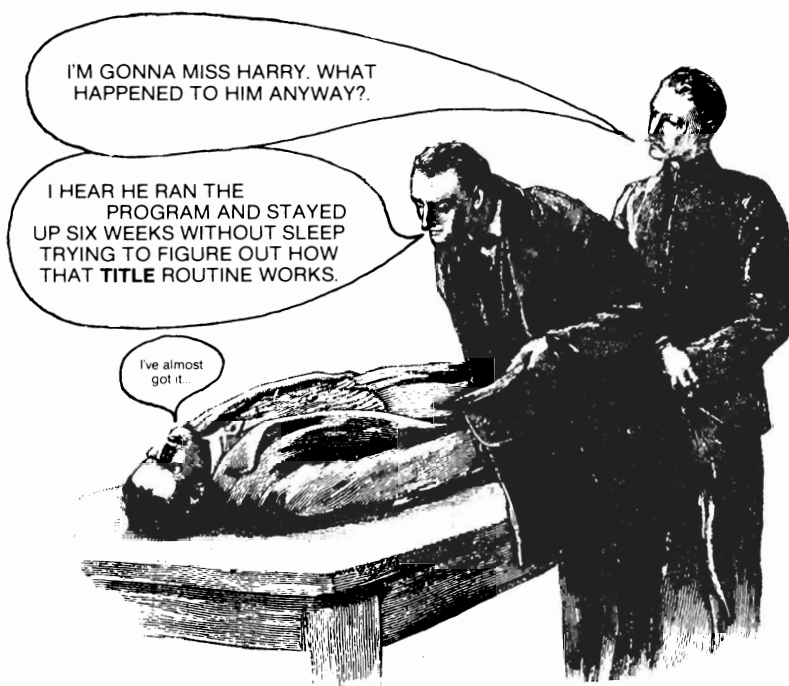
This routine gives you a real simple sort. Lines 10–40 establish the number of things you wish to put in order, then get their “names.” (This sort would work the same if it were acting upon elements from a DATA statement or reading records from a file.) The actual sort begins at line 50. It can be made smaller by combining the commands—thereby crunching it down into fewer lines.

```
10 INPUT "NUMBER OF ELEMENTS"; N
20 DIM A$(N)
30 FOR X = 1 TO N
40 PRINT X;: INPUT A$(X): NEXT
50 FOR A = 1 TO (N-1)
60 B = A: FOR X = (A+1) TO N
70 IF A$(X) > A$(B) THEN 90
80 B=X
90 NEXT
100 Z$ = A$(A)
110 A$(A) = A$(B)
120 A$(B) = Z$
130 NEXT
140 FOR X = 1 TO N
150 PRINT A$(X): NEXT
```

MESSAGE FLASHER

This programming trick will add some visual punch to screen displays such as the old, typical, "Press any key to continue" message. It makes your important information flash on or off in one second intervals. Line 1010 looks at the last digit of the time clock, then uses the AND statement to reduce the second counter to odd (1) or even (0)—and changes the reverse print flag as required.

```
1 GOSUB 1000:END
1000 PRINT "WARNING, REACTOR MELTDOWN! {RVS-ON}
      {CRSR-UP}"
1010 PRINT IF (VAL(RIGHT$(TI$,1))AND1)=1 THEN PRINT
      "{RVS-OFF}";
1020 GET A$: IF A$ = " " THEN 1000
1030 PRINT "RVS-OFF": RETURN
```



CENTERED SCREEN PRINTING

A routine that will center titles, headlines, and important messages, regardless of length, is always helpful. It can even be combined with the flashing message routine above to get a really splashy effect.

This trick is designed for a 40-column screen. If you're working in 80 columns, or a smaller text window on the 128, change the number in line 20 to match the number of character spaces across the allowed area.

```
10 INPUT A$
20 PRINT TAB((40-LEN(A$))/2) A$
```

CENTERED PRINTER PRINTING

Here's a variation on the center-me theme above, arranged for symmetrical output to a symphony of printers.

```
10 INPUT A$
20 OPEN 7:4
30 PRINT#7 SPC((80-LEN(A$))/2) A$
40 PRINT#7: CLOSE 7: END
```

LINE UP DECIMAL POINTS

It's always nice to see something displayed in an orderly manner. Sometimes it's even a requirement. With a series of numbers (up to four digits) this will produce a line of decimal points so straight that any neurotic neat-nick will admire it.

```
10 FOR X = 0 TO 1000 STEP 10.5
20 PRINT SPC(ABS(X<1000)) SPC(ABS(X<100))
   SPC(ABS(X<10)) X
30 NEXT
```

RIGHTHAND JUSTIFIED NUMBERS

We use this routine because it's an easy trick to righthand justify numbers. On their own, numbers try to lefthand justify themselves. Used as a subroutine, this will do a number on all your numbers.

```

10 INPUT A: GOSUB100: END
100 A$ = STR$(A)
110 FOR S = 1 TO 15-(LEN(A$))
120 A$ = "{SPACE}"+A$
130 NEXT
140 PRINT A$: RETURN

```

DOLLAR FORMATTING

There are a lot of ways to make change for a dollar, but in a program that must manipulate money values mathematically, the answer must be returned with both cents digits in place—no less and no more. That's a fact that can't be changed. The Commodore, however, will round the amount to the shortest display it can, and chop off all the pennies if possible. Or a user may input a value with too many decimal digits, or a negative number, or zero. This trick will fix anybody that tries to trick your money handling routines. We count on it.

P.S. Notice how the dollar value goes in as a numeric variable (D) and comes out as the string (D\$).

```

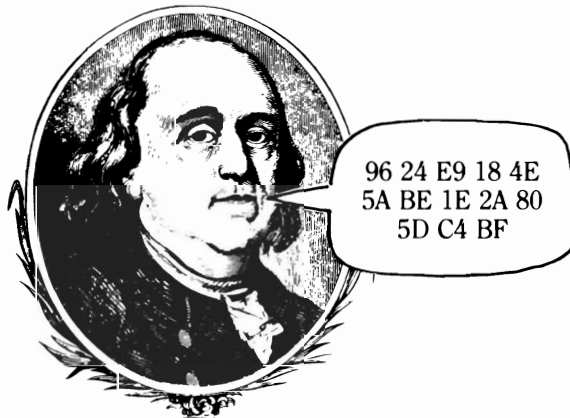
10 INPUT D
20 D = (INT(D*100+.5))/100
30 D$ = STR$(D)
40 D$ = "000"+D$
50 IF MID$(D$, (LEN(D$)-1), 1) = "." THEN D$ = D$+"0": GOTO 70
60 IF MID$(D$, (LEN(D$)-2), 1) <> "." THEN D$ = D$+".00"
70 D$ = RIGHT$(D$, LEN(D$)-3)
80 PRINT D$

```

VENDING MACHINE CHANGE

For a programming change, here's a change-making program. If entered correctly, it gives the correct change to be returned when a dollar bill is used to make a purchase of less than a dollar. Notice how line 50 gives you your money's worth, by handling the possibility of more than one dime in change. We like that.

```
10 INPUT "AMOUNT OF SALE"; S
20 IF S < 1 OR S > 100 THEN 10
30 C = 100 - S: IF C > 50 THEN H = 1: C = C - 50
40 IF C >= 25 THEN Q = 1: C = C - 25
50 IF C >= 10 THEN D = D + 1: C = C - 10: GOTO 50
60 IF C >= 5 THEN N = 1: C = C - 5
70 P = C
80 PRINT H, "HALF-DOLLAR"
90 PRINT Q, "QUARTER"
100 PRINT D, "DIMES"
110 PRINT N, "NICKEL"
120 PRINT P, "PENNIES"
```



QUICK AMORTIZATIONS

This program will provide equally divided monthly payments to a bail bondsman. For the rest of us, it's also good for estimating car loan payments, home mortgages, and student loan paybacks.

```
10 INPUT "AMOUNT"; A
20 INPUT "YEARS"; Y: M = Y*12
30 INPUT "APR"; R: R = R/12/100
40 A = INT(100*A*R/(1-1/(1+R)M)+.5)/100
50 PRINT "EACH PAYMENT IS"; A
```

CHAPTER 9

PROTECTION

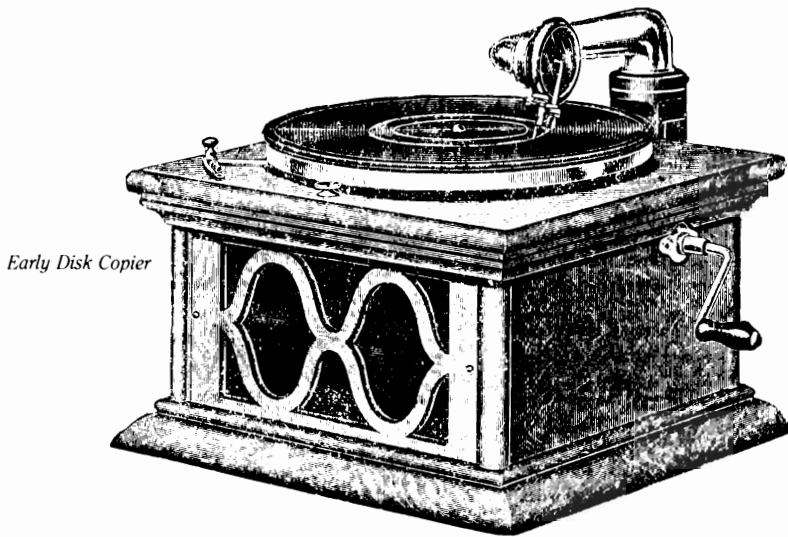
“You can’t stop all of the people from copying a disk all of the time, but you can slow some of them down some of the time.”

R. T. W. Sector
from the *Apple Big Tip Book*

HOW TO MAKE A MILLION

If you want to quit your job and make a million dollars selling software, you need only sell 2,000 copies of a program at \$500 each to reach the golden goal. It sounds simple, right? You just need the right program and it’s off to the Bank of the Bahamas. Unless, of course, someone rips off your good program code and makes common knowledge of all those algorithmic ideas.

So let’s talk protection. Copy protection, password protection, LIST protection, SAVE protection, INPUT protection, LOAD protection, STOP protection, RUN protection, and more. (Like protection from the mob, and protection from wild bears.)



Early Disk Copier

DISK COPY PROTECTION

Commodore disks are one of the hardest computer formats to copy protect. It's because DOS is built into the system at the chip level and it's not very accessible.

In other systems like Apple and IBM, DOS is easy to get at because it's a separate program on disk—it can be disassembled and messed with. This makes it easier to confuse the very disk operating system that will be needed to copy an entire disk.

Most Commodore disk protection schemes (the method to keep someone from copying the entire disk; not just a program file), involve putting an "error" on the disk. In its best form, the error will be slight and *will not copy* onto a duplication disk. In other words, the disk will work if the error is present, and won't if the error's not there.

What's the trick? The main program or a special loader program on the disk will instruct the drive to open and read the file with the error at the known location on the disk. A DOS error condition will occur and the drive red light may flash. The software then opens a channel to the disk and reads the error condition:

```

9000 OPEN 5,8,15
9001 INPUT#15, A$, B$, C$, D$
9003 CLOSE 15

```

This command will find:

A\$ The error number
 B\$ The error description
 C\$ The track number where the error occurred
 D\$ The block/sector location of the error

The program goes on by evaluating the variables A\$, B\$, C\$, and/or D\$. If everything is okay with DOS, then A\$=00, B\$=OK, C\$=00, and D\$=00. But since we are looking for the error and not "00", typical copy protection code might read:

```

9004 IF A$ = "00" THEN PRINT "Y0-THIEF": NEW

```

Remember this works only if the error cannot be copied onto another disk. Most commercial Commodore disk copying programs these days copy everything—errors included. And lots of utility software is around that will put “errors” onto disks for you, to help you make your own protection scheme, or (no-no of no-nos) build your own bootleg copy of pirate software. Either way, here’s a list of the most common disk protection “errors”:

Error No. (A\$)	Error Type (B)
21	READ ERROR (no sync character)
23	READ ERROR (block checksum error)
20	READ ERROR (no block header)
27	READ ERROR (header checksum error)
29	READ ERROR (disk ID mismatch)
22	READ ERROR (data block not present)

P.S.

POKE 157,0 disables all error messages in the 64 mode, POKE 157,64 turns them back on.

OPERATION PROTECTION

Copy protection is one way to make a million, the other way is to make sure your bestselling program can't be crossed up while in operation. It ain't good software if it's broken, and it ain't gonna get broken if it's protected.

To make sure the program does what it's supposed to, all the way to the end, disable the 64's RUN/STOP and RESTORE keys. This also prevents major damage that comes from pounding on the RUN/STOP-RESTORE keys during disk I/O operations.

POKE 788,52	Disables STOP
POKE 788,49	Enables STOP
POKE 792,193	Disables RESTORE
POKE 792,71	Enables RESTORE
POKE 793,203	Disables RESTORE
POKE 808,234	Disables RUN/STOP-RESTORE
POKE 808,225	Disables RUN/STOP-RESTORE and LIST
POKE 808,239	Disables STOP
POKE 808,127	Disables RUN
POKE 808,237	Enables RUN/STOP-RESTORE

NO KEYBOARD

Another good way to keep a program running is to turn off the 64 keyboard, because once it's disabled, turning off the power becomes the last resort.

POKE 649,0	No keyboard input
POKE 649,10	Enables keyboard

PROTECTING THE IMAGE OF YOUR CHARACTERS

A character switch should always be used in the beginning of a program to set the display to uppercase/graphics or lowercase/uppercase. The SHIFT-COMMODORE keys only toggle between character sets in the direct mode. Once your program is going, the SHIFT-COMMODORE key combination is close to useless.

POKE 53272,23	Switches to uppercase/lowercase
POKE 53272,21	Switches to uppercase/graphics
PRINT CHR\$(14)	Switches to uppercase/lowercase
PRINT CHR\$(142)	Switches to uppercase/graphics

Also:

POKE 657,1	Disables SHIFT-COMMODORE
POKE 657,0	Enables SHIFT-COMMODORE

SHIFTY BUSINESS

Once a character set has been set, shut off the SHIFT keys. That will help protect your protection.

PRINT CHR\$(8)	Disables the SHIFT key
PRINT CHR\$(9)	Enables the SHIFT key

KEY REPEATS

Sometimes you may want a key pressed down to repeat its input, sometimes not. Most word processors repeat keys; single-keypress menu driven programs, for safety, do not. When you first power on your 128 or 64 there is a normal mixed bag of keypress response. To change it, type:

POKE 650,64	No keys repeat
POKE 650,128	Repeat on every keypress
POKE 650,0	Normal repeat on cursor and control keys only

LOOK AT THIS LIST LISTING

It's nice to keep things private. Especially if your program has built-in copy protection that you don't want defeated, or some tricky programming, or some proprietary data embedded in the code. Here's a list of LIST tricks.

POKE 774,131	Disables LIST and locks up system
POKE 774,164	Disables LIST, no lock up
POKE 774,226	Disables LIST and clears screen
POKE 774,26	Enables LIST
POKE 775,252	Disables LIST and clears screen
POKE 775,168	Disables LIST
POKE 775,191	Disables LIST
POKE 775,200	Disables LIST
POKE 775,167	Enables LIST
POKE 808,225	Disables LIST; upsets character color and display
POKE 808,32	Enables LIST
POKE 808,237	Enables LIST
POKE 774,1	Displays listing of first line number only
POKE 774,0	Displays listing of every line number

REM SHIFT-L Produces SYNTAX ERROR message and halts listing

POKE 2049 Upsets pointers to lines

Enough???



NOW FOR THE SAVE COMMAND....

If you liked the LIST list, save these 64 SAVE pokes:

POKE 818,180	Disables SAVE (a personal favorite)
POKE 818,32	Disables SAVE

POKE 818,131	Disables SAVE
POKE 818,164	Disables SAVE
POKE 818,226	Disables SAVE and clears screen
POKE 818,252	Disables SAVE and clears screen
POKE 818,237	Enables the SAVES above
POKE 819,246	Disables SAVE
POKE 819,245	Enables SAVE

ULTIMATE INSURANCE

The best way to disable the SAVE command that we have ever seen is to put a piece of tape over the write-protect notch on the disk. It is an original Beagle Bros. idea.

UNLOADING PROTECTION

These POKES will turn off—and back on—the computer's ability to LOAD anything in 64 mode, even a directory.

POKE 816,157	No LOAD, not ever
POKE 816,165	Enables LOAD

DIRECTORY PROTECTION

There are even of ways to put control characters into file names to confuse the disk directory display. Chapter 12 has the Beagle Bros.' list of good file/directory protection tips.

MYSTERY POKE

Since the Commodore has so many PEEK holes, another protection scheme is to poke a mystery value into a location with a “loader program” and the PEEK at the location from the “main program” to see if the value is correct. For example:

```
5 REM ** LOADER PROGRAM **
10 REM {SHIFT-L}
20 REM LINE 10 DISABLES LIST SO NOBODY CAN SEE MY
   POKE
30 POKE 728,124
40 LOAD "MAIN PROGRAM", 8

5 REM ** MAIN PROGRAM **
10 IF PEEK(728) <> 124 THEN 6000
20 REM PROGRAM STARTS HERE
5999 END
6000 REM NERD TRAP #1
6010 PRINT "YOU NERD!"
6020 NEW
```

FOR WARPED MINDS

If you really want to get a disk copier, here's another rotten trick to be used with the “mystery POKE” method.

```
6000 REM NERD TRAP #2
6010 PRINT "REMOVE WRITE-PROTECT TAB"
6020 PRINT "PRESS ANY KEY TO CONTINUE"
6030 GET A$: IF A$ = " " THEN 6030
6040 OPEN 15,8,15, "S:*"
6050 CLOSE 15
```

INPUT CRASH PROTECTION

A user can crash a program that uses simple INPUT statements by just pressing RETURN. This input protection routine avoids a bunch of bad empty string problems.

```
10 INPUT "QUESTION HERE {SPACE} {SPACE} {SHIFT-  
    SPACE} {3 CURSOR LEFT}"; A$  
20 IF A$ = "{SHIFT-SPACE}" THEN 10
```

INPUT LETTERS ONLY

If you want to limit the input above to *letters* only, add this line:

```
30 A = ASC(A$): IF A < 65 OR A > 90 THEN 10
```

INPUT NUMBERS ONLY

If you want to limit the input above to *numbers* only, add this line:

```
30 A = ASC(A$): IF A < 48 OR A > 57 THEN 10
```

INPUT PROTECTION FOR LETTERS AND NUMBERS

If you want to limit the input above to letters and numbers only, with no punctuation, and no weird Commodore control or graphics characters, then use these lines of Big Tip code.

```
30 A = ASC(A$): IF A < 48 OR A > 90 THEN 10  
40 IF A > 57 AND A < 65 THEN 10
```

CHAPTER 10

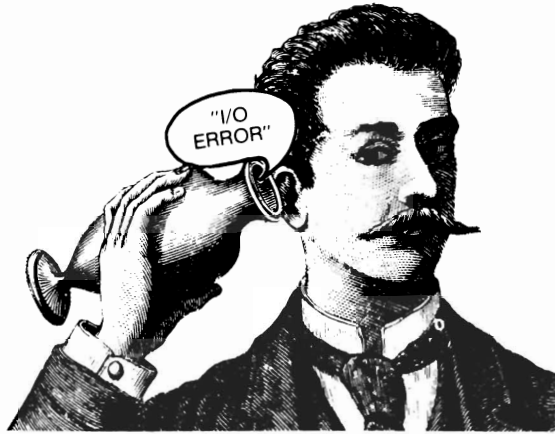
ADVANCED PROGRAMMING TRICKS

NEVER READY

The one prompt that we see more than anything else on the 64 is the READY prompt. Well here's a way to make sure you'll never see that word again. In the following algorithm, A\$ = BEAGLE BROS., and forever more (or until you press RUN/STOP-RESTORE,) our favorite name will replace the ho-hum READY prompt.

The first lines take a few seconds to execute, because they PEEK at BASIC in ROM, and POKE it back into RAM where we Beagle Bros. can mess with it.

```
10 FOR X = 40960 TO 49151: POKE X,PEEK(X): NEXT  
20 POKE 1,PEEK(1) AND 254  
30 A$ = CHR$(158)+"BEAGLE BROS."+CHR$(154)  
40 A$ = CHR$(13)+CHR$(13)+A$+CHR$(13)+CHR$(0)  
50 FOR Y = 1 TO LEN(A$): POKE  
    41845+Y,ASC(MID$(A$,Y,1)): NEXT
```



ELIMINATION OF ERRORS

Now, if you're READY, we can eliminate the word ERROR from all your error messages. It will have to be replaced with another word (in B\$). We have used "SCREW-UP"—which, of course, we never do. . . .

```
10 FOR X = 40960 TO 49151: POKE X,PEEK(X): NEXT
20 POKE 1,PEEK(1) AND 254
30 B$ = " SCREW-UP"+CHR$(0)
40 FOR Y = 1 TO LEN(B$): POKE
   41832+Y,ASC(MID$(B$,Y,1)): NEXT
```

MULTICOLOR LISTING

Designer's delight! Make a listing of your program print on the screen in any little-old-color that Picasso could choose. If you like, use this trick with each of your structured programming modules, to highlight different areas of the code.

```
10 REM" {PRESS SHIFT-RETURN}
```

Then, for purple printouts, reposition the cursor next to the quotation mark and press:

```
{CONTROL-9} {SHIFT-M} {COM LOGO-MINUS SIGN}
{RETURN}
```

When the program is listed, everything after line 10 will be as pretty as purple can make it. If purple isn't your color, then substitute another keypress for SHIFT-MINUS SIGN, and paint your listing to death. Just be sure it's printed in reverse characters, and it follows a reverse SHIFT-M.

Key Combination	Color
SHIFT-P	black
E	white
£	red
COMMODORE LOGO-ASTERISK	cyan
COMMODORE LOGO-MINUS SIGN	purple
UP ARROW	green
LEFT ARROW	blue
SHIFT-UP ARROW	yellow
SHIFT-A	orange
SHIFT-U	brown
SHIFT-V	light red
SHIFT-W	dark gray
SHIFT-X	medium gray
SHIFT-Y	light green
SHIFT-Z	light blue
SHIFT-PLUS SIGN	light gray

STRUCTURED LISTINGS

One of the best ways to identify parts of a program in a large software project is to “structure” the programming by breaking the tasks into independent modules. Then, for instance, when a sort is needed, you just

GOSUB to the sort routine; or GOSUB to the get-a-character-and-screen-the-input routine; or GOSUB to sound-Bluah-the-error-buzzer routine.

The bad news is that the modular parts of a program all run together when the program is listed . . . unless you use this Beagle Bros. Structured Listing Trick as a REM statement at the beginning of each module:

```
100 REM" {PRESS SHIFT-RETURN}
```

Then reposition the cursor next to the quotation mark and press:

```
{CONTROL-9} {SHIFT-M} {Q} {Q} {CONTROL-0} ANY  
WEIRD MODULE NAME
```

In the above code, CONTROL-9 turns on reverse characters, reverse SHIFT-M forces the execution of any following commands, reverse Qs cause carriage returns, CONTROL-0 brings us back from reverse printing and, presto—your module header prints on top of the next block of the listing.

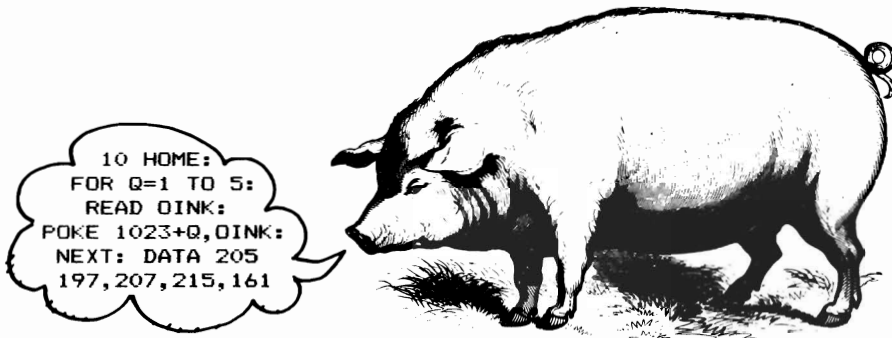
NEW CONTROL CODES

The CTRL key has many useful functions. Here are some suggestions for Commodore—are you listening???—and for the next computer in line (which could be named the C-256, the Admiral, or better yet, the Commodore's Daughter).

CTRL-D	Destroys the program in memory—forever
CTRL-T	Turns the monitor into a full-featured television
CTRL-S	Scrambles the keycaps
CTRL-V	Changes computer to a vacuum that cleans a room
CTRL-E	Electrocutes anyone touching the keyboard
CTRL-B	Turns the system into a six-pack of beer
CTRL-\$	Makes the printer print money

GOTO NOWHERE

The command GOTO can be used without a line number. Actually, it will command the program to jump to line 0 (zero). But, if you don't have a line 0 it won't work. And, if you never need to GOTO the beginning of a program, you won't care.



MOOSE CALLS

Every good program, and every bad program, should have a moose call routine. It is the one element that links natural languages and computer science. This algorithm calls for turning up the volume, and turning on every moose in the neighborhood.

```
10 FOR X = 54272 TO 54295: POKE X,0: NEXT
20 POKE 54296,15
30 POKE 54278,240
40 POKE 54277,96
50 POKE 54276,33
60 FOR X = 1 TO 20: POKE 54273,X: NEXT
70 FOR X = 1 TO 550: NEXT
80 POKE 54296,0
```

For a moose in heat add:

```
90 GOTO 30
```

NO, WE SAID “LIST”!

When you RUN this program on the 64, it will LIST instead. To make matters even more confusing, when it's done listing, it will run again, and list itself . . . then run again, and list itself again . . . then run again . . . and again . . . and again. . . .

```
10 PRINT "{CLEAR HOME} {CRSR DN} {CRSR DN}"
20 PRINT "LIST"
30 PRINT "{? CRSR DN'S} RUN"
40 PRINT "{HOME}"
50 POKE 631,13: POKE632,13: POKE 198,2
```

Why? It's a technique called a *dynamic keyboard*. The program prints the word LIST at a very specific place on the screen, and the word RUN at another specific location. Then in line 40, the cursor is sent to the home location, and some values in line 50 are poked into memory. At this point the program is almost over, and the READY prompt appears on the screen—one line, not coincidentally, above the word RUN.

The trick is in line 50. Two RETURN keypresses—CHR\$(13)s—are POKEd into the memory location of the keyboard buffer. The buffer will “remember” up to ten keypresses, in locations 631 to 640: location 198 is the counter of the number of keypresses stored there. In our example, there are two.

When the program is over, it acts as though somebody is actually pressing the RETURN key—once when the cursor is on the command LIST, and again when it is on the command RUN.

With some Beagle-clever programming, you can use a dynamic keyboard to get the program to executive all kinds of commands in the direct mode, LOAD, SAVE, etc., etc. Or, make self-modifying programs that print new lines of code and then save and replace themselves on the disk. Or to change variables into hard DATA statements that become a per-

manent part of the program. The key to the trick is printing things with correct syntax, placing the statements in the right location, and getting the cursor to end up on top of the commands.

DYNAMIC KEYBOARD SAVE

For the lazy among us, use this instead of an END statement. It's for people who like to do a little programming, RUN the program to test it, then SAVE it automatically, every time, no matter what.

```

9000 PRINT "{CLEAR HOME} {CRSR DN} {CRSR DN}"
9020 PRINT "SAVE" CHR$(34) "@:PROGRAM NAME"
      CHR$(34) ".8"
9030 PRINT "{HOME}"
9040 POKE 631,13:POKE 198,1

```

DEALING WITH COMMODORE MATHEMATICS

Rounding errors can be the end of a program that does mathematical manipulation of data. An accounting program, for instance, is not worth much if it makes up numbers along the way, and Commodore's math sometimes has a vivid imagination. To see what we've discovered in our "whoops" department, try either of these statements in the direct mode:

```
X = 98 - 100.1: PRINT X
```

or

```
FOR X = 1 TO 1000 STEP 1.1: PRINT X: NEXT
```

The extra digits are bad enough, but if you begin to factor rounding error numbers with multiplication and division, the figures get worse. So count on sending all your numbers back through the number fat farm to

get them back into shape. This little routine will fix anything. Use it before you print, and Commodore won't keep you from winning the Field's medal in mathematics.

```
X = INT(X*100+.5)/100: PRINT X
```

DATA INPUT

INPUT statements place a big question mark on the screen. Some grammarians think the question mark is redundant—like someone saying the word “huh?” at the end of every question asked. Do you have a C-64, huh? So do you want to get rid of the question mark, huh? Don't like it either, huh? Can you manage a simple POKE statement, huh?

POKE 19,64	Disables question mark on INPUT
POKE 19,0	Enables question mark

CUSTOM INPUT PROMPTS

Once the question mark is eliminated from the INPUT sequence, you are free to customize the prompt. This will put a flashing yellow star on the screen.

```
10 POKE 19,64
500 INPUT "{CTRL-B} * {COM LOGO-7} {CRSR
LEFT}" ; A*
```

GET'S CURSOR

GET statements are often used to bring data into a program, in place of INPUT statements. The disadvantage is that GET provides no onscreen cursor, unless you use this subroutine in 64 mode, below:

```
2000 POKE 20410: GET X$: IF X$ = " " THEN 2000
2010 POKE 20411: RETURN
```

GET THE BEST INPUT

Our favorite trick for getting input doesn't use the INPUT statement, and makes its own cursor instead of tickling the system with funny POKEs. It works with the GET statement, screening characters one at a time, and if they pass the "OK Character Test", we add 'em to our final string.

It's a long subroutine, but in a big program that relies on user input (this is important stuff,) it's worth the space.

```
9010 GET A$: IF A$ = " " THEN GOSUB 9100: GOTO
    9010
9020 A = ASC(A$)
9030 IF A < 65 OR A > 90 THEN 9010
9040 IF A = 13 THEN RETURN
9050 PRINT CHR$(154); A$
9060 Z$ = Z$+A$
9070 GOTO 9010
9100 PRINT"{CTRL-2} {SHIFT-PLUS SIGN} {CRSR
    LEFT}"
9110 PRINT"{CTRL-8} {*} {CRSR LEFT}"
9120 PRINT"{CTRL-6} {SHIFT-V} {CRSR LEFT}"
9130 PRINT"{CTRL-5} {SHIFT-Q} {CRSR
    LEFT}":RETURN
```

Line 9010 looks for a keypress. If there is none, it goes to the subroutine that makes the cursor flash, then looks for another keypress.

Line 9020 converts the key pressed to a number value.

Line 9030 screens for alpha input only. For numbers, make it <48 or >57.

Line 9040 looks for a RETURN keypress, signaling the end of the input.

Line 9050 sets the screen printing color back to light blue, and prints the last key pressed.

Line 9060 makes Z\$ equal to the sum of all keys pressed, and in their proper sequence. Z\$ is carried into the rest of the program.

Lots of things can be added to this character GET subroutine, including checks for line length—LEN(Z\$)—buzzers, bells, and if needed, moose calls.

CHAPTER 11

MACHINE LANGUAGE

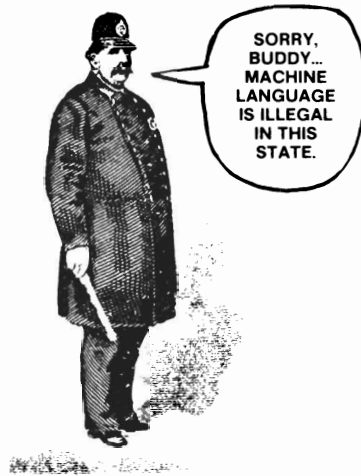
ABOUT MACHINE LANGUAGE

Machine language (ML) is the only programming language your 64 or 128 understands. All of the BASIC commands you enter into the computer are interpreted by a humongous ML program called the *operating system*, and that's how the work gets done. Now if you bypass the process of sending every piece of BASIC code through the hasher to have it interpreted, things get done a lot faster. Sometimes twice as fast, sometimes hundreds of times faster, sometimes thousands of times faster. Savvy?

ML stand-alone programs usually have to be LOAded with the ,8,1 suffix. From within a BASIC program, ML subroutines are called with the SYS #### and USR #### commands, which tell the computer to JUMP to a memory address that's specified in the command. Then, the ML routine inside must end with the RTS instruction, to return to BASIC.

The other way to get a ML program into memory (a small ML program, that is) is by REAding DATA statements, and POKEing them into memory. The program below uses this last method to do absolutely nothing of value. Bro. Ben, however, who likes to look at test patterns on the TV set, thinks you should try this:

```
10 FOR A = 728 TO 742: READ B: POKE A,B: NEXT:  
   SYS 728  
20 DATA 160,0,140,32,208,140,33,208,200,32,  
   228,255,240,244,96
```



SPEED CHECK

This program is the BASIC version of the ML algorithm above. Try it just to see how much faster ML things run. We suggest you put on sunglasses first.

```
10 FOR X = 0 TO 15: POKE 53280,X: POKE 53281,X:  
   NEXT: GOTO 10
```

DEC TO HEX CONVERTER

Machine language usually means hexadecimal (base 16) notation. Our studies show this is because both the Commodore 64 and 128 have eight fingers on each hand. Here's a short program to convert our decimal (ten-finger) numbers to hex:

```

10 A$ = "0123456789ABCDEF"
20 INPUT N
30 D = 4096: FOR X = 1 TO 4: A = INT(N/D)
40 PRINT MID$(A$,A+1,1);
50 N = N-A*D: D = D/16
60 NEXT
70 PRINT: GOTO 20

```

HEX TO DEC

What goes around, comes around. This converts a hex notation back to numbers most used car salesmen can understand.

```

10 INPUT A$
20 FOR X = 1 TO LEN(A$)
30 B = ASC(MID$(A$,X,1))-48
40 IF B > 9 THEN B = B-7
50 D = D*16+B: NEXT
60 PRINT D
70 D = 0: GOTO 10

```

BINARY TO DECIMAL CONVERSIONS

The following program will convert a binary string of ones and zeros to its equivalent decimal notation. Be careful when you use it, because like the conversion routines above, there is no error checking.

```

10 INPUT A$
20 FOR X = 1 TO LEN(A$)
30 D = D+1-(MID$(A$,X)>="1"): NEXT
40 PRINT D
50 D = 0: GOTO 10

```

DECIMAL TO BINARY

This is your ticket to binary notation, provided you don't need to convert a decimal figure over 255—but then, who would?

```

10 INPUT N
20 FOR X = 0 TO 7
30 A$ = CHR$(N AND 1 OR 48)+A$
40 N = N/2
50 PRINT A$
60 PRINT: A$ = " ": GOTO 10

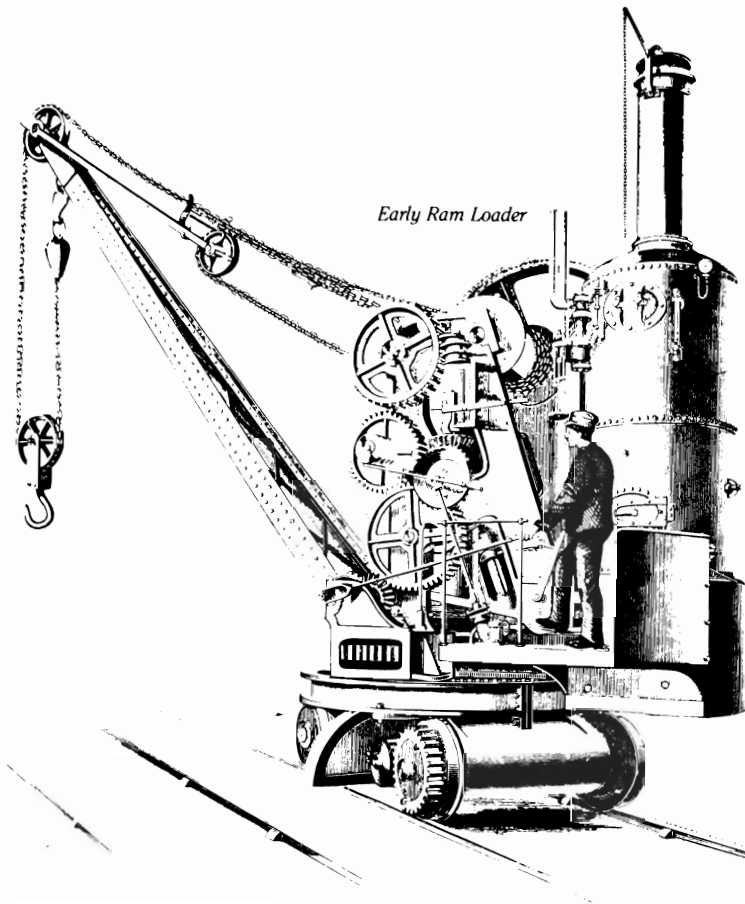
```

HEXADECIMAL BANKING ON THE 128

The 128 displays a five-digit hexadecimal address when you're in the monitor, instead of the four-digit hex address we're all accustomed to seeing. The extra digit on the *left* (the high-order digit) specifies the bank configuration at the time the command is to be carried out. Here's a 16-character list of what's at the bank's address:

- 0 RAM 0 only
- 1 RAM 1 only
- 2 RAM 2 only
- 3 RAM 3 only
- 4 INT ROM, RAM 0, I/O
- 5 INT ROM, RAM 1, I/O
- 6 INT ROM, RAM 2, I/O
- 7 INT ROM, RAM 3, I/O

- 8 EXT ROM, RAM 0, I/O
- 9 EXT ROM, RAM 1, I/O
- A EXT ROM, RAM 2, I/O
- B EXT ROM, RAM 3, I/O
- C KERNEL and INT RAM 0, I/O
- D KERNEL and EXT RAM 1, I/O
- E KERNEL and BASIC, RAM 0, CHARROM
- F KERNEL and BASIC, RAM 0, I/O



INTO THE MACHINE LANGUAGE MONITOR

The quick way in the door to the Machine Language Monitor on the 128 is to use the F8 key. On the poor-little-old-me 64, you'll need to write (or, most likely, buy) a machine language assembler program. An assembler—sometimes called an “ajumbler” around this Beagle Bros. Branch Office—allows you to write machine language instructions using a standard mnemonic format. Using an assembler and writing mnemonic commands is a lot easier than writing strings of numbers.

Let's review . . . on the 128 press the F8 key. On the 64, go out and buy an assembler. In either case, here's an alphabetized list of the microprocessor's instruction set:

ADC	Add memory to accumulator with carry
AND	Logical AND memory with accumulator
ASL	Shift left one bit (for memory or the accumulator)
BCC	Branch on carry clear
BCS	Branch on carry set
BEQ	Branch on result of zero
BIT	Test bits in memory with accumulator
BMI	Branch if minus flag is set
BNE	Branch if result is not zero
BPL	Branch if the minus flag is clear
BRK	Software interrupt, forced break
BVC	Branch on overflow clear
BVS	Branch if overflow is set
CLC	Clear carry flag
CLD	Clear decimal flag
CLI	Clear interrupt flag to enable interrupts
CLV	Clear the overflow flag
CMP	Compare the accumulator with memory
CPX	Compare the X register with memory
CPY	Compare the Y register with memory
DEC	Decrement memory location by one
DEX	Decrement the X register by one
DEY	Decrement the Y register by one
EOR	Exclusive OR between the accumulator and memory

INC	Increment memory location by one
INX	Increment the X register by one
INY	Increment the Y register by one
JMP	Jump to an address (like BASIC's GOTO)
JSR	Jump to a subroutine (like BASIC's GOSUB)
LDA	Load a value into the accumulator
LDX	Load a value into the X register
LDY	Load a value into the Y register
LSR	Logical shift right
NOP	No operation
ORA	Inclusive OR between the accumulator and memory
PHA	Push accumulator onto the stack
PHP	Push processor status byte onto the stack
PLA	Pull accumulator from off the stack
PLP	Pull processor status byte from off the stack
ROL	Rotate left
ROR	Rotate right
RTI	Return from interrupt
RTS	Return from subroutine
SBC	Subtract with borrow
SEC	Set carry flag
SED	Set decimal flag
SEI	Set interrupt flag
STA	Store accumulator in memory
STX	Store X register in memory
STY	Store Y register in memory
TAX	Transfer accumulator to X register
TAY	Transfer accumulator to Y register
TSX	Transfer stack pointer to X register
TXA	Transfer X register to accumulator
TXS	Transfer X register to stack pointer
TYA	Transfer Y register to accumulator

UNSUNG OPCODES

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

BBB	Buy Beagle Bros.
CYM	Call Yo-Mamma
GAS	Go Ask Somebody
GBO	Get Bugs Out
LGN	Let's Get Naked
NNA	No Nerds Allowed
RTM	Read The Manual
TAN	Take A Nap

128 MONITOR COMMANDS

Machine language programs written with the 128's built-in machine language monitor can run as stand-alone programs, or called as ultra-fast-and-tricky subroutines from a BASIC program. (You've got to be careful, however, to position the program in memory so that it doesn't clobber some important operating system functions or expose itself to being overwritten by BASIC.)

To get directly into the monitor, press F8. From a BASIC program, use the command MONITOR. Once you're in, here's the list of KEYWORDS to use:

Keyword	Purpose
A ASSEMBLE	Enter a line of assembly code
C COMPARE	Compare two areas of memory
D DISASSEMBLE	Disassemble 8502 code
F FILL	Fill a range of memory with a given byte
G GO	Restore all registers and begin execution of a program as the specified address

Keyword	Purpose
H HUNT	Hunt through memory for all locations with a given byte
L LOAD	Load a file from disk
M MEMORY	Give the hex values in a memory dump
R REGISTER	For the status register, program counter, accumulator, X and Y registers, and the stack pointer
S SAVE	Save to disk
T TRANSFER	Transfer code from one memory location to another
V VERIFY	Verify or check a file in memory against one on disk
X EXIT	Exit back to BASIC 7.0
@	Check the disk drive's status
\$	Precede hex numbers
+	Precede base 10 decimal figures
&	Precede octal representation
%	Precede binary values

WHERE TO STICK IT TO THE 64

The hot place to stick your machine language routines on the 64 is from \$C000 to \$CFFF, as long as the code is less than 4K.

If 4K is not enough, reserve a seat at the top of memory from BASIC; normally \$9FFF.

If that's still not good enough, change the MEMTOP with POKE in the direct mode through BASIC:

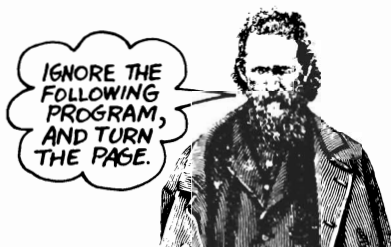
```
10 L = LOW VALUE: H = HIGH VALUE
20 POKE 51,L: POKE 55,L
30 POKE 52,H: POKE 56,H: CLR
```

Make sure the values POKEd from BASIC are in decimal, not hex. So in the example above, if you were reserving \$9000-\$9FFF, let L=0 and H=144.

BIT SPLITTER

Psssst. Want to know how to smash a decimal number into teeny bits?

```
10 INPUT N
20 FOR BIT = 1 TO 8: PRINTTAB (9-BIT)
30 PRINT N-INT(N/2)*2;
40 N = INT(N/2): NEXT
50 PRINT: GOTO 10
```



VERIFY ML

The VERIFY command works as well with machine language programs as with BASIC files. To make the command work, use the suffix of ,8,1. Just because a machine language program is being verified, we're sorry to say, doesn't make the process faster. VERIFY is one slow-poke command; BASIC, ML, or otherwise.

```
VERIFY "PROGRAM NAME",8,1
REM TAKE NAP
```

CHAPTER 12

DISKS AND DRIVES

WHAT'S DOS?

Commodore DOS means the disk operating system, a machine language program that tells the computer how to find and store files when you give it the LOAD, SAVE, and other disk I/O commands. DOS handles the communication between the computer and the drive. It prepares generic diskettes for use on the Commodore system. It puts an addressing system on the disk so it can locate existing programs or identify empty space to store incoming data. And more.

On a lot of computers, like IBM and Apple, DOS must be first loaded into the computer from a DOS disk, or “installed”—manually added onto a program disk. Otherwise the computer will not “know” how to LOAD or SAVE or even use the disk drive. In these machines DOS will also eat up some of the available memory.

Commodore drives have DOS built right into them, and do not use the computer's resources. Furthermore, the 1541 disk drive has its own 6502 microprocessor, two 6522 I/O chips, 2K of RAM, and a DOS program permanently stored in 15K of ROM. Pretty good, huh?

WHY DOES MY DISK DRIVE MAKE ALL THAT NOISE?

The most frequently asked Commodore disk drive question is why the drive sometimes bangs, shakes, and makes “broken equipment” sounds. It’s because this “smart peripheral” with all its microchips has also been given an autoinitialization feature. When the drive encounters an error it cannot seem to correct, the drive begins the process of recentering the read/write head over the middle of the disk (circular track 18). To be sure it comes back to the middle it first goes far to the outside and the stepper motor pulley bangs it against the mechanical metal stop (lots of times). Then the head comes back a measured distance to a position precisely over reserved track number 18, where the DOS listing of What’s-Where is stored on each disk.

DISK DRIVE SILENCER

Some programs have heavy-duty copy protection that seems to beat your drive to death. To minimize the bumps, grinds, chatters, and cries for mercy, run this BASIC program. When the READY prompt appears, load whatever disk according to whatever instructions. Most of the forced repositioning of the drive will be gone.

```
10 OPEN 15,8,15
20 PRINT #15, "M-W"; CHR$(106) CHR$(0) CHR$(1)
   CHR$(133)
30 CLOSE 15
```

This memory write (M-W) command POKes new values into the disk RAM that may, depending on the copy protection, keep your disk from loading at all.



WHAT'S IN THE BLACK ENVELOPE?

Don't get the scissors, trust us. Inside the diskette jacket is a well-organized data storage system. A nice man we call Mr. DOS already knows this system, so he knows where to look for a particular program. We asked Mr. DOS's boss, BOSS DOS (who wrote Beagle Bros.' *DOS BOSS*), to tell us about life inside the black envelope. Here's what was said:

"It's dark inside the disk, dark as the far reaches of outer space. Since we can't see, all processes are done from memory, including the storage of data on the disk in a series of concentric circular paths called *tracks*. These are like the rings around Saturn. The tracks are referred to by number, starting with track 1 at the outside edge of the disk, and track 35 in close to the inside edge.

"Each track is further divided into *sectors*: areas for storage of your programs and files. Each sector can store up to 256 characters. Tracks around the outside of the disk are, of course, larger than the little ones in the center, and have more sectors. Expect to see 21 sectors on the outside rings and 17 sectors inside on track 35. Sectors, by the way, are also called *blocks*, and correspond to the numbers seen on the left of the screen when you list a directory.

"The middle ring is track 18, and Mr. DOS reserves all of track 18

for making lists and kicking bytes around. Track 18 has all the juicy gossip about what is going on around the disk. Some of the best information is stored in the Block Availability Map, or BAM; the register of all the other sectors is kept here. Sector 0 also stores the 16-letter name the disk was formatted with, and the important two-character ID number. If the name was less than 16 letters, or the ID less than two, Mr. DOS padded the end with shifted-space characters.

“The first time any disk input or output occurs, including file reads or writes, a SAVE, or a LOAD, Mr. DOS looks at the BAM and places it into the disk drive’s memory (not the computer’s), including the disk name and ID number. (Clever, right?) The RAM on the disk drive holds knowledge about what’s on your disk. This is why Commodore disk drives are called *smart peripherals*.

“Now, Mr. DOS knows if you switch disks, because before the next read or write operation is carried out, he reads the disk ID number. If it is the same ID, Mr. DOS assumes it is operating on the same disk and uses the BAM stored in memory to determine which tracks and sectors are available to store incoming data. If the ID is different, the new BAM is read first, so that the drive can get a correct Block Availability Map.

“But what happens if you swap disks and the ID number of the two are the same. Wooooaahhhhh! Big trouble in this black hole! Mr. DOS will think he is dealing with the same disk and use the BAM in memory to direct the data for storage. Mr. DOS will likely overwrite some or all of your files. Gone, destroyed forever, never to be recovered.”

Morale of the story from the DOS BOSS:

A tip you should take from me,
Is format disks with unique IDs.
Or the read/write head
makes your data go dead,
And the drive’s done a number on thee.

HOW TO FORMAT A DISK

Sixty-four mode. Place the disk into the drive, with the little square notch on the left side and the label facing upward, then close the drive latch. It's trapped, which is good. Enter the following:

```
OPEN 15,8,5,"NO:DISKNAME,ID"
```

The little red light will go on and the drive will whirl for a about one minute and 20 seconds.

In the command above:

- 15,8,15 opens a channel to the drive.
- N stands for NEW, but not the kind that erases files in memory.
- 0 is the disk drive device number.
- DISKNAME is where you enter up to 16 characters for your own disk name.
- ID is where you add two unique disk ID letters or numbers.

On the 128, use the HEADER command. Enclose the (up to) 16-character disk name within quotation marks, type a comma, the letter I, and then any unique two-character disk ID.

```
HEADER "MYDISK",I87
```

WARNING—WARNING RED ALERT: The Commodore 128 will format *both* sides of the disk, even if you are using a “floppy” and the second side has the write-protect notch covered.

FLIPPY DISKS

Both sides of a disk are coated with the same magnetic materials, so in most cases, both sides will work just fine. The disk manufacturers who sell you these “single-sided” disks warn against using both sides because

1) they lose money when you buy only half as many disks, and 2) the side you are about to use may not be tested for quality.

To use both sides of a single-sided disk and commit this semidangerous deed, use scissors or a hole punch to cut a write-protect notch on the opposite side of the disk, and flip it over, label-side down. We doubt you will ever have any problems because we never have.

DIRECTORY ASSISTANCE

Commodore 128 owners need only type the command CATALOG to see a disk directory. On the 64, a catalog listing of what's on the disk can be seen by typing:

```
LOAD "¢",¢ {RETURN} {LIST}
```

If you want to print this directory, turn on your printer and then enter:

```
OPEN 7,4: CMD 7: LIST
```

Then close the printer channel with the command CLOSE 7.

WHEN THINGS RUN AMOK...

The little red light on the drive starts flashing when something has gone wrong with a disk I/O operation. Use this little red light program to find out what it is.

```
9000 OPEN 15,8,15
9001 INPUT #15, A$, B$, C$, D$
9002 PRINT A$, B$, C$, D$
9003 CLOSE 15
```

This has to be done from within a program, as the INPUT# statement will not work in the direct mode. Be sure to use big line numbers: ones that do not overwrite and replace the good ones in your program. Delete these lines when you are done.

If you think this may happen again, leave these lines in the program, preceded by an END statement. Now any time the disk drive runs amok, just type:

```
GOTO 9000
```

and the error message and type will be displayed.



Copyright © 1983, Beagle Bros Inc.

AFTER THE CRASH

After an error has hung the drive it is sometimes difficult to make it work again. What the smart-#@*!!? drive needs is to be reset—so the computer can see a status byte that says “ready-to-go.” One way to do this is just turn the drive off—then on again. Another sometimes successful 64 trick is:

```
POKE 144,0
```

DIRECTORIES THAT COST YOU NOTHING

Loading a directory with the Commodore 64's LOAD"\$",8 command will cause the expensive loss of a program in memory. This tip will help you avoid the cost of losing your program.

```
POKE 44,PEEK(46)+1
```

Then load and list the directory. When it's time to go back to programming use the POKE below. Your program listing will reappear.

```
POKE 46,PEEK(44)-1: POKE44,8
```

BY THE WAY

In case you didn't know it, the 1541 disk drive head reads the bottom surface of the disk, not the top. The 1571 drive reads both sides of the disk.

LOAD THIS

Using the command `LOAD "*",8`, will load the last file accessed, thanks to our smart peripheral disk drive. That is, of course, unless you have put a new disk in the drive—the same command will then load the first file on the directory. To use the wildcard "*" to get the first file every time, insert a colon in front of the asterisk.

```
LOAD ":*",8
```

AND JUST TRY TO LOAD THESE

There are lots of way to fix your file name and make a program on the disk difficult (impossible) to load. Some people call this "protection".

Disk Directory Protection Trick 1

Add a shifted space as the first character of the file you wish to SAVE. (This really confuses the directory display as shifted spaces are normally added by DOS to pad short names in the system.) When the program is displayed in the directory, both quotation marks will be at the front of the name instead of surrounding the name. To LOAD, use the normal convention, just be real sure to add the shifted space.

Disk Directory Protection Trick 2

Use two or more shifted spaces. Lots more if you like. You can even use more than the allocated 16 characters. Just remember what the name is, because you must get it right to reload the program.

Disk Directory Protection Trick 3

Begin the SAVE command in the normal way, using a simple name, then press SHIFT-RETURN. This will move the cursor down to the next line without processing the input. Next, move the cursor back up to the command line and press INSERT to add a space between the leading quotation mark and the file name. Press the DELETE key. A reverse up-arrow will be displayed there. Last, press the RETURN key to process the command. When the catalog is displayed, no leading quotation mark will appear. Repeat the process above to LOAD the program back into memory.

Disk Directory Protection Trick 4

Use trick #3 above, and insert lots of spaces instead of just one. Up to 31 DELETE characters can be used with a one-character file name. When the directory is displayed, this phony file name will erase part of the screen.

This one is great. With lots of DELETE characters it will back up the cursor enough to even erase the display of files *above* this trick name.

Disk Directory Protection Trick 5

Use the normal convention for SAVE, adding a comma and the letter S or U to the end of the name.

```
SAVE "FILENAME,S"↵
```

or

```
SAVE "FILENAME,U"↵
```

The directory will display the file as a SEQ (sequential file) or aUSR file, rather than showing the program name followed by PRG. This will confuse most anyone, as well as other programs that need to load these data files.

Disk Directory Protection Trick 6

Use the SAVE command with a normal name:

```
SAVE "BEAGLEFILE",8
```

Then press SHIFT-RETURN to move the cursor down to the next line. Next move the cursor back up and insert two spaces between the first quotation mark and the file name. Press SHIFT-SPACEBAR and then CONTROL-7. (CONTROL-7 is the shift into printing blue characters—which is by design the default background color on the 64 screen; use green for the 128.) When the directory is displayed on the blue 64 screen, ha-ha, it's not there. Neither is the cursor. Too bad.

DON'T USE THIS

The 1541 disk drive has a SAVE AND REPLACE command with a convention of SAVE"@0:FILENAME". Use it to replace an updated version of an old program without changing its name every time. It is just too good to be true . . . this Commodore DOS function has a bug in it. Every once in a while the SAVE@ command erases the wrong file and replaces it.

It is especially dangerous when used with a nearly full disk. This is because the command works by first saving the new version of the program, without erasing the first or checking to see if enough space is available on the disk.

What's better? Erase (SCRATCH) the old version on the disk, then save the new one.

```
OPEN 15,8,15,"S0:FILENAME": CLOSE 15
SAVE"FILEMANE",8
```

DISK TIPS

It is alright to leave a disk in the drive and turn the computer on and off. It is *not* alright to turn off the power to the disk drive with a disk in it, because you (yes, YOU) may lose some of the data stored on the disk.

DID YOU KNOW....

When the computer encounters a command, it first checks to see if it is a DOS command, then checks to see if it is a BASIC keyword. DOS is first because the Dos Boss says DOS is best.

But if the system cannot match the command with its list of acceptable keywords, you see the famous...

```
? S Y N T A X   E R R O R  
R E A D Y .
```

NINE FREQUENTLY USED DISK COMMANDS

FORMAT

Before you use a new floppy disk, you must **FORMAT** (also called **HEADER** or **NEW**) the disk. This prepares it for use in the Commodore system.

DIRECTORY

This is a **CATALOG** or listing of all the programs and files stored on the disk. It's like a table of contents.

VALIDATE

When you've SAVED and SCRATCHed a whole bunch of files on the disk, things get disorganized. (Un-disk-organized.) This command tells DOS to reconstruct its Block Availability Map, which shows where information is stored, and get rid of references to old programs or partial files.

Rename

For the occasional need to change a file or program name.

Copy

To back up an existing file on the diskette, by placing an identical second copy of the program on the disk under a different name. The rules for copy are 1) the new file must have a different name, 2) it won't work on relative files, and 3) you gotta know that you have enough room on the disk before starting.

SCRATCH

This gets rid of an unwanted file. WARNING: don't use the SCRATCH command on an unclosed file. Unclosed files appear in the directory as having 0 blocks and an asterisk preceding the file type (*SEQ, *PRG, etc.). It could also trash data somewhere else on the disk. Eliminate unclosed files with the VALIDATE command instead.

Initialize

No, this has nothing to do with formatting, although Apple owners won't believe it. Initializing a diskette forces Mr. DOS to read the disk ID and the current BAM into the drive memory. Initializing the disk every time

you insert a new disk is a good habit to get into, especially if you think all your disks have the same (heaven forbid) ID number.

VERIFY

This is a BASIC command that can be used in either direct or program mode to compare the contents of a file stored on the disk to the one in memory. It is most often used right after the SAVE command. If something is different, you'll see:

```
? VERIFY ERROR
```

CONCAT

Concatenation is the combining of the contents of one file into another, making one big set of records. It's used mostly for the merging of two sequential data files (like word processing documents) that already exist on disk. It's a 128 command only.

FORMAT {NEW}

```
  64: OPEN 15,8,5,"NO:DISKNAME,##": CLOSE 15
 128: HEADER "DISKNAME",I ##
```

DIRECTORY

```
  64: LOAD " ",8 {RETURN} {LIST}
 128: CATALOG
 128: DIRECTORY
 128: F3
```

VALIDATE

```
  64: OPEN 15,8,15,"V": CLOSE 15
```

RENAME

```
  64: OPEN 15,8, 15,"R:NEWNAME=OLDNAME": CLOSE 15
 128: RENAME "OLDFILE" TO "NEWNAME", D1
```

COPY

```
  64: OPEN 15,8,15,"C:BACKUP=ORIGINAL": CLOSE 15
```

```
128: COPY "SOURCEFILE", DO TO  
      "DESTINATION.NAME", DL
```

SCRATCH

```
  64: OPEN 15,8,15,"SO:FILENAME": CLOSE 15  
128: SCRATCH "FILENAME", DL
```

INITIALIZE

```
  64: OPEN 15,8,15,"I": CLOSE 15
```

VERIFY

```
  64: VERIFY "HELLO",8  
128: DVERIFY "HELLO", DL
```

+CONCAT

```
128: CONCAT "FILE1", DO TO "FILE2", DL
```


CHAPTER 13

AUDIO AND VIDEO COMMUNICATIONS

SPEED: MODEMS VS. SECRETARIES

Baud rate is not an accurate translation of a modem's speed because the modem has to translate parity bits, start bits, and stop bits, as well as data bits. A 300-baud modem averages about 35 real characters per second and a 1,200-baud modem about about 140 characters per second.

A good secretary typing at 60 words per minute averages about five characters per second. Using this information, the following ratings should apply to telecommunications:

- 300-baud modem = seven SPMs (secretaries per minute)
- 1,200-baud data transfer = 28 SPM
- 9,600-baud printers = typing speed of 224 SPM
- 19,200-baud satellite transmission = 448 SPM

TELEPHONE AUTODIALER

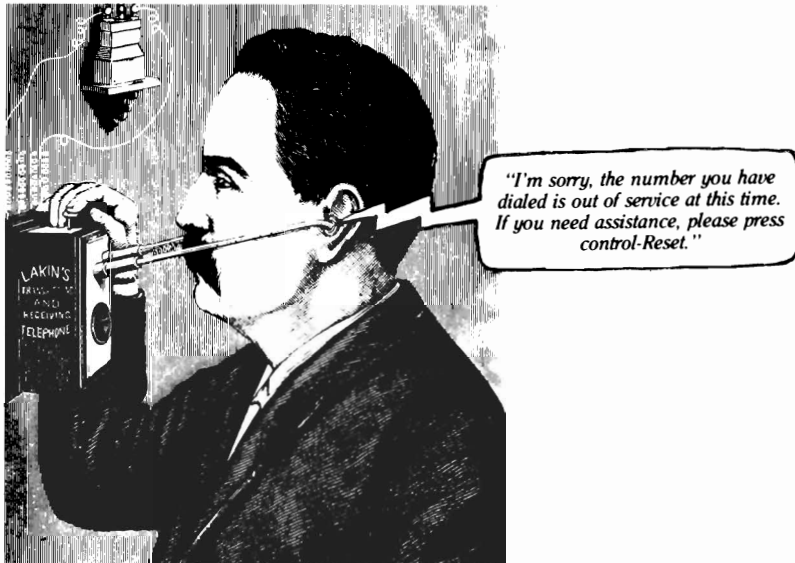
We found a way to turn a 64 into a secretary that will place all your calls. Just turn up the volume and hold the telephone receiver next to the speaker. This will make even a rotary style phone work on a touch-tone system, as many of the long distance carriers require.

Enter a number exactly as you would dial it, and don't add the area code unless it is necessary to reach the party from your exchange. Test (yes test,) the effectiveness of the dialer before putting it to use because the audio frequency of some exchanges, your telephone equipment, or the volume of the speaker may alter its operation. And, make darn sure you test the program with local calls. We're not responsible for calls to the North Pole, the Tijuana Jail, Aunt Millie's car phone, or any other wrong number.

```

10 INPUT "NUMBER PLEASE ":PH$
20 A = .975: B = .059
30 FOR N = 0 TO 3: READ F: F = F*A/B
40 D(N) = INT(F/256): E(N) = F-256*H(N): NEXT
50 DATA 697, 770, 852, 941
60 FOR N = 0 TO 2: READ F: F = F*A/B
70 H(N) = INT(F/256): E(N) = F-256*H(N): NEXT
80 DATA 1209, 1336, 1477
90 T1 = 54272: T2 = T1+7
100 POKE T1+5,0: POKE T1+6,240
110 POKE T2+5,0: POKE T2+6,240: POKE T1+24,15
120 FOR J = 1 TO LEN(PH$): N$ = MID$(PH$,J,1)
130 IF N$ = "*" THEN N$ = "10": GOTO 170
140 IF N$ = "0" THEN N$ = "11": GOTO 170
150 IF N$ = "#" THEN N$ = "12": GOTO 170
160 IF ASC(N$) < 48 OR ASC(N$) > 57 THEN 240
170 Z = VAL(N$)-1
180 R% = Z/3: C% = Z-3*R%
190 POKE T1+1,D(R%): POKE T1,E(R%)
200 POKE T2+1,H(C%): POKE T2,E(C%)
210 POKE T1+4,17: POKE T2+4,17
220 FOR N = 1 TO 400: NEXT
230 POKE T1+4,0: POKE T2+4,0
240 PRINT N$;: NEXT

```



MODEM U.

For a fee, a San Francisco company will connect your Commodore via modem to universities around the country, to enable you to get a college degree (including computer science, they say), from your bedroom. Use the autodialer and call (800) 225-3276. In California, autodial (800) 445-3276.

A CLASS IN TELEVISION GRAPHICS

When you get bored looking at lines of program code, the professor suggests you plug a VCR right into your Commodore 1702 monitor. Just connect the VCR's audio-out and video-out to the monitor, using the video-audio inputs on the front of the panel monitor. Be sure to check your local listing for the Beagle Bros. Indoor Sports Show.

RADIO WAVES

The rumor around campus is that you can play your stereo through your computer while you are programming—if you have one of those monitor cables with some unused RCA plug hanging from the back of the 1702 or other monitor. Use a patch cord to connect the Walkman, blaster, or whatever to the cable, then, in the direct mode, type:

```
POKE 54296,15
```

to crank up the volume.

MORE ON MONITOR CABLES

Some Commodores have a five-pin monitor cable socket, others have an eight-pin socket. If you somehow bought an eight-pin monitor connector cable and it won't go into your five-pin socket, then get the pliers and rip out pins 6, 7, and 8.

Don't worry, this won't turn out the lights on your monitor. Pins 7 and 8 (up top, near the notch) make no connection anyway. Pin 6 in the middle was the chroma signal output on an eight-pin DIN system, but isn't needed on the five-pin connection you need to make.

LIGHTS OUT!

We have a computer game that has a menu option called "Hide, the boss is coming!" When a certain key is pressed, the screen goes black. When the key is pressed again, back comes the game. On the 64 you can turn the monitor off to hide nasty things, or to disguise a slowly drawn display, with:

POKE 53265,43 Screen goes night-night
POKE 53265,27 Turns screen on again

Another way to do it is to POKE the screen border, background, and text color to black. Be sure to make the software PEEK at the values in memory's color locations so they can be set back from within the program.

POKE 53280,0 Border goes black
POKE 53281,0 Background color goes black
POKE 646,0 Text goes black

WAKE-UP CALLS FROM THE 128

The SLEEP command on the 128 can be used for a very unusual wake-up call from your computer.

1. Use the great graphics to draw a picture of breakfast.
2. Write a program to map the screen, then blank the picture, then:
3. Use the SLEEP command to turn the screen on at the desired time and wake you up with breakfast waiting. With SLEEP, you can make the computer hang around and do nothing for up to 18 hours and 20 minutes.

SLEEP 3600 Doze for one hour

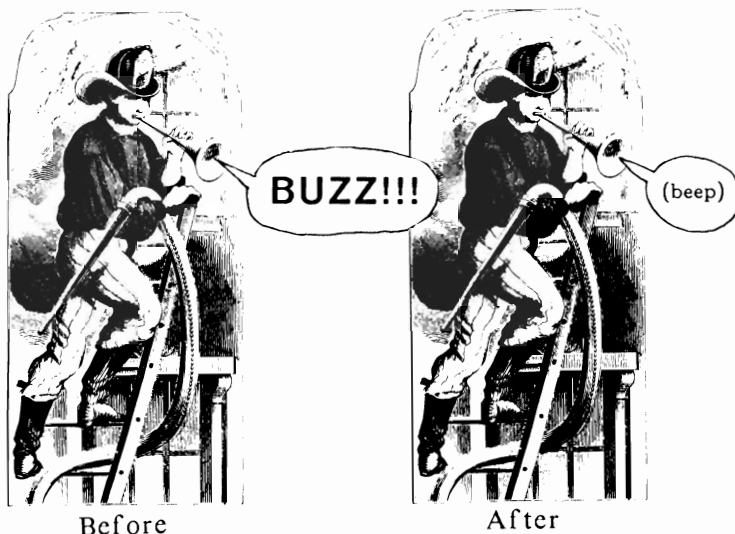
SLEEP 28800 Snore for eight hours

FOR X = 1 TO 41257: SLEEP 65535: NEXT Coma for the
next 85.7 years

AN AWFUL BUZZER

Some people are really into heavy-weight bad noise, alarm clocks, and otherwise. This 64 program has got to be the worst buzzer in the known world.

```
10 F = 54272
20 FOR X = 0 TO 24 : POKE F+X,0 : NEXT X
30 POKE F+5,15
40 POKE F+15,9
50 POKE F+24,15
60 POKE F+1,39
70 POKE F+4,21
80 FOR T = 1 TO 2000 : NEXT T
```



BUZZING TVs

Using a TV set for a monitor? Does it have an annoying low buzz from the speaker, even with the sound turned off? (They all do.) At Beagle Bros. Headquarters we fix this problem by plugging something into the earphone jack on the set. Real headphones are best—clothes hangers or paper clips are not recommended.

KIDS! DON'T TRY THIS AT HOME

```
POKE808,234: POKE 646,PEEK (53281)
```

It will make you go blind.

COMMUNICATIONS-CAY

This is the best terminal emulator transient utility one liner with syntax designed to address expressions on a local area network.?!??...

```
10 INPUT A$: A$ = RIGHT$(A$,LEN(A$)-
  1)+LEFT$(A$,1)+"AY"
PRINT A$: GOTO 10
```

It makes pig Latin out of everything.

AN ORCHESTRA IN AN ENVELOPE

Different musical instruments (because of different construction, of course) make different sounds. The 128 has a built-in command to reproduce some instrument sounds, in the ENVELOPE command. ENVELOPE sets the attack, decay, sustain, and release of a musical note. Here's a listing of the way to get to the ten-piece band waiting inside a Commodore 128:

Command	Instrument
ENVELOPE 0	piano
ENVELOPE 1	accordion
ENVELOPE 2	callopie
ENVELOPE 3	drum

Command	Instrument
ENVELOPE 4	flute
ENVELOPE 5	guitar
ENVELOPE 6	harpsichord
ENVELOPE 7	organ
ENVELOPE 8	trumpet
ENVELOPE 9	xylophone

CHAPTER 14

PRINTERS AND PRINTING TIPS

HOW THE PRINT#(ER) WORKS

Did you know that both the 128 and the 64 look at every peripheral device as a file? Yep. It's true for the printer, the disk drive, the monitor, and even the keyboard. To be compatible with Commodore's logic, all information coming in or sent out must be organized in one standard format, called a *file*. Commodore BASIC has standard commands that make it easy to exchange data with any file—and therefore all the peripherals can be talked to in the same way. They all respond to the same BASIC file commands.

Thus, if you understand one file format you understand them all, and this file format makes it easy to control output to the printer. All you need to do is assign a logical file number to the printer, then just write to that number any time you want.

The BASIC printer file commands on the 128 and 64 are OPEN, PRINT#, CMD, and CLOSE. Simple. Here's what each does:

OPEN

This command is used for three things. First it assigns a file number for the print output. Second it confirms the device number of the printer (it's usually 4). Third it prepares the printer to receive data. In a normal statement like `OPEN 7,4` we would make a file numbered 7, directed to device number 4, and the printer would be told to get ready.

PRINT#

This writes the data to the file. It works like the `PRINT` statement, only it directs the output to the indicated file number instead of the screen. (That's why the number sign is attached to the end of the command, in case you haven't figured it out.) To match our `OPEN` statement above, the printer command would be `PRINT#7,"YOUR MESSAGE HERE"`.

CMD

This command changes all `PRINT` output from the screen to the designated file, but the file must be `OPEN` before the `CMD` statement is used. With the example above, we would use `CMD7`. Once `CMD` has been used everything with a normal go-to-the-screen `PRINT` statement will go to the printer.

CLOSE

This command is used when you are finished with the file. It's good form to use it. It is like turning out the lights when you leave the room. Commodore will let you have ten files open at one time. Any more and you'll blow the circuit. The correct syntax for our example is `CLOSE7`.

PRINTER COMMAND SHORTHAND

If you are lazy and don't like to type, use these abbreviations for the printer commands. They may look funny when you first enter them, but when you relist your program the whole command word will appear.

Command	Abbreviation
OPEN	O SHIFT-P
PRINT#	P SHIFT-R (No ?# won't work!)
CMD	C SHIFT-M
CLOSE	CL SHIFT-O

More abbreviations are listed in Appendix B. Don't program with these shortcuts if you're a professional being paid by the hour.



128s ONLY

You can use CP/M's ECHO command to make a printed hard copy of all or part of a program file. First use the TYPE command to start the screen display of the listing. Press CTRL-S to stop the display . . . press CTRL-P to enable printer echo . . . then CTRL-Q to resume the display and start printing. Use another CTRL-S, CTRL-P, CTRL-Q sequence to stop the function when you're done.

SCREEN DUMPINGS

This little dumper is a super way to move the screen display from the monitor (device 3) to the printer (device 4). If you use it at the end of a program you'll be able to GOTO it any time to help debug. The variable 1000 is the number of characters that fit on a 25-line by 40-column screen ($25 \times 40 = 1,000$). Be sure to change that number if you're working in 80-column mode, or only the top one-half of the screen will be printed.

```
9999 END
10000 OPEN 3,3: OPEN 4,4: PRINT"{HOME}"
10010 FOR X = 1 TO 1000: GET #3,A$: PRINT#4,A$;
10020 NEXT: CLOSE 3: CLOSE 4
```

PROGRAM DUMPINGS

Another valuable statement at the end of a program is a LIST-ME routine. Just GOTO this when you want some hard copy:

```
20000 OPEN 7,4: CMD7: LIST
20010 PRINT#7: CLOSE 7: END
```

128 ECHO AUTO-DUMP

Use CP/M's CTRL-P to make screen output "echo" to the printer. While printer echo is in effect all the characters that appear on the screen will also list on the printer. To stop echo, press CTRL-P again.

CLR COMMAND VS. CMD STATEMENT

If the CLR command is used in your program, be sure it's used *before* you open a channel to the printer and begin printing. Otherwise CLR will automatically erase all files from memory—including the printer file. If you've CLRed when you try to print, you'll see:

```
DEVICE NOT PRESENT ERROR
```

DOUBLE SPACED LISTINGS

The file number you assign the printer can be any value from 1 to 255. If the number is 128 or greater the computer will make the printer double space everything.

```
OPEN 128,4: REM DOUBLE SPACE
```

LISTING A DISK DIRECTORY

It's hard to see the programs inside the disk, so someone invented the disk directory. If it's still hard to remember what's there once you've seen it, print a listing of the directory and tape it to the disk envelope.

```
LOAD "A",8  
OPEN 7,4: CMD 7: LIST: END  
PRINT#7: CLOSE7
```

Use ECHO in the 128 CP/M mode (press CTRL-P) with the DIR command to make a listing of the files on the disk.

PAPER SAVING TIP #1

You can save paper by reusing your old printouts and listings. Turn the paper over and use the reverse side.

PAPER SAVING TIP #2

You can save even more paper by making a giant Mobius strip. Get a long loop of paper, hold one end fixed, rotate the other 180 degrees, then tape the ends together. Be sure to get the right spacing on the tractor feed holes.

If you don't care much what your printouts look like, this will make your paper last forever. Feeding the Mobius strip back into the printer may, however, be difficult.

IT'S GOOD FORM TO PRINT HERE

With the Commodore and many other printers, sending them a CHR\$(16) (the print-here command) allows you to horizontally position the print head prior to printing. This is real helpful if you are going to write on preprinted forms (like—yuk—IRS tax forms). The format is PRINT#7,"nn". A two-digit number (nn) from 00 to 80 must be used. The syntax of either of the commands below will print orders for Mr. Scott at 45 spaces from the left:

```
PRINT#7,CHR$(16)"45"; "BEAM ME UP SCOTTY!"  
PRINT#7,CHR$(16)"45BEAM ME UP SCOTTY!"
```

BUYING A PRINTER

The most important thing to check when buying a printer is to make darn sure that it will work with your Commodore 128 or 64. Don't let a salesperson tell you about a printer's great qualities—make them show you—with the printer connected to a Commodore.

A non-Commodore printer must have the proper interface in order for it to connect and work correctly. So be prepared to take your Commodore into the store for a compatability test if you are buying a non-Commodore printer. Then try it out. And be sure to print out a listing of one of your programs that has some programming statements with cursor or color control codes entered in quote mode. That's the big test.

Most non-Commodore printers need a special interface because Commodore sends signals that can't otherwise be understood. Commodore output is sent through the serial port and all the data is sent and received over one wire. It's like a one-lane road. Most printers, however, are built to receive data through a parallel port, which uses eight separate wires—like an eight-lane highway. Think of the interface as the junction where the number of lanes is increased from one to eight, without a traffic jam. Also, there are lots of special Commodore graphic characters that are a foreign language to many printers, and these have to be created instead by the interface.

MORE PRINTER CHR\$ CODES

To see how CHR\$ codes can be used to make your printer go weird, try out a few of the following. They can even be used in combination with one another, so it's sometimes possible to have reversed, extra-wide, double strike, italicized, underlined characters, all at once. Your printer manual booklet may even list more codes than this. . . .

Code	Function
CHR\$(8)	Enter the land of bit image (graphics mode)
CHR\$(10)	Line feed
CHR\$(13)	Carriage return

Code	Function
CHR\$(14)	Extra wide, double width, fat-so characters
CHR\$(15)	Standard (skinny) width characters
CHR\$(16)	Start-printing-in-this-column command
CHR\$(17)	Enter uppercase/lowercase mode
CHR\$(18)	Print reversed characters
CHR\$(26)	Repeat a selected bit-mapped image
CHR\$(27)	Specify a dot address/escape
CHR\$(141)	Return without a line feed
CHR\$(145)	Enter uppercase/graphics mode
CHR\$(146)	Turn off reverse printing please

PRINTER RIBBON REINKING

The inking wick for your printer ribbon is built into the cartridge. The ink goes on as the ribbon goes through the little plastic case, and comes off as the print head rubs it on the paper. So the simple let's-darken-the-print reinking solution is to run the ribbon without touching the paper. In other words, command the computer to print blank spaces in an endless loop.

This trick won't work if you have an expensive, smart bidirectional printer that evaluates characters coming out of its own 640K buffer. But it will work with some cheap-o Commodore models. Be sure to take the paper out before you try this, because paper feeds through whether it's printed on or not.

```
10 OPEN 7,4
20 PRINT#7, "
30 GOTO 20
```

RIBBON REINKING #2

Drop some sewing machine oil on the ribbon as it goes through the cartridge. (Someone suggested mayonnaise will work too, but it sounds awful.) The oil thins the ink, and makes it go a little farther.

EMERGENCY REINKING #3

Hold a fresh rubber stamp pad against the ribbon as it travels through its path. Then wash your hands. This is messy.

BLURRY PRINTING

Print heads wear out and the printing gets blurry. Before you start requesting repairs, have someone else look at your listing for a second opinion. You may need glasses instead.

UNDERLINE TRICK

Some printers don't have underline capabilities. To get around this, print CHR\$(141) at the end of the line. (It's a RETURN without a line feed.) Then PRINT the correct number of underline characters, and go on.

```
10 OPEN 7:4
20 PRINT#7, "HA! I'LL MAKE YOU
   UNDERLINE";CHR$(141);
30 FOR X = 1 TO 28: PRINT#7,"{CMD-@}";:NEXT
40 PRINT#7: CLOSE 7
```

PRINTER NOISE

A thick felt pad under the printer dramatically cuts down on printer noise. Besides, everyone likes a padded seat.

APPENDIX A

64 AND 128 BASIC COMMANDS

Hey, pay particular attention to the functions that are available in 64 mode and those available in the 128's BASIC version 7.0. This will make conversion of programs from one machine to another much easier. Note that all the 64 commands are usable on the 128, but not all the 128 logic makes sense to the Commodore 64.

64 and 128 ABS()

This is a numeric operation that gives the absolute value of a number. It will make any negative number into a positive value.

64 and 128 AND

A logic operator that tests to see if arguments on both sides of the AND statement are true.

128 APPEND #

A file-handling command to combine the contents of two or more files. It eliminates some tricky programming steps necessary with the 64.

64 and 128 ASC

An examination function that gives the Commodore ASCII value for the first element of a string.

64 and 128 ATN()

A numeric operator that returns the value of an angle measured in radians that has the tangent value specified within the parentheses.

128 AUTO n

Another BASIC 7.0 command to make programming easier. This turns on (or off) automatic line numbering at the line number n specified.

128 BACKUP D# to D#

A DOS command for dual disk drives. It copies the entire contents of one disk to another in the second specified drive.

128 BANK #

Used to designate a specific memory bank (0–15) from within BASIC 7.0.

128 BEGIN BEND

A conditional statement where, if true, a series of statements can be executed, until the BEND (B-END) portion of the statement is found. If the argument is false, all lines between BEGIN and BEND will be passed over.

128 BLOAD

A DOS command to load a binary file. A familiar sight to Apple programmers.

128 BOOT

Automatically loads and runs a machine language program from the disk.

128 BOX

A graphics command to draw a filled square or rectangle.

128 BSAVE

Like BLOAD above, this command is used with binary files to save information to the diskette.

128 BUMP #

Used with sprite graphics to see which sprites (0–7) have collided.

128 CATALOG

A disk command to display the contents of the diskette directory without upsetting the program in memory.

128 CHAR

Used to place a character on the screen and set its individual characteristics.

64 and 128 CHR\$()

Used to find or print the character associated with the specified Commodore ASCII code number.

128 CIRCLE

A graphics command similar to BOX, used instead to place an ellipse on the screen.

64 and 128 CLOSE #

A DOS command to close an open disk file.

64 and 128 CLR

A BASIC command to clear the variables stored in memory.

64 and 128 CMD #

This command directs the operating output to a specified device number.

128 COLLECT D#

Used to optimize disk space by freeing any unnecessary records or files from the diskette.

128 COLLISION

Used to direct the program to a specified line number in the program *if* sprites have collided.

128 COLOR #,#

A command to specify one of seven screen formats (0–6) and to further assign one of the 16 available colors (0–15).

128 CONCAT

A DOS command to merge the contents of two files on the disk.

64 and 128 CONT

For CONTInue; use to restart the operation of the program halted by STOP or END.

128 COPY

In its different expressions, COPY can be used to copy one, some, or all of the files from one disk to a second drive. On a one-disk drive system, the COPY command can be used to rename a file.

64 and 128 COS

A mathematical operation to return the cosine for an angle. The value is expressed in radians.

64 and 128 DATA

Used in BASIC in conjunction with a READ statement to provide the necessary information for the given operation. Elements of a DATA statement must be separated by commas.

128 DCLEAR D#

This disk command clears all the open channels on the given drive number.

128 DCLOSE D#

This disk command clears all the open files on the given drive number.

128 DEC

Provides the decimal value of a hexadecimal quantity.

64 and 128 DEF FN () =

Used to create shorthand command to execute a mathematical operation that will occur many places in a program.

128 DELETE # - #

Deletes a range of line numbers in a BASIC program.

64 and 128 DIM ()

Specifies the number of elements in an upcoming array with more than ten items.

128 DIRECTORY

A command to look at the files on a disk in the drive given.

128 DLOAD

Loads a BASIC program from the disk.

128 DO LOOP (DO UNTIL WHILE EXIT)

All statements or BASIC line numbers will be executed for the specified number of loops, between the command DO and its counterpart LOOP; unless an evaluator of UNTIL, WHILE, or EXIT is encountered in its true state.

128 DOPEN

Opens a disk file.

128 DRAW

A graphics command for placing shapes, lines, and defined color sources onto the screen.

128 DSAVE

Saves a BASIC program on the disk.

128 DVERIFY

Used to check a program on the given disk drive against a program in memory.

128 EL

Gives the program line number when an error is encountered. See TRAP.

64 and 128 END

Signals the finishing point of the program.

128 ENVELOPE

For defining the quality of a sound to be produced. Its specifications determine the attack, decay, sustain, release rate, waveform, and pulse. Any parameter not specified will retain its previous characteristics. The Commodore 128 has ten predefined envelopes for 0 piano, 1 accordion, 2 calliope, 3 drum, 4 flute, 5 guitar, 6 harpsichord, 7 organ, 8 trumpet, and 9 xylophone.

128 ER

Provides the error condition found. See TRAP.

128 ERR\$

Gives the error message encountered. See TRAP.

128 EXIT

This command can be used as part of a DO LOOP statement.

64 and 128 EXP (n)

A numeric function to calculate the constant e at the given n.

128 FAST

A command to begin the 2-MHZ mode. It causes the screen to go blank. Operations and graphics will not become visible until the program encounters the SLOW command.

128 FILTER

Defines sound from the SID chips parameters.

64 and 128 FOR NEXT loops (FOR TO STEP NEXT)

Defines a section of the program code (between the FOR and the NEXT statement) to be continually executed until the value specified after the TO statement is reached. The STEP command controls the way the counter increments the values when adding one to each value is not desired.

64 and 128 FRE

Used to find the number of free bytes available.

64 and 128 GET

Looks for one character from the keyboard, but does not wait for a key to be pressed.

128 GETKEY

Looks for a single character, but waits for a keypress. GETKEY can also wait for multiple keypresses, for instance, the command GETKEY A\$,B\$,C\$,D\$ will wait until four keys have been pressed, and will assign the corresponding key to the variables in the sequence.

64 and 128 GET#

Like the GET statement above, but instead of looking for keyboard input, GET# is used with file data.

128 GO64

Switches to Commodore 64 mode of operation.

64 and 128 GOSUB ##### (RETURN)

Upon encountering a GOSUB statement, the program will jump to the line number specified and execute all commands in the subroutine until a RETURN is found—causing the operation to go back to the previous location.

64 and 128 GOTO #####

An unconditional jump to the specified line number.

128 GRAPHIC

Sets the 128 into one of the six graphics modes; 0 40-column text, 1 standard bit-mapped graphics, 2 split-screen bit-mapped graphics, 3 multicolor bit-mapped graphics, 4 split-screen multicolor bit-mapped graphics, 5 80-column text mode.

128 GSHAPE

See SSHAPE GSHAPE.

128 HEADER

Formats a disk.

128 HELP

When used after an error has occurred, it identifies the line number of a portion of the code where the error existed.

128 HEX\$()

Returns the four-character hexadecimal representation of a decimal number.

64 and 128 IF THEN

When the condition between the IF and THEN statement is true, the program will execute the command statements following the THEN statement. If the argument is false the program will not read any further into that line, and will drop to the next line number.

128 [IF THEN] ELSE

The 128 has the ability to hold a conditional clause to the IF/THEN statement. The condition will be operated upon provided the ELSE resides within the same line number as the rest of the statement.

64 and 128 INPUT

Requests data from the user and waits until the RETURN key is pressed.

64 and 128 INPUT#

Gets data from a disk file rather than from the user.

64 and 128 INT()

A numeric operation to round a value to its nearest whole number.

128 KEY

Used to assign a string of functions to the eight special functions keys (F1 to F8) on the keyboard.

64 and 128 LEFT\$(,#)

This function returns a specified number of left-most characters from the string specified within the parentheses.

64 and 128 LEN()

Used to determine the length of a string.

64 and 128 LET

Assigns a variable to an expression.

64 and 128 LIST

Displays the line listings of a program in memory.

64 and 128 LOAD

The command to recall a program stored on the diskette.

128 LOCATE X,Y

For placing the pixel cursor (PC) at any specified location on the high-resolution screen. X is the column designation, Y specifies the row coordinate.

64 and 128 LOG()

A mathematical operator that provides the the natural logarithm of a number.

128 LOOP

See DO LOOPS.

64 and 128 MID\$(,#,#)

Used to find a portion of the string specified immediately within the parentheses, starting at a character position number, for the length given.

128 MONITOR

Used to enter the machine language monitor on the 128.

128 MOVESP #,X,Y

To position a sprite designated by its number, to the X and Y coordinates given. Besides absolute coordinates, the X and Y expressions can indicate a relative position, and speed of movement.

64 and 128 NEW

This command erases a program from the computer's memory, as well as emptying variable storage areas.

64 and 128 NEXT

See FOR NEXT loops.

64 and 128 NOT

Used to find the logical NOT of an expression being evaluated.

64 and 128 ON

Specifies conditional branching to a portion of the program (line number) according to the results of the statement execution. By using the ON...GOTO, or ON...GOSUB combination, several lines of IF/THEN evaluations can be eliminated.

64 and 128 OPEN

Opens files for input or output.

64 and 128 OR

A logic operator that tests to see if an argument on either side of the OR statement is true.

128 PAINT

A graphics statement used to fill a designated area on the screen.

64 and 128 PEEK()

This statement returns the decimal numerical value of the specified memory location.

128 PI [pi symbol n]

Provides the value of 3.14159265.

128 PLAY

Defines musical notes and elements of the tone. The specifications in the statement determine, in order: voice, octave, tune envelope, volume, and filter.

64 and 128 POKE #####,n

Used to locate a specific memory location, and change the value stored there to the decimal amount of n.

64 and 128 POS()

Finds the cursor position within the window specified at the time the statement is encountered.

64 and 128 PRINT(or ?)

The most often used method of placing output to the screen.

64 and 128 PRINT#

Used to direct the output to a peripheral device, such as the disk drive or printer.

128 PRINT USING

A statement to define the display output characteristics of the string that will be printed. Commonly used to establish the number of decimal places and the actual location of numeric evaluations.

128 PUDEF "nnnn"

Redefines the PRINT USING statement. Up to four characters can be used.

64 and 128 READ

This statement directs operation to the DATA statements to gather information. See DATA.

128 RECORD

Positions a relative file pointer to any location within a record of a relative file.

64 and 128 REM (REMark)

Used to add comments to program code. When the REM statement is encountered by the program, all line information following will be passed over.

128 RENAME

A command to issue a new name to a file on the disk.

128 RENUMBER n,n2,#

Directs the system to renumber the BASIC line numbers starting with a line number of n, incrementing by a value of n2, beginning at the line number specified.

64 and 128 RESTORE()

Resets the READ pointer so that previously used data can be reevaluated.

128 RESUME()

Sends program operation back to the line specified within the parentheses once error evaluation is complete. See TRAP.

64 and 128 RETURN

See GOSUB.

128 RGR()

Used to determine the current graphics mode.

64 and 128 RIGHT\$(,n)

This function returns a specified number of right-most characters from the string specified within the parentheses.

64 and 128 RND

A function to generate a random number within the range specified. For instance, the expression $X = \text{INT}(\text{RND}(1) * 16)$ would return a whole number value between 0 and 15.

128 RSPCOLOR()

Finds the color value of a given sprite.

128 RSPPOS #,n

Returns the identification of the sprite number give, and its speed and position n.

128 RSPRITE

Provides the identifying characteristics of any sprite defined. The characteristics are: 0 enabled/disabled, 1 color, 2 front/behind objects, 3 expanded in X direction, 4 expanded in Y direction, 5 multicolor.

64 and 128 RUN

Starts the program in memory.

128 RWINDOW (n)

Size of the current window, i.e., when n equals 0, the number of lines are given, when n = 1 number of rows equals 40, and when n = 2 the number of rows equals 2 80.

128 SCALE

SCALE 1 turns on scaling. SCALE 0 turns off scaling.

128 SCNCLR

Clears the screen in the graphics mode specified.

128 SCRATCH

Used to erase files from the disk.

64 and 128 SGN()

Returns the sign, whether positive or negative, for the number or variable given.

64 and 128 SIN()

Used to identify the sine of a number. Results are expressed in radians.

128 SLEEP()

A command to delay execution and pause at the location of this statement for the number of seconds specified. 65535 is the maximum value allowed.

128 SLOW

Returns the computer to the one MHz speed after encountering the FAST command.

128 SOUND

Use to set the parameters of a tone, such as voice, frequency, duration, direction, minimum frequency, step value, waveform, and pulse width.

64 and 128 SPC()

Used in conjunction with PRINT and PRINT USING statements to space a given number of places to the right.

128 SPRCOLOR

Use to set multicolor sprites.

128 SPRDEF

This command allows entry into the 128's sprite editor.

128 SPRSAV

Saves sprite images.

64 and 128 SQR()

Finds the square root of the number given within the parentheses.

128 SSHAPE GSHAPE

Statements used to define a rectangular bit-mapped area of the screen to be saved or recalled.

64 and 128 ST

Looks at the status byte after an I/O operation has been completed.

64 and 128 STEP

See FOR NEXT.

64 and 128 STR\$()

A function to provide the string representation of the value contained within the parentheses.

64 and 128 SYS #####

Calls and executes a machine language program at the given address.

64 and 128 TAB()

Used with PRINT statements to move or “tab” to a horizontal position on the screen or printer output.

64 and 128 TAN()

A numeric evaluator that provides the tangent of a given angle, expressed in radians.

128 TEMPO()

Determines the execution speed of musical statements. Speed can be expressed as a decimal value between 0 and 255.

64 and 128 TI and TI\$

Reserved system variables used to respond to functions involving the internal clocks of the Commodore.

128 TRAP

An error detection statement that directs program execution to a designated location in the program if an error is encountered. See EL, ER, ERR\$, and RESUME.

128 TRON TROFF

Statements to turn on, or off, the trace mode when debugging BASIC programs.

128 UNTIL

See DO LOOP.

64 and 128 USR()

Calls a user defined subprogram by jumping to that machine language program.

64 and 128 VAL()

A function to convert a string to its numeric value for mathematical processing.

64 and 128 VERIFY

This command is most often used after a SAVE to be sure the version copied to disk is the same as the one in memory.

128 VOL

Sets the volume in a music statement.

64 and 128 WAIT

Suspends program execution until a given memory address recognizes a specific bit pattern or character.

128 WHILE

See DO LOOP.

128 WIDTH

Sizes the pixel cursor (PC). WIDTH 1 sets a single size; WIDTH 2 sets double width.

128 WINDOW

The statement command identifies a rectangular area of the screen. X1,Y1 is the top left—X2,Y2 the bottom right.

128 XOR(n1,n2)

A function to provide the exclusive OR of the argument values of n1 and n2, where the maximum value for either does not exceed 65535.

APPENDIX B

ABBREVIATIONS FOR BASIC KEYWORDS

Why type all those command words when you are entering a BASIC program? There's a shortcut; Commodore has provided a way to let-your-fingers-do-the-resting. Use the abbreviations below in uppercase/graphics mode. Press the letter key(s), then hold down the SHIFT key and type the letter following the word SHIFT. Later, when you list your program, all the abbreviations will be automatically changed by the system into the correct long format command structure.

Keyword	Key Combination
ABS	A SHIFT-B
APPEND	A SHIFT-P
ASC	A SHIFT-S
ATN	A SHIFT-T
AUTO	A SHIFT-U
BACKUP	BA SHIFT-C
BANK	B SHIFT-A
BEGIN	B SHIFT-E
BEND	BE SHIFT-N
BLOAD	B SHIFT-L
BOOT	B SHIFT-O
BOX	none

Keyword	Key Combination
BSAVE	B SHIFT-S
BUMP	B SHIFT-U
CATALOG	C SHIFT-A
CHAR	CH SHIFT-A
CHR\$	C SHIFT-H
CIRCLE	C SHIFT-I
CLOSE	CL SHIFT-O
CLR	C SHIFT-L
CMD	C SHIFT-M
COLINT	none
COLLECT	COLL SHIFT-E
COLLISION	COL SHIFT-L
COLOR	COL SHIFT-O
CONCAT	C SHIFT-O
CONT	none
COPY	CO SHIFT-P
COS	none
DATA	D SHIFT-A
DCLEAR	DLC SHIFT-E
DCLOSE	D SHIFT-C
DEC	none
DEF FN	none
DELETE	DE SHIFT-L
DIM	D SHIFT-I
DIRECTORY	DI SHIFT-R
DLOAD	D SHIFT-L
DO	none
DOPEN	D SHIFT-O
DRAW	D SHIFT-R
DSAVE	D SHIFT-S
DVERIFY	D SHIFT-V
EL	none
END	none
ENVELOPE	E SHIFT-N
ER	none

Keyword	Key Combination
ERR\$	E SHIFT-R
EXIT	EX SHIFT-T
EXP	E SHIFT-X
FAST	none
FETCH	F SHIFT-E
FILTER	F SHIFT-I
FN	none
FOR	F SHIFT-O
FRE	F SHIFT-R
GET	G SHIFT-E
GETKEY	GETK SHIFT-E
GET#	none
GOSUB	GO SHIFT-S
GO64	none
GOTO	G SHIFT-O
GRAPHIC	G SHIFT-R
GSHAPE	G SHIFT-S
HEADER	HE SHIFT-A
HELP	none
HEX\$	H SHIFT-E
IF/THEN	none
INPUT	none
INPUT\$	I SHIFT-N
INSTR	IN SHIFT-S
INT	none
JOY	J SHIFT-O
KEY	K SHIFT-E
LEFT\$	LE SHIFT-F
LEN	none
LET	L SHIFT-E
LIST	L SHIFT-I
LOAD	L SHIFT-O
LOCATE	LO SHIFT-C
LOG	none
LOOP	LO SHIFT-O

Keyword	Key Combination	
MID\$	M	SHIFT-I
MONITOR	MO	SHIFT-N
MOVESHAPE		none
MOVESP	M	SHIFT-O
NEW		none
NEXT	N	SHIFT-E
OPEN	O	SHIFT-P
PAINT	P	SHIFT-A
PEEK	PE	SHIFT-E
PEN	P	SHIFT-E
PI		none
PLAY	P	SHIFT-L
POKE	PO	SHIFT-K
POS		none
POT	P	SHIFT-O
PRINT		?
PRINT USING	?US	SHIFT-I
PUDEF	P	SHIFT-U
RBUMP	RB	SHIFT-U
RCLR	R	SHIFT-C
RDOT	R	SHIFT-D
READ	RE	SHIFT-A
RECORD	R	SHIFT-E
REM		none
RENAME	RE	SHIFT-N
RENUMBER	REN	SHIFT-U
RESTORE	RE	SHIFT-S
RESUME	RES	SHIFT-U
RETURN	RE	SHIFT-T
RGR	R	SHIFT-G
RIGHT\$	R	SHIFT-I
RLUM		none
RND	R	SHIFT-N
RREG	R	SHIFT-R
RSPCOLOR	RSP	SHIFT-C

Keyword	Key Combination
RSPPOS	R SHIFT-S
RSPR	none
RSPRITE	RSP SHIFT-R
RUN	R SHIFT-U
RWINDOW	R SHIFT-W
SAVE	S SHIFT-A
SCALE	SC SHIFT-A
SCNCLR	S SHIFT-C
SCRATCH	SC SHIFT-R
SGN	S SHIFT-G
SIN	S SHIFT-I
SLEEP	S SHIFT-L
SLOW	none
SOUND	S SHIFT-O
SPRCOLOR	SPR SHIFT-C
SPRDEF	SPR SHIFT-D
SPRITE	S SHIFT-P
SPRSAY	SPR SHIFT-S
SQR	S SHIFT-Q
SSHAPE	S SHIFT-S
STASH	S SHIFT-T
STEP	ST SHIFT-E
STOP	ST SHIFT-O
STR\$	ST SHIFT-R
SWAP	S SHIFT-W
SYS	none
TAB	T SHIFT-A
TAN	none
TEMPO	T SHIFT-E
TI	none
TI\$	none
TO	none
TRAP	T SHIFT-R
TROFF	TRO SHIFT-F
TRON	TR SHIFT-O

Keyword	Key Combination
UNTIL	U SHIFT-N
USR	U SHIFT-S
VAL	none
VERIFY	V SHIFT-E
VOL	V SHIFT-O
WAIT	W SHIFT-A
WHILE	W SHIFT-H
WIDTH	WI SHIFT-D
WINDOW	W SHIFT-I
XOR	X SHIFT-O

APPENDIX C

COMMODORE CHARACTER CODES

This appendix will show you almost all the characters that will appear if you `PRINT CHR$(n)`. The value of `n` must be between 0 and 255. For the most part, the `CHR$` codes are the same on the 64 and 128. If “64” or “128” appears in the function description, then the `CHR$` code operation is for that computer only. So program away.

This chart will not show the exact composition of the graphics characters. To get all those little symbols, type the following program. Then color in this chart, and remember to stay within the lines.

Notice that in line 20 we have used one of the `CHR$` commands in a statement. `PRINT CHR$(5)` tells the system to always revert back to printing white characters to the screen. This way we can continue to see everything—unless, of course the background color on the monitor is set to white—and if so, change it. Now!

```
10 FOR X = 0 TO 255: PRINT X, CHR$(X)
20 PRINT CHR$(5): FOR T = 1 TO 100: NEXT: NEXT
```

You may want to direct this program to a graphics printer, and make your own free chart that does not need to be colored in. Hook everything together, or use tape, nails, and super glue, and type:

```

10 OPEN 7,4: FOR X = 0 TO 255
20 PRINT#7, X, : PRINT#7,CHR$(X)
30 NEXT: CLOSE 7

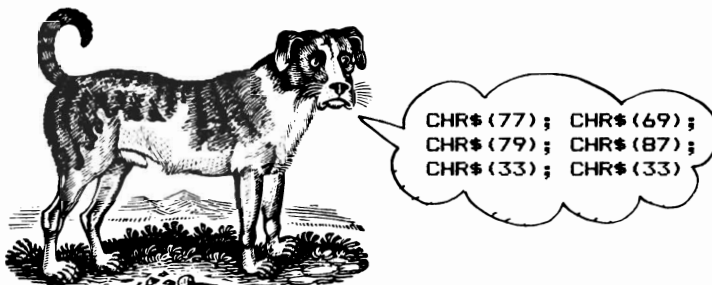
```

WARNING—YOUR PRINTER MAY GO CRAZY. Some of the CHR\$ codes in the following chart are printer commands that will make the printing go into expanded/wide mode, force form and line feeds, or go into reverse printing mode. Reverse [CHR\$(18)] is the worst. You may, therefore, wish to add this line to the free chart PRINT routine:

```

15 IF X = 18 THEN NEXT

```



Code	Function
CHR\$(0)	
CHR\$(1)	
CHR\$(2)	Underline (80) on 128
CHR\$(3)	
CHR\$(4)	
CHR\$(5)	
CHR\$(6)	Print white characters
CHR\$(7)	Produce bell tone on 128
CHR\$(8)	Disable character set change on 64
	Print in graphics mode on 64
CHR\$(9)	Enable character set change on 64
CHR\$(10)	Printer line feed/carriage return on 64

Code	Function
CHR\$(11)	Enable character set change on 128
CHR\$(12)	Disable character set change on 128
CHR\$(13)	Form feed to printer
	Line feed to computer
	Line feed to printer
CHR\$(14)	Enter a line of BASIC
	Set character set to upper- or lowercase
	Print double-width characters on printer
CHR\$(15)	Turn on flash (80) on 128
	Print standard with character on printer
CHR\$(16)	Address of start print position
CHR\$(17)	Move cursor down one row on screen or printer
CHR\$(18)	Reverse printing on printer or screen
CHR\$(19)	Clear, Home; move cursor to top left corner of screen
CHR\$(20)	Delete last character, move all characters to the right one space left to close gap
CHR\$(21)	
CHR\$(22)	
CHR\$(23)	
CHR\$(24)	Tab set or clear on 128
CHR\$(25)	
CHR\$(26)	
CHR\$(27)	Escape; used in conjunction with other codes
CHR\$(28)	Set character color to red
CHR\$(29)	Move cursor one column to the right
CHR\$(30)	Set character color to green
CHR\$(31)	Set character color to blue
CHR\$(32)	Space or spacebar
CHR\$(33)	!
CHR\$(34)	"
CHR\$(35)	#
CHR\$(36)	\$
CHR\$(37)	%
CHR\$(38)	&

Code	Function
CHR\$(39)	'
CHR\$(40)	(
CHR\$(41)	)
CHR\$(42)	*
CHR\$(43)	+
CHR\$(44)	'
CHR\$(45)	-
CHR\$(46)	.
CHR\$(47)	/
CHR\$(48)	0
CHR\$(49)	1
CHR\$(50)	2
CHR\$(51)	3
CHR\$(52)	4
CHR\$(53)	5
CHR\$(54)	6
CHR\$(55)	7
CHR\$(56)	8
CHR\$(57)	9
CHR\$(58)	:
CHR\$(59)	;
CHR\$(60)	<
CHR\$(61)	=
CHR\$(62)	>
CHR\$(63)	?
CHR\$(64)	@
CHR\$(65)	A
CHR\$(66)	B
CHR\$(67)	C
CHR\$(68)	D
CHR\$(69)	E
CHR\$(70)	F
CHR\$(71)	G
CHR\$(72)	H
CHR\$(73)	I

Code	Function
CHR\$(74)	J
CHR\$(75)	K
CHR\$(76)	L
CHR\$(77)	M
CHR\$(78)	N
CHR\$(79)	O
CHR\$(80)	P
CHR\$(81)	Q
CHR\$(82)	R
CHR\$(83)	S
CHR\$(84)	T
CHR\$(85)	U
CHR\$(86)	V
CHR\$(87)	W
CHR\$(88)	X
CHR\$(89)	Y
CHR\$(90)	Z
CHR\$(91)	[
CHR\$(92)	£
CHR\$(93)	]
CHR\$(94)	↑
CHR\$(95)	←
CHR\$(96) to CHR\$(128)	Graphics characters
CHR\$(129)	Set character color to orange (40) purple (80)
CHR\$(130)	Underline off (80)
CHR\$(131)	Run a program. Note that this does not work with PRINT CHR\$(131), but works from keyboard buffer
CHR\$(132)	
CHR\$(133)	F1 Special function key
CHR\$(134)	F3
CHR\$(135)	F5
CHR\$(136)	F7
CHR\$(137)	F2

Code	Function
CHR\$(138)	F4
CHR\$(139)	F6
CHR\$(140)	F8
CHR\$(141)	Send a carriage return without entering a BASIC line
CHR\$(142)	Set character set to uppercase/graphics
CHR\$(143)	Turn off flash (80) on the 128
CHR\$(144)	Set character color to black
CHR\$(145)	Move cursor or printing position up one row
CHR\$(146)	End reverse printing on screen or printer
CHR\$(147)	Home, clear screen; move cursor to top left of screen
CHR\$(148)	Insert, move all characters from cursor position right toward end of line one column
CHR\$(149)	Set character color to brown (40) yellow (80)
CHR\$(150)	Character color to light red
CHR\$(151)	Character color to dark gray (40) dark cyan (80)
CHR\$(152)	Character color to medium gray
CHR\$(153)	Character color to light green
CHR\$(154)	Character color to light blue
CHR\$(155)	Character color to light grey
CHR\$(156)	Character color to purple
CHR\$(157)	Move cursor left one column
CHR\$(158)	Character color to cyan (40) light cyan (80)
CHR\$(159) to CHR\$(191)	Graphics characters
CHR\$(192) to CHR\$(223)	Same as 96–127
CHR\$(224) to CHR\$(254)	Same as 160–190
CHR\$(255)	Same as 126

APPENDIX D
**ESCAPE CODES
FOR THE 128**

All the escape codes listed below are for the Commodore 128 only. That's because the 64 doesn't have a REAL escape key. The codes listed are for system manipulation within the computer, and shouldn't be confused with those standard escape control codes used to format printer output. Okay? Got it?

Escape Key	Function
ESC-C	Cancel quote/insert mode
ESC-Q	Erase to end of current line
ESC-P	Erase to start of current line
ESC-@	Clear to bottom of screen
ESC-J	Move to start of the current line
ESC-K	Move to end of the current line
ESC-O	Enable overstrike mode
ESC-A	Enable auto-insert mode
ESC-D	Delete current line
ESC-I	Insert a line at the cursor position
ESC-Y	Set default tab stops (eight spaces)
ESC-Z	Clear all tabs
ESC-L	Enable scrolling
ESC-M	Disable scrolling

Escape Key	Function
ESC-V	Scroll up
ESC-W	Scroll down
ESC-G	Bell tone ding
ESC-H	Disable bell tone
ESC-E	Nonflashing cursor
ESC-F	Flashing cursor
ESC-T	Set top of window to cursor position
ESC-B	Set bottom of window to cursor position
ESC-X	Switch between 40- and 80-column mode

For 80-column screen display only:

ESC-U	Underline cursor
ESC-S	Block cursor
ESC-N	Normal screen video output
ESC-R	Reverse video screen output

INDEX

A

Abbreviations
 keywords, 157–162
 printing command, 133
Advanced programming
 control codes, suggestions for changes, 90
 data input, 94
 custom input prompts, 94
 get the best input, 95–96
 Get statements, 95
 dynamic keyboard technique, 92–93
 error elimination, 88
 GOTO, use without line number, 91
 moose call routine, 91–92
 multicolor listing, 88–89
 READY, erasing prompt, 87–89
 rounding errors, correcting, 93–94
 structured listings, 89–90
Amortization, 76
Applications
 amortization, 76
 change-making program, 75–76
 decimal points, lining up, 73
 dollar formatting, 74
 justified numbers, on right, 74
 message flasher, 72
 printer printing, centered, 73

 screen printing, centered, 72–73
 sorting, 71
Arithmetic calculations
 integers, 25
 LET statements, 20–21
 rounding errors, correcting, 93–94
 symbols/meaning, 23–24
Autodialer, telephone, 124

B

Bank configuration, hex on 128, 100–101
BASIC, 11
 commands (Appendix A), 141–156
 keywords/abbreviations (Appendix B), 157–162
Baud rate, 123
Bells, key combinations/results, 58–59
Binary to decimal conversions, 99–100
Buzzer, 127–128

C

Change-making program, 75–76
Characters
 character codes (Appendix C), 163–168
 protecting image of, 81

CHR\$ codes, printing, 137–138
 CLEAR-HOME, line numbering, 67
 CLOSE, printing command, 132
 CLR command, 135
 memory, 65
 CMD command, printing, 132
 Color, 58–59
 key combinations/results, 58–59, 89
 multicolor listing, 88–89
 , comma, 16
 Commands, BASIC, commands (Appendix A),
 141–156
 See also specific commands.
 Concatenation, concat, disk command, 120–121
 Control codes, suggestions for changes,
 advanced programming, 90
 COPY, disk command, 119
 Crash, rebooting after, 114
 Crunching, saving space, 17, 49
 Cursor
 placement by subroutine, 55–56
 speed, 69
 use of, 55

D

Data input
 advanced programming, 94
 get the best input, 95–96
 Get statements, 95
 input prompts, 94
 DCLEAR on 128, 65
 Decimal points, lining up, 73
 Decimal to binary conversion, 100
 Decimal to hexadecimal conversion,
 99
 Delay loops, speed, 69
 Direct mode, 12, 30
 Directory
 disk command, 118
 disk directory protection tricks, 84, 115–117
 loading, and losing program, 114
 Disk commands
 concat, 120–121
 copy, 119
 directory, 118
 format, 118
 initialize, 119–120
 rename, 119
 SCRATCH, 119
 validate, 119
 verify, 120
 Disk directory, printing, 135

Disk drive
 placement of, 3
 red light, 32, 112
 program for finding error, 112–113
 save and replace command, 117
 silencer for, 108
 as smart peripheral, 110
 Disks
 copy protection, 78–80
 disk, as library of directory, 45
 filling up, rule against, 45
 formatting, 111
 saving program, 32
 sectors of, 109
 single-sided, using both sides, 112
 switching disks, 110
 tracks of, 109
 write-protect tabs, 8
 Dollar formatting, 74
 DOS, 107–108
 Double space, 135
 Dumping, program, 134
 Dynamic keyboard technique, advanced
 programming, 92–93

E

Echo auto-dump 128, 134
 Error elimination, advanced programming, 88
 Error messages, 11
 Escape codes, for 128 (Appendix D), 169–170

F

F8 key, into ML Monitor, in 128, 102–104
 Formatting
 command for, 118
 disks, 111
 FRE command, memory, 65
 Function keys, 28–29
 reprogramming, 29

G

GOTO, use without line number, advanced
 programming, 91
 Graphics, 53
 bit mapped graphics, 53
 colors, printing on screen, 58–59
 mazemaking routines, 57
 pixel-by-pixel programming, 53
 sprites, 54
 text graphics, 53

H

Hexidecimal to decimal conversion, 99

I

Initialize, disk command, 119–120

Input crash protection, 86

letters and numbers, 86

letters only, 86

numbers only, 86

Integers, 25

J

Justified numbers, on right, 74

K

Keyboard, 27–30

function keys, 28–29

HELP, 31

LIST, 31

lowercase/uppcase mode, 27

RETURN keys, 27

RUN/STOP keys, 27

symbols, review of, 30

uppercase/graphics mode, 27

Key repeats, protection, 82

Keywords, 11

abbreviations, 66, 157–162

L

LET, 20–21

Lines

numbering

automatic programs, 35, 67

CLEAR-HOME, 67

removing, 15

renumbering, 35

LIST tricks, protection, 82–83

Loading from directory, programs, 34

M

Machine language, 97–106

bank configuration, hex on 128, 100–101

binary to decimal conversions, 99–100

decimal to binary conversions, 100

decimal to hexadecimal conversion, 99

F8 key, into ML Monitor, on 128, 102–104

getting program into memory, 97–98

hexidecimal to decimal conversion, 99

Monitor commands on 128, 104–105

opcodes, 104

operating system, 97

placement of routines, on 64, 105–106

speed check, 98

VERIFY, 106

Mazemaking routines, 57

Memory

CLR command, 65

DCLEAR on 128, 65

FRE command, 65

programming tips, 64–65

WAIT statement, 66

Message flasher, 72

Modems, 123

baud rate, 123

connection number, 125

Monitors, 4–5

color adjustment, 5

commands on, 128, 104–105

monitor cables, 126

TV set, buzz from, 128

Moose call routine, advanced programming, 91–92

Multicolor listing, advanced programming, 88–89

Musical instruments, sounds, 129–130

O

ON statement, speed, 68

Opcodes, 104

OPEN, printing command, 132

Operating system, 97

Operation protection, 80

P

Paper saving tips, 136

Password tricks, 43–44

Placement of routines, on 64, 105–106

POKE, 68, 69

protection, 85

repeating keys, 67

Portable programs, guidelines for, 39–40

Power strips, use of, 6

PRINT, 12–13
 formatting, 15
 print zones, 16
 semicolons, eliminating, 48
 Printer, buying of, 137
 Printing
 blurry printing, 139
 centering, 137
 CHR\$ codes, 137–138
 CLR command, 135
 commands
 abbreviations, 133
 CLOSE, 132
 CMD, 132
 OPEN, 132
 Print #, 132
 disk directory, 135
 double space, 135
 echo auto-dump, 128, 134
 noise, cutting down, 140
 on 128, 133
 paper saving tips, 136
 print-here command, 136
 program dumping, 134
 ribbon re-inking, 138–139
 screen dumping, 134
 underline trick, 139
 Program conversion
 Apple to Commodore BASIC, 56
 commands to avoid, 41–42
 translation guide, 38
 Programs
 crunching programs, 17
 erasing, 33, 46–47
 examining directory, getting around bug in,
 49–50, 64
 LET statements, 20–21
 line numbering, automatic, 35, 67
 line renumbering, 35
 loading from directory, 34
 making changes in, 14
 new, 13–14
 password tricks, 43–44
 portable programs, guidelines for, 39–40
 REM statements, 19
 saving, 32–33
 saving space, crunching, 17, 49
 See also Applications;
 Advanced programming.
 Prompt, READY, erasing, 87–89
 Protection
 characters, protecting image of, 81
 directory protection, 84
 disk copy protection, 78–80

 disk directory protection tricks, 115–117
 input crash protection, 86
 letters and numbers, 86
 letters only, 86
 numbers only, 86
 key repeats, 82
 LIST tricks, 82–83
 operation protection, 80
 POKE method, 85
 SAVE tricks, 83–84
 unloading protection, 84

Q

Quotation marks
 quote mode, getting out, 35
 use of, 12, 13, 18

R

Radio, played through computer, 126
 Random numbers, use of, 50–51
 READY, erasing prompt, advance program-
 ming, 87–89
 REM, use of, 19
 Rename, disk command, 119
 Repeating keys, POKE, 67
 Ribbon re-inking, 138–139
 Rounding errors, correcting, 93–94
 Run time tricks, 47–48

S

SAVE tricks, protection, 83–84
 Saving, 32–33
 big/developing programs, 46
 bug related to, 33
 dynamic keyboard, 93
 fast save and backup, 62
 periodic saving, 45
 and VERIFY, 33
 Saving space, crunching programs, 17
 SCRATCH, 33, 46
 disk command, 119
 Screen
 blacking out screen, 126–127
 screen dumping, 134
 screen printing, centered, 72–73
 Scrolling
 horizontal scroll, 54
 vertical scroll, 54

Sorting, 71
Sounds
 buzzer, 127–128
 musical instruments, 129–130
 radio, played through computer, 126
 wake-up calls, 127
Speed
 cursor, 69
 delay loops, 69
 fast save, 62
 fast/slow on 61, 128
 line numbering, CLEAR-HOME, 67
 ON statement, 68
 programming tips, 63–64
 repeating keys, POKE, 67
 speed check, machine language, 98
 speeding up 1541 drive, 61–62
 zero, abbreviating, 67
Starting up
 burning in, 1
 cable tie-ups, 2
 control ports, 3
 disk drive placement, 3
 hook-up, 2
 monitors, 4–5
 power strip for equipment, 6
STOP, disabling key, 68
String variables, rules for, 21
Structured listing, advanced programming,
 89–90
Syntax, 11

T

Telephone, autodialer, 124
Television graphics, 126
Terminal emulator, 129
TV set, buzz from, 128

U

Underline trick, 139
Unloading protection, 84

V

Validate, disk command, 119
Variables, 11, 20
VERIFY, 34, 106, 120

W

Wake-up calls, 127

Z

Zero, abbreviating, 67

ABOUT THE AUTHORS

JOHN ANNALORO is the President of C.Ed. Software Development corporation, a programming house that specializes in the authoring of mass-market software releases for major publishers. His clients includes Harcourt Brace and Jovanovich, Simon & Schuster, Richard D. Irwin, MicroSci, Mastertronic International, and others. He is also a part-time professor teaching computer graphics and art at Southwestern College. As a writer, John Annaloro is the author of the SOFTWARE PUBLISHING HANDBOOK.

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 Scrapbook*. Some of Bert's best work can be found buried in Beagle Brothers' famous ads, tip books, and catalogs.

GEOS—New Operating System For The Commodore 64/64c/128

Eight-bit Commodores will never be the same again!

A desktop user interface, including pull-down menus, dialog boxes and proportionally spaced fonts, is the exciting visual hallmark of GEOS. Underlying this is a host of other features that will revolutionize the way people use and program the Commodore 64 and 128 series.

However, developing applications for GEOS will require proficiency in *event driven* programming—a method with which most Commodore programmers are not yet familiar.

Written by Berkeley Softworks, the creator of GEOS, here is the definitive guide for those who program under this new environment. Learn everything you need to know about GEOS, including:

Hi-Res Bit-Map Mode—Redefining the Commodore 64 screen for GEOS
Display Buffering—Managing the screen display under tight memory constraints
Input Drivers—Using the mouse to control your program
Variable Length Indexed Record (VLIR)—Breaking up files into records to speed up data movement
Disk Turbo—Speeding up disk access time for better memory control

Available wherever computer books are sold.

7" × 9 1/4" paperback edition

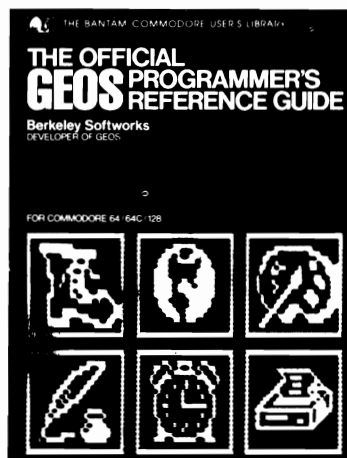
34404-8 \$19.95/\$24.95 in Canada

**For Fast, Convenient Service
CALL TOLL FREE
1-800-223-6834, extension 479**

Please have your Visa, Mastercard or
American Express Card available.

**A BANTAM COMPUTER BOOK
from BANTAM ELECTRONIC PUBLISHING**

Price and availability subject to change without notice.



Tips, Tricks, and Commodore Programming Savvy

Programming tips and tricks are timesavers that make a programmer's life much easier. **The Big Tip Book For The Commodore** contains some of the most popular and useful shortcuts ever designed for the Commodore 64/64c/128—tips and tricks which have been developed and used by other users and programmers.

While the authors provide an in-depth guide to the treatment of BASIC programming, this book also provides tips for dealing with hardware, disks and drives, printers and printing tips, communications, machine language, and every other facet of the Commodore 64/64c/128.

Newcomers and old-timers alike will appreciate the wide range of coverage presented in this book, which includes:

- BASIC programming do's and don'ts.
- How to use the function keys properly and creatively in your program.
- How to write programs that run on both C64 and C128.
- Introducing password access to prevent unauthorized use.
- How to control that annoying vertical scroll.
- Using PRINT statement to control screen colors.
- How to get more speed out of your trusty 1541 disk drive.
- How to center screen and printer output.

With over two million Commodore personal computers currently in active use, sharing tips and tricks not only makes perfect sense, but it's also fun. The programming strategies in this book have been collected with wit and wisdom by The Beagle Bros., a well-known software developer, who have included tips and tricks with their products for years.

"Most of the tips have come from my experience of writing software for the computer market place, I hope you find them to be educational and enjoyable."

—JOHN ANNALORO

John Annaloro is the author of **SOFTWARE PUBLISHING HANDBOOK**. He was involved in the development of **STAR TREK: THE KOBAYASHI ALTERNATIVES** and **J.K. LASSER'S YOUR INCOME TAX** computer programs. He is President of C.Ed. Software Development Corporation. Bert Kersey is founder of Beagle Bros., a leading software utility publisher. He also co-wrote **THE BIG TIP BOOK FOR THE APPLE II SERIES** and **DOSTALK SCRAPBOOK**.

Also in the Bantam Commodore User's Library:

The Official GEOS Programmer's Reference Guide
The Commodore 128 Programmer's Reference Guide
The Commodore 128 Subroutine Library
The Commodore 64 Survival Manual

