

# COMMODORE 64/128 POWER BASIC

More than two dozen powerful utilities and programs, written in BASIC, for the Commodore 64 and 128.



A **COMPUTE!** Books Publication

\$16.95





COMMODORE  
**64/128**  
**POWER**  
**BASIC**



**COMPUTE!** Publications, Inc. 

Part of ABC Consumer Magazines, Inc.  
One of the ABC Publishing Companies

Greensboro, North Carolina

The following articles were originally published in *COMPUTE!'s Gazette* magazine, copyright 1984, COMPUTE! Publications, Inc.:

"Auto Line Numbering" (February); "ASCII/POKE Printer" (March); "Numeric Keypad" (April); "Step Lister" (May); "String Search" (August); "Screen Headliner" (September); "Hi-Res Screen Dump" (October); "Slowpoke" (November); "Time Clock" (December).

The following articles were originally published in *COMPUTE!'s Gazette* magazine, copyright 1985, COMPUTE! Publications, Inc.:

"Stop and Go" (January); "Color Swap" (February); "Quick Character Transfer" (March); "Triple 64" (April); "Searchlight" (May); "Tape Program Rescue" (June); "Disk Title Changer" (July); "QuickScan" (September); "USR Joystick Reader" (October); "Screen Customizer" (November); "List Pager" (December).

The following articles were originally published in *COMPUTE!'s Gazette* magazine, copyright 1986, COMPUTE! Publications, Inc.:

"The Printmaker" (January); "Blink Mode on Commodore Machines" (February); "Keyboard to Joystick Converter" (March); "Input Windows" (April); "Blick" (May); "Help Screens" (June); "64 RAM Disk" (July); "BASIC Line Extender" (August); "Sound Off" (September); "Line Count" (October); "Instant Keywords" (November); "Program Mis-Matcher" (December).

Copyright 1987, COMPUTE! Publications, Inc. All rights reserved.

Reproduction or translation of any part of this work beyond that permitted by Sections 107 and 108 of the United States Copyright Act without the permission of the copyright owner is unlawful.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-87455-099-8

The authors and publisher have made every effort in the preparation of this book to insure the accuracy of the programs and information. However, the information and programs in this book are sold without warranty, either express or implied. Neither the authors nor COMPUTE! Publications, Inc., will be liable for any damages caused or alleged to be caused directly, indirectly, incidentally, or consequentially by the programs or information in this book.

The opinions expressed in this book are solely those of the authors and are not necessarily those of COMPUTE! Publications, Inc.

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is part of ABC Consumer Magazines, Inc., one of the ABC Publishing Companies, and is not associated with any manufacturer of personal computers. Commodore 64 and Commodore 128 are trademarks of Commodore Electronics Limited.

# Contents

---

---

<i>Foreword</i> .....	v
<b>Chapter 1. BASIC Programming Utilities</b> .....	1
Program Mis-Matcher	
<i>Mark Jordan</i> .....	3
Instant Keywords	
<i>Shawn K. Smith</i> .....	6
Line Count	
<i>Kenneth J. Rogerson</i> .....	9
Sound Off	
<i>John Martin</i> .....	14
BASIC Line Extender	
<i>Jay A. Reeve</i> .....	16
Searchlight	
<i>Katherine Myers</i> .....	20
Triple 64	
<i>Feeman Ng</i> .....	23
String Search	
<i>Glen Colbert</i> .....	25
String Lister	
<i>E. A. Cottrell</i> .....	29
ASCII/POKE Printer	
<i>Todd Heimarck</i> .....	32
Auto Line Numbering	
<i>Jeff Young</i> .....	35
<b>Chapter 2. Screen Control</b> .....	37
Help Screens	
<i>Jaffer Siddiqui</i> .....	39
Blick	
<i>Plummer Hensley</i> .....	45
Input Windows	
<i>Thorpe Thompson</i> .....	48
Blink Mode on Commodore Machines	
<i>David Sanner</i> .....	54
The Printmaker	
<i>Manu Gambhir</i> .....	64
Screen Customizer	
<i>Christian Elfers</i> .....	67

QuickScan	
<i>Daan Deenik</i> .....	70
Quick Character Transfer	
<i>Fabio Coronel</i> .....	72
Color Swap	
<i>Lee Noel, Jr.</i> .....	75
Time Clock	
<i>David W. Martin</i> .....	82
Slowpoke	
<i>Daniel R. Widyono</i> .....	85
Hi-Res Screen Dump	
<i>Gregg Peele</i> .....	89
Screen Headliner	
<i>Todd Heimarck</i> .....	93
<b>Chapter 3. Input/Output Routines</b> .....	<b>99</b>
64 RAM Disk	
<i>MiAngelo Moore</i> .....	101
Keyboard to Joystick Converter	
<i>David A. Dunn</i> .....	104
List Pager	
<i>Robert A. Stoerrle</i> .....	109
USR Joystick Reader	
<i>Tim Gerchmez</i> .....	114
Disk Title Changer	
<i>Michael Broussard</i> .....	117
Tape Program Rescue	
<i>John R. Hampton</i> .....	122
Stop and Go	
<i>Jim Pejsa</i> .....	124
Numeric Keypad	
<i>Charles Kluepfel</i> .....	126
<b>Appendices</b> .....	<b>129</b>
A. How to Type In Programs .....	131
B. The Automatic Proofreader / <i>Philip I. Nelson</i> .....	133
C. Screen Location Table .....	136
D. Screen Color Memory Table .....	137
E. 40-Column Screen Color Codes .....	138
F. Standard ASCII Codes .....	139
G. Screen Codes .....	142
H. Commodore 64 and 128 Keycodes .....	147
Index .....	151
Disk Coupon .....	153

# Foreword

---

---

COMPUTE! Publications has been publishing time-saving utilities for Commodore machines for years. Now, the best routines for the 64 and 128 home computers have been collected together in *Commodore 64/128 Power BASIC*. The 32 programs included here perform an amazing number of tasks that will become an indispensable part of your programming repertoire.

Divided into three sections, *Commodore 64/128 Power BASIC* includes BASIC programming utilities, screen-control utilities, and input/output routines. You'll find programs that perform a variety of tasks, including one that switches text characters between two different colors at varying speeds. Another program, "Slowpoke," enables you to control the speed of the PRINT statement—it's good for slowing down program listings and for debugging screen graphics. "Auto Line Numbering" is a handy utility that automatically generates a line number for the current BASIC program statement being entered. It begins with line 100 and increments by 10's, or you can modify it to suit your needs.

Although the Commodore's operating system does not directly support a RAM disk, "64 RAM Disk" gives you one. It makes 8K available for program storage, and you have almost instant access to the information stored there—without having to wait for data to be loaded from or saved to an external storage device. It's an excellent program-development tool.

These programs and many others make *Commodore 64/128 Power BASIC* a book you'll keep at your side as you're programming. You'll find numerous uses for these routines whether you're already an expert programmer or are just starting out. The appendices contain a wealth of supplementary material, including ASCII codes, a screen location table, and screen color codes.

Written in the clear, easy-to-understand style that COMPUTE!'s publications are known for, *Commodore 64/128 Power BASIC* is packed solid with programs that you'll wonder how you ever did without.

All the programs in *Commodore 64/128 Power BASIC* are ready to type in and run. However, if you prefer, you can purchase a disk that includes all the programs in the book. Call toll-free 1-800-346-6767 (in New York, 1-212-887-8496). Or use the coupon in the back of the book.



## CHAPTER 1

---

---

# BASIC Programming Utilities



# Program Mis-Matcher

---

---

*Mark Jordan*

*Programmers will appreciate this handy utility that compares two BASIC programs. Output is to screen or printer. Requires a disk drive. For the Commodore 64 and 128.*

It happens frequently when you are programming: You want to know exactly how the program you're working on differs from a previously saved version. Or maybe you have two versions and you're not sure which one to load because you can't remember which is more recent—GAME2 or GAMEB. Maybe you've been doing some experimenting with an old program and like some of the changes you've made, but not others. Wouldn't it be helpful to compare the experimental version with the previous one to see which lines have been altered or added? "Program Mis-Matcher" helps out in situations like these. It compares two BASIC programs saved on disk.

## **Comparing**

To use the program, type it and save it on disk. When you run the program, it asks for the names of two files. Type in the program names, pressing RETURN after each one. Mis-Matcher will stop if it can't find the files on disk. The program then asks whether you want a listing of the differences to go to the printer. Press N to send the output to the screen only. The program assumes that the printer is connected as device 4. You'll need to change line 80 if you use some other configuration.

Commodore 128 owners can use the 64 version. The program must be run in Commodore 64 mode, but it can be used to compare both BASIC 2.0 (Commodore 64) and BASIC 7.0 (Commodore 128) programs. One special restriction applies to comparing 128 programs if you have a 1571 disk drive: Both of the programs being compared must be on the front side of the disk. When used in conjunction with 128 mode, the 1571 can store data on both sides of the disk. Mis-Matcher will fail if either or both of the program files to be tested are on the

## CHAPTER 1

second side. If they are, save both programs on a disk formatted while the drive is in 1541 mode (as when the computer is set for 64 mode). You can then use Mis-Matcher.

There are two ways in which one program may differ from another. First, one might have a line that's missing in the other. Second, the programs might have lines with the same line number which are not identical. Mis-Matcher recognizes and reports both kinds of differences.

Some assemblers, including the popular PAL for the 64, use the BASIC editor to enter source code. For such assemblers, Mis-Matcher can be used to compare two source-code files, so machine language programmers can also benefit from this tool.

### 64/128 Program Mis-Matcher

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
KJ 10 SA=49152:CS=10600:FR=251:PRINT"{CLR}{WHT}"
SM 20 FORT=SATOSA+65:READA:X=X+A:POKET,A:NEXT
JH 30 IFX<>CSTHENPRINT"ERROR IN DATA STATEMENTS.":
    STOP
GG 40 PRINTCHR$(14)
CA 50 INPUT"NAME OF FIRST PROGRAM ";F1$:INPUT"NAME
    OF SECOND PROGRAM";F2$
RB 60 PRINT"{CLR}OUTPUT TO PRINTER?{2 SPACES}Y/N"
EQ 70 GET SP$:IFSP$="" THEN70
GX 80 IFSP$="Y"THEN OPEN 4,4,7:PRINT#4,"COMPARING
    {SPACE}"F1$" TO "F2$".":PRINT#4
RH 90 J$="{20 SPACES}{20 LEFT}"
AK 100 OPEN 2,8,2,F1$:OPEN 3,8,3,F2$:GOSUB650
EG 110 FORT=1TO2:GET#2,A$:GET#3,B$:NEXT:REM
    {2 SPACES}THROW AWAY FIRST 2 BYTES
KB 120 FORT=1TO2:GET#2,A$:GET#3,B$:NEXT:REM
    {2 SPACES}THROW AWAY LINE LINKS
JE 130 A=0:IFA$<>""THENA=ASC(A$)
AA 140 B=0:IFB$<>""THENB=ASC(B$)
EQ 150 IF A=0 AND B=0 THEN300
BX 160 IFA=0THENLA=64000:GOTO180
PG 170 GOSUB480:REM GET LINE NUMBER
DG 180 IFB=0THENLB=64000:GOTO200
AH 190 GOSUB520:REM GET LINE NUMBER
JB 200 IFLA<>LB THEN GOSUB320:REM{2 SPACES}CHECK F
    OR DIFFERENT LINE #
XF 210 IFLA=64000ANDLB=64000THEN300
BA 220 PRINT"CHECKING LINE"LA:PRINT"{UP}";
RF 230 SYS SA
SF 240 IF PEEK(FR+1)=0 THEN120
```

## BASIC PROGRAMMING UTILITIES

```
XP 250 PRINT J$ "LINES"LA"DO NOT AGREE."
DC 260 IF SP$<>"Y" THEN120
AG 270 IF TG=1 THEN PRINT#4
GP 280 TG=0:PRINT#4, "LINES"LA"DO NOT AGREE."
GE 290 GOTO120
RA 300 CLOSE2:CLOSE3:CLOSE4:END
EQ 310 REM --{37 SPACES}EXTRA LINE
JJ 320 IF LA>=LB THEN390
BX 330 PRINT J$"LINE"LA"IN {RVS} "F1$" {OFF} NOT I
    N {RVS} "F2$".
SC 340 IF SP$="Y" AND TG=0 THEN PRINT#4
BK 350 TG=1:IF SP$="Y" THEN PRINT#4, "LINE"LA"IN "F
    1$" NOT IN "F2$".
QA 360 POKE FR,2:SYS SA+55
CF 370 GET#2,A1$:GET#2,A2$:IF A1$="" AND A2$="" TH
    EN LA=64000:GOTO320
PQ 380 GOSUB480:GOTO320:REM GET NEW LINE #
RX 390 IF LB>=LA THEN460
XP 400 PRINT J$"LINE"LB"IN {RVS} "F2$" {OFF} NOT I
    N {RVS} "F1$".
XS 410 IF SP$="Y" AND TG=0 THEN PRINT#4
GD 420 TG=1:IF SP$="Y" THEN PRINT#4, "LINE"LB"IN "F
    2$" NOT IN "F1$".
AP 430 POKE FR,3:SYS SA+55
RH 440 GET#3,B1$:GET#3,B2$:IFB1$=""ANDB2$=""THENLB
    =64000:GOTO320
FX 450 GOSUB520:GOTO320
KD 460 RETURN
MR 470 REM -- GET LINE NUMBERS
AF 480 GET#2,A1$:GET#2,A2$
BQ 490 A1=0:IFA1$<>""THENA1=ASC(A1$)
AJ 500 A2=0:IFA2$<>""THENA2=ASC(A2$)
BD 510 LA=A2*256+A1:RETURN
PJ 520 GET#3,B1$:GET#3,B2$
BG 530 B1=0:IFB1$<>""THENB1=ASC(B1$)
DB 540 B2=0:IFB2$<>""THENB2=ASC(B2$)
XX 550 LB=B2*256+B1:RETURN
QR 560 DATA 169,0,133,252,162,2,32,198
GM 570 DATA 255,160,255,200,32,207,255,240
XP 580 DATA 6,153,66,192,76,11,192,132
SQ 590 DATA 251,162,3,32,198,255,160,255
PB 600 DATA 200,32,207,255,240,10,217,66
QP 610 DATA 192,240,245,230,252,76,32,192
DS 620 DATA 196,251,240,2,230,252,96,166
JF 630 DATA 251,32,198,255,32,207,255,208
BH 640 DATA 251,96
QQ 650 OPEN15,8,15:INPUT#15,A,B$,C,D:IFATHENPRINTA
    ,B$,C,D:STOP
AB 660 RETURN
```

# Instant Keywords

---

*Shawn K. Smith*

*Save time and typing effort with this short utility for the Commodore 128. Up to 52 keywords can be entered—each an easy-to-remember, two-key combination.*

“Instant Keywords” can drastically reduce the time it takes you to type in a program. This utility prints a BASIC 7.0 keyword when the Commodore or SHIFT key is pressed in conjunction with a letter key. For instance, pressing the SHIFT and L keys displays the keyword LOOP. A total of 52 keywords can be displayed in this fashion. Refer to the table below for a list of the key combinations. Also, pressing the SHIFT or Commodore key while in quote mode displays the standard graphics characters rather than a BASIC keyword.

Instant Keywords is short and easy to use. Although it contains mostly machine language (ML), you don't have to know any ML to use it. In fact, you can just type it in and run it as a BASIC program. First, type in the program and then save a copy. When you run it, the BASIC loader stores the ML in an area of RAM which is determined by the value S in line 100 (changing the value of S will relocate the utility). Once the data has been stored in RAM, the utility is activated, the address to deactivate/reactivate it is displayed, and the loader is erased from memory. Pressing RUN/STOP and RESTORE is another way to deactivate the program.

## **Modifying the Program**

You may want to rearrange the utility to support a different set of keywords. This can be accomplished with minor changes to the utility. But, first, a quick background note about keywords is in order. BASIC 7.0 contains 130+ commands or keywords. Most of the keywords (including all of the keywords in the 64's BASIC 2.0) are represented by one-byte tokens. For instance, the command PRINT is stored in the computer with a token value of \$99 (153 decimal). Because the 128 has a larger vocabulary, the designers of the 128 decided

## BASIC PROGRAMMING UTILITIES

to use two-byte tokens to represent some of the new commands. All the new two-byte commands use \$CE or \$FE as the first byte of the token. Instant Keywords will allow you to use any keyword except ones that begin with \$CE as the first token value. (This eliminates the use of only eight keywords.)

### Key Combinations

Letter	SHIFT	Commodore
A	SLEEP	STR\$
B	BEGIN	BEND
C	CHR\$	COLOR
D	DOPEN	DCLOSE
E	ELSE	ENVELOPE
F	FOR	FILTER
G	GOTO	GOSUB
H	HEX\$	DEC
I	INPUT	INSTR
J	JOY	PLAY
K	DRAW	CHAR
L	LOOP	LOCATE
M	MID\$	MOVSPR
N	NEXT	COLLISION
O	TAB(	SPC(
P	PRINT	PAINT
Q	GSHAPE	SSHAPE
R	RETURN	RESTORE
S	SPRITE	SOUND
T	THEN	TEMPO
U	USING	UNTIL
V	READ	DATA
W	WHILE	WINDOW
X	POKE	PEEK
Y	GRAPHIC	CIRCLE
Z	LEFT\$	RIGHT\$

The last 52 hexadecimal values in the loader (beginning with 0B in line 200) are the token values of the keywords displayed by Instant Keywords. The first 26 hex values are for the SHIFT key. (The token for SHIFT-A is the first, and the token for SHIFT-Z is the twenty-sixth.) The last 26 values are for the Commodore key. If you plan to add tokens for any

## CHAPTER 1

two-byte commands, leave off the first byte (\$FE)—the program knows that it is a two-byte command and will adjust itself accordingly. If you're unsure of the token value of any keyword, type in this program:

```
GA 10 GOTO30
EF 20 REM ** PLACE KEYWORD HERE **
HS 30 BANK15:B=PEEK(45)+12+PEEK(46)*256:PRINT "KEY
      WORD VALUE(S) = ";
BD 40 H=PEEK(B):PRINTRIGHT$(HEX$(H),2);" ";
AE 50 IFH=254ORH=206THENB=B+1:GOTO40
```

In line 20, type the keyword for which you want to find the token value. Run the program, and it will display the token value for the keyword you've inserted.

### Instant Keywords

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
AE 100 PRINT"{CLR}":S=DEC("1300"):BANK15:FORD=STOS
      +176:READ A$:POKE D,DEC(A$):NEXT
HB 110 M=S+22:V%=M/256:V=M-256*V%:POKE S+12,V%:POK
      E S+14,V:M=D-52:V%=M/256:V=M-256*V%
FM 120 POKE S+65,V%:POKE S+64,V:SYS(S):PRINT"
      {2 DOWN}{RVS}{WHT}INSTANT KEYWORDS ACTIVATE
      DI
RK 130 PRINT"{DOWN}{RVS}TO DE/ACTIVATE SYS"S:NEW
JG 140 DATA A9,C8,CD,37,03,F0,04,A2,05,D0,04,A9,13
      ,A2,16,8D,37,03,8E,36
KE 150 DATA 03,60,EA,48,A6,F4,F0,04,68,4C,05,C8,A4
      ,D4,AD,3E,03,85,24,AD
AR 160 DATA 3F,03,85,25,B1,24,EA,38,E9,41
QA 170 DATA C9,1A,B0,E6,A2,02,E4,D3,D0,02,69,19,A8
      ,B9,7D,13,C9,27,B0,09
EE 180 DATA 69,7E,AA,A0,09,A9,46,D0,05,AA,A0,17,A9
      ,44,84,24,85,25,A0,00
MK 190 DATA CA,10,0F,B1,24,48,E6,24,D0,02,E6,25,68
      ,10,F4,30,EF,C8,B1,24
BH 200 DATA 30,05,20,0C,56,D0,F6,29,7F,20,0C,56,68
      ,60,EA,0B,18,C7,0D,D5
KA 210 DATA 81,89,D2,85,CF,E5,EC,CA,82,A3,99,E3,8E
      ,07,A7,FB,87,FD,97,DE
QD 220 DATA C8,C4,19,E7,0F,0A,03,8D,D1,D4,04,E0,E6
      ,06,17,A6,DF,E4,8C,DA
KQ 230 DATA 05,FC,83,1A,C2,E2,C9
```



# Line Count

---

*Kenneth J. Rogerson*

*A boon to programmers, "Line Count" helps increase the speed of BASIC programs by reporting which lines are executed most often so that your efforts may be concentrated where they will be most effective. Although it's written in machine language, you don't need to know any machine language to use it. For the Commodore 64 and 128.*

"Line Count" is a program I had waited a long time for someone else to write. I finally learned enough machine language to write it myself, however, and to make it as useful and efficient as possible. Line Count is a utility that reports which lines are executed most often in a BASIC program. With this information, you can target inefficient or redundant areas of code that slow down the program.

## Typing It In

Type in the version appropriate for your computer: Program 1 for the 64, Program 2 for the 128. Each of the programs has a checksum feature in line 20 that will detect typing errors in the DATA section. It's also recommended that you use "The Automatic Proofreader," found in Appendix B.

After you've finished typing in the program, save a copy to disk or tape. To use it, load it and type RUN. If you've entered the program correctly, the machine language will be POKEd into memory (locations 49152-49663 on the 64 and 3072-3517 on the 128).

After you've loaded and run Line Count, load the BASIC program you want Line Count to check. Before running your program, enter **SYS 49155** (64 version) or **SYS 3075** (128 version) in direct mode, and press RETURN. This activates Line Count. In a moment the READY prompt and cursor will reappear. Now run your BASIC program. When it has finished running—or if it stops for any reason—you can then get a report from Line Count. To do this, enter in direct mode **SYS 49152** (64 version) or **SYS 3072** (128 version), and press RETURN. Line Count will list the line numbers that your BASIC

# CHAPTER 1

---

program executed, followed by a number indicating how many times each line was executed.

Since each section of Line Count reinitializes itself when activated, you can check several programs in succession without having to rerun Line Count. Simply load and run each program to be checked, remembering to activate Line Count with the appropriate SYS and to generate the report with the other SYS listed above.

## Features and Limitations

Because it adds its own routine to the normal functions of BASIC, Line Count causes your program to run more slowly than usual. Line Count cannot keep track of more than 896 lines. If your program is longer, Line Count will record only the *first* 896 lines. And Line Count can count up to 65535. If a line is executed more times than that, Line Count will stop your program with an error message. Also note that Line Count may not work with other cartridges or utilities.

The program will not work reliably if either of the SYS commands (used to activate or generate a report) is used within a program. Use Line Count's SYS commands only in direct mode.

How your program is numbered or how many commands are on each of your program lines is not important.

If a program runs *without* Line Count resident, it will run *with* Line Count, unless your BASIC program uses any of the following locations:

64 Version	128 Version
49152-53247	3072-3517
251-254	251-254
776-777	776-777

If your BASIC program contains DATA statements, how they are counted will depend on where they are placed. DATA statements at the end of your program, which are not referenced by a GOTO or GOSUB and are preceded by END or STOP, will not be counted, no matter how often the DATA is read. If your program contains DATA statements placed in the normal flow of the program, and therefore scanned by the BASIC interpreter, they will be counted once each time they are scanned no matter how often the DATA is read.

## BASIC PROGRAMMING UTILITIES

---

You may notice strange results when a FOR-NEXT loop is split between two or more lines. The explanation is simple. Assume that the program below is being checked and that there are no other lines in the program. Because there are no commands on line 100 after the FOR-TO portion of the loop, this line will be counted once, while lines 110 and 120 will be counted 100 times. This will occur in your programs in similar circumstances. If there is a command or statement on the same line after the FOR-TO command, the line will be counted the number of times that the loop itself is executed—in this case, 100 times.

```
100 FOR X=1 TO 100
110 REM
120 NEXT X
```

### Program 1. Line Count—64 Version

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
MS 10 FORA=49152TO49539:READB:POKEA,B:C=C+B:NEXT
PH 20 IFC<>51719THENPRINT"{CLR}DATA ERROR":STOP
MJ 30 PRINT "{CLR}{2 DOWN}{WHT}SYS 49155: TO INITI
ALIZE":PRINT "{DOWN}SYS 49152: FOR A REPORT"
MM 40 DATA 76,13,193,32,41,192,162,14,160,0,169,25
5,145,251,136,208,251,230,252
MS 50 DATA 202,208,246,169,50,141,8,3,169,192,141,
9,3,169,255,141,129,193,141
JC 60 DATA 130,193,96,169,0,133,251,169,194,133,25
2,96,32,41,192,165,58,201,255
PP 70 DATA 240,98,205,130,193,208,7,165,57,205,129
,193,240,86,160,1,185,57,0
MB 80 DATA 153,129,193,136,16,247,160,0,177,251,20
0,49,251,201,255,208,17,165,58
KF 90 DATA 145,251,136,165,57,145,251,32,160,192,3
2,160,192,208,47,160,0,177,251
GP 100 DATA 217,57,0,208,32,200,192,2,208,244,32,1
60,192,208,28,32,88,193,169
JQ 110 DATA 99,160,193,32,30,171,166,57,165,58,32,
205,189,162,128,108,0,3,32
EK 120 DATA 64,193,76,82,192,76,228,167,160,2,177,
251,24,105,1,145,251,208,7
XH 130 DATA 200,177,251,105,0,145,251,96,169,0,141
,131,193,32,41,192,160,0,177
CP 140 DATA 251,200,49,251,201,255,240,65,160,1,17
7,251,153,129,193,136,16,248,160
HD 150 DATA 4,177,251,56,237,129,193,200,177,251,2
37,130,193,176,33,169,1,141,131
AF 160 DATA 193,160,7,177,251,72,136,16,250,160,4,
104,145,251,200,192,8,208,248
```

# CHAPTER 1

---

KA 170 DATA 160,0,104,145,251,200,192,4,208,248,32  
,64,193,76,187,192,173,131,193  
RA 180 DATA 208,167,96,32,179,192,32,41,192,169,13  
,32,210,255,160,0,177,251,170  
JA 190 DATA 200,49,251,201,255,240,25,177,251,32,7  
8,193,169,58,32,210,255,200,177  
FG 200 DATA 251,170,200,177,251,32,78,193,32,64,19  
3,208,214,76,88,193,165,251,24  
AX 210 DATA 105,4,133,251,165,252,105,0,133,252,96  
,140,131,193,32,205,189,172,131  
BB 220 DATA 193,96,169,228,141,8,3,169,167,141,9,3  
,96,13,84,79,79,32,77  
EM 230 DATA 65,78,89,32,69,88,69,67,85,84,73,79,78  
,83,32,79,70,32,76  
AP 240 DATA 73,78,69,32,0,0,0,0

## Program 2. Line Count—128 Version

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

KJ 10 FORA=3072TO3517:READB:POKEA,B:C=C+B:NEXT  
BH 20 IFC<>53196THENPRINT"{CLR}DATA ERROR":STOP  
CS 30 PRINT "{CLR}{WHT}{2 DOWN}SYS 3075: TO INITIA  
LIZE":PRINT "{DOWN}SYS 3072: FOR A REPORT"  
FG 40 DATA 76,63,13,169,48,141,0,255,32,56,12,162,  
14,160,0,169,255,145,251  
RS 50 DATA 136,208,251,230,252,202,208,246,169,65,  
141,8,3,169,12,141,9,3,169  
FG 60 DATA 255,141,187,13,141,188,13,169,240,141,1  
9,18,169,0,141,0,255,96,169  
SP 70 DATA 0,133,251,169,240,133,252,96,169,48,141  
,0,255,32,56,12,165,60,201  
CQ 80 DATA 255,240,10,205,188,13,208,7,165,59,205,  
187,13,240,116,160,1,185,59  
PK 90 DATA 0,153,187,13,136,16,247,160,0,177,251,2  
00,49,251,201,255,208,17,165  
QR 100 DATA 60,145,251,136,165,59,145,251,32,210,1  
2,32,210,12,208,77,160,0,177  
QD 110 DATA 251,217,59,0,208,62,200,192,2,208,244,  
32,210,12,208,58,32,155,13  
GB 120 DATA 169,159,160,12,32,125,255,13,84,79,79,  
32,77,65,78,89,32,69,88  
ES 130 DATA 69,67,85,84,73,79,78,83,32,79,70,32,76  
,73,78,69,32,0,166  
JX 140 DATA 59,165,60,32,133,13,162,128,108,0,3,32  
,119,13,76,102,12,76,162  
SK 150 DATA 74,160,2,177,251,24,105,1,145,251,208,  
7,200,177,251,105,0,145,251  
XP 160 DATA 96,169,0,141,189,13,32,56,12,160,0,177  
,251,200,49,251,201,255,240

## BASIC PROGRAMMING UTILITIES

---

---

SX 170 DATA 65,160,1,177,251,153,187,13,136,16,248  
,160,4,177,251,56,237,187,13

FK 180 DATA 200,177,251,237,188,13,176,33,169,1,14  
1,189,13,160,7,177,251,72,136

RA 190 DATA 16,250,160,4,104,145,251,200,192,8,208  
,248,160,0,104,145,251,200,192

RF 200 DATA 4,208,248,32,119,13,76,237,12,173,189,  
13,208,167,96,169,48,141,0

SF 210 DATA 255,32,229,12,32,56,12,169,13,32,171,1  
3,160,0,177,251,170,200,49

PF 220 DATA 251,201,255,240,25,177,251,32,133,13,1  
69,58,32,171,13,200,177,251,170

PQ 230 DATA 200,177,251,32,133,13,32,119,13,208,21  
4,76,155,13,165,251,24,105,4

CB 240 DATA 133,251,165,252,105,0,133,252,96,140,1  
89,13,72,169,0,141,0,255,104

BD 250 DATA 32,50,142,169,48,141,0,255,172,189,13,  
96,169,162,141,8,3,169,74

CE 260 DATA 141,9,3,169,0,141,0,255,96,72,169,0,14  
1,0,255,104,32,210,255

FB 270 DATA 169,48,141,0,255,96,0,0,0

# Sound Off

---

*John Martin*

*The Commodore 64 is usually accommodating when it comes to alerting you to errors. But when you're working with hi-res screens or custom characters, error messages can be invisible or unreadable. This short machine language program is the answer. For the Commodore 64 and 128 in 64 mode.*

We've all seen error messages on our screens. As unwelcome as they may be, they're helpful in pointing to the type of problem or mistake in the program. In some cases, however, an error message will not appear—for example, when a high-resolution screen is displayed. Or, if you've redefined the character set, the error message will print, but won't be readable. Any number of hidden errors can cause a program to stop during setup—with no error message.

"Sound Off" provides a solution. It's a short machine language routine that creates a bell-like tone and prints an error message when an error is encountered. It works in both program and direct mode. And it is not disabled by the RUN/STOP-RESTORE key combination. The only way to disable it is to reset the computer by typing SYS 64738 or by turning the computer off and then on again. Each of these methods erases the program currently in memory, so be sure to save the program before resetting the computer.

**Note to 128 users:** To simulate the effects of Sound Off in 128 mode, use the TRAP statement to enable an error-handling routine. Within that routine, you could use GRAPHIC0 to turn off the hi-res screen and SOUND to create a beep. A machine language routine such as Sound Off provides is not necessary on the 128; you can trap errors with BASIC alone.

## Using the Program

After you have typed in the program, be sure to save a copy. To use Sound Off, just load it and type RUN. The machine language data is POKEd into a safe area of memory (53047-53171), and the program is activated. Note that any programs

## BASIC PROGRAMMING UTILITIES

---

or cartridges that use memory addresses 53047–53171 will disable Sound Off. However, the 64 Super Expander cartridge will not interfere.

### Sound Off

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
GK 10 FORA=53047TO53171:READB:C=C+B:POKEA,B:NEXT
QE 20 IFC<>13412THENPRINT"DATA ERROR":STOP
FR 30 PRINT"{CLR}{2 DOWN}SYS 53161 TO ACTIVATE"
CK 40 DATA 169,0,162,212,141,253,0,142,254,0,160,0
    ,169,0,145,253
GR 50 DATA 152,201,23,240,7,24,105,1,168,76,67,207
    ,169,15,141,24
DH 60 DATA 212,169,0,141,5,212,169,247,141,6,212,1
    69,17,141,4,212
KE 70 DATA 169,40,141,1,212,169,0,141,0,212,169,0,
    141,160,0,141
CK 80 DATA 161,0,141,162,0,173,162,0,201,10,240,3,
    76,124,207,169
SX 90 DATA 16,141,4,212,96,141,60,3,142,61,3,140,6
    2,3,138,201,31,176,3,32,55
RJ 100 DATA 207,173,60,3,174,61,3,172,62,3,76,139,
    227,169,140,141
FE 110 DATA 0,3,169,207,141,1,3,96
```

# BASIC Line Extender

---

*Jay A. Reeve*

*This short but powerful utility gracefully removes Commodore's "two screen lines only" programming restriction. For the Commodore 64 and 128 in 64 mode.*

How often have you hit the two-line squeeze while programming? Your BASIC statement won't quite fit into what's left of the two lines allowed by the Commodore screen editor. You go back and change full keywords to abbreviations—it still doesn't fit. You could simply move the last statement to the next line, except it follows an IF-THEN statement.

Next, you replace the statement with GOTO and guess at a line number that you hope won't be in the way later on. Then you retype the statement on the new line (how did that go again?) and try to recall the logic you were using for the program. You're left with a feeling that your program has just lost a little of its elegance.

## **Ending the Squeeze**

"BASIC Line Extender" is a utility for BASIC programmers that allows you to add to the end of any line in your program. You're still limited to entering two screen lines at a time, but you can build up program lines which take six or more screen lines to list. Once entered, these extended lines will save, load, list, and run properly whether or not BASIC Line Extender is active.

Although BASIC Line Extender can be used to extend any program line, it's most valuable for continuing IF-THEN statements, allowing you to handle an error, for instance, on the same line in which it is detected. It's also good for adding descriptive REM statements to program lines and for creating extra long DATA lines or ON-GOTO statements. When memory is at a premium, an extension saves at least four bytes over starting a new line.



## BASIC PROGRAMMING UTILITIES

---

### Preparing BASIC Line Extender

The program itself is short and contains some machine language in the form of DATA statements. Type it in and be sure to save a copy before running it—the program will erase itself after creating and activating the machine language routine.

Line 10 protects a space for the machine language program by lowering the pointer to the top-of-BASIC by 256 bytes. If you'd like to locate BASIC Line Extender elsewhere in memory, replace the value for the variable AD in line 10 with the starting address you want.

To disable BASIC Line Extender, enter

```
POKE772,124:POKE773,165
```

### Creating Extensions

An extension is a line to be added to the end of a program line already in memory. Enter the extension with the *same line number* as the line to which it will be added. The first character after the line number will be one of three signal characters which never begin a normal BASIC line: a colon (:), a quotation mark ("), or a British pound sign (£).

1. When the extension begins with a new statement, follow the line number with a colon. The colon will be included in the extended line as a statement divider. If you enter the line 10 PRINT "THIS IS A LINE", it will list normally. Next, enter the line 10 :PRINT "AND ITS EXTENSION", and the listing will read 10 PRINT "THIS IS A LINE":PRINT "AND ITS EXTENSION".
2. When the separation falls between quotation marks (in a PRINT statement or string definition), the original line must *not* have a closing quotation mark. (BASIC does not require end quotes at the end of a line, anyway.) The extension line, however, must begin with a quotation mark ("), which will be omitted from the extended line. Enter 20 PRINT "THIS IS A LINE followed by 20 " AND ITS EXTENSION", and the listing will show 20 PRINT "THIS IS A LINE AND ITS EXTENSION".
3. To continue in midstatement, begin the extension line with the British pound sign. This signal character, of course, will not be included in the program. Enter 15 A=A+B, then 15 £\*256, to get 15 A=A+B\*256. A statement may be broken and continued at any point *except within a BASIC keyword*.

(BASIC stores and uses keywords as single-byte tokens. Extensions are tokenized separately, so the tokenizing routine would have no opportunity to recognize a keyword that was partly in the original line and partly in the extension.)

Watch carefully for split keywords—they will appear to be correct in listings, but will cause syntax errors or, worse, erroneous results when the program is run. Numbers and variable names may be broken: 25 SYS49 and 25 £152 will result in 25 SYS49152.

If you prefer to use a different signal character (other than £), try replacing the number 92 at the end of line 60 with 170 (for +), 95 for (-), or 173 for (/).

You can, if you like, add extension after extension—the tokenized line may be up to 244 bytes long. If an extension exceeds this limit, a STRING TOO LONG error message will be returned, leaving the line unchanged. The trade-off for this dramatic increase in flexibility is some difficulty in editing.

### **Editing Extended Lines**

If you ever use abbreviations for BASIC keywords, you know that line listings sometimes run over onto a third line, and that run-over lines can be difficult to edit. The extra long lines generated with BASIC Line Extender may be no easier to edit, but in most cases they'll be no more difficult either. List the extended line and reenter it, two screen lines at a time, by using abbreviations to shrink the lines enough to insert the line number and the appropriate signal character. Be careful to reassemble any split keywords before entering and making any necessary changes as you go.

If you anticipate the need to edit extended lines, you may wish to enter them first as partial lines with different, very high line numbers (you can even include the signal characters, as long as a line of that number does not already exist), then combine them by changing the line numbers. To edit, you can list the partial lines, make any changes, and recombine them by again replacing the line numbers. When the program is completed, the partial lines can easily be deleted from the end.

You no longer need to count spaces, refer to lists of abbreviations, and retype statements that won't fit. Activate BASIC Line Extender at the start of each programming session and let your creative logic flow—and forget about the two-line squeeze.

## BASIC PROGRAMMING UTILITIES

---

### BASIC Line Extender

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
AQ 10 POKE55,0:POKE56,PEEK(56)-1:CLR:AD=PEEK(56)*2
56:PRINT"{CLR}{DOWN}WORKING...":C=0
KQ 20 FORI=ADTOAD+129:READB:C=C+B:POKEI,B:NEXT:IFC
<>13921THENPRINT"DATA ERR":STOP
QF 30 POKEAD+4,PEEK(772):POKEAD+5,PEEK(773)
QP 40 POKE773,AD/256:POKE772,AD-256*PEEK(773)
PA 50 PRINT"{CLR}{DOWN}"CHR$(34)"BASIC LINE EXTEN
DER"CHR$(34)" IS OPERATIONAL.":NEW
AG 60 DATA138,240,3,76,124,165,32,124,165,173,0,2,
201,58,240,9,201,92
HF 70 DATA240,5,201,34,240,1,96,132,11,32,19,166,1
04,104,176,3,76,237,164
CB 80 DATA162,2,134,39,202,173,0,2,201,58,208,3,23
0,11,202,134,38,160,1,177
JD 90 DATA95,170,136,177,95,208,1,202,134,96,170,2
02,229,95,24,101,11,144
HG 100 DATA5,162,23,108,0,3,134,95,165,45,133,90,1
65,11,233,6,133,11,101,90
JD 110 DATA133,88,164,46,132,91,144,1,200,132,89,3
2,184,163,165,49,164,50
FM 120 DATA133,45,132,46,164,11,177,38,145,95,136,
16,249,76,42,165,0
```

# Searchlight

---

*Katherine Myers*

*You can save time with this short machine language routine that spotlights errors in BASIC programs. A good tool for debugging programs you've written or those you've typed in. For the Commodore 64 and 128 in 64 mode.*

Your program comes to an abrupt halt and the words SYNTAX ERROR IN 50 are on the screen. You list line 50 and study it time and again, wondering where the problem lies. Why, with all the amazing speed and power of your computer, won't it show you where your error is?

I know I've spent many hours—especially when I first got my computer—staring at lines I thought were written correctly and wondering where that error was. If you've spent too much time searching for errors, this program is for you.

"Searchlight" is written in machine language (in the form of a BASIC loader) and is easy to use. Type in the program, save a copy, and run it before you load or start typing the program you'll be working on.

## **A Safe Section of Memory**

Searchlight first asks you for the starting address so that you can locate it where it won't interfere with your program or with any other machine language utilities in memory. Since it's only 121 bytes long, it will fit in the cassette buffer. Of course, if you're using tape, this location would be inappropriate.

It's important to put the program somewhere safe in memory where BASIC can't interfere with it. Here are some suggestions for a starting address: Disk users can use 828 (the beginning of the cassette buffer) or the 4K section at 49152–53247. (Remember to allow for 121 bytes.) Also, 50000 would work, and it's easy to remember.

You can also create a protected area of memory. Before running Searchlight, follow these instructions:

- Enter

```
TM=PEEK(55)+256*PEEK(56):SA=TM-121:PRINTSA
```

## BASIC PROGRAMMING UTILITIES

---

- and write down the number that will be the starting address.
- Next, move the top of memory down with

**HB=INT(SA/256):LB=SA-256\*HB:POKE55,LB:POKE56,HB:CLR**

Line 20 is written for the user's protection—it prevents the program from being inappropriately located (to screen RAM, for example). It also won't let you locate it in an address below 820 or in ROM.

After a location has been chosen, the loader POKES in the data for the machine language routine and erases itself with the NEW command if all the data entered is correct. Be sure to note the location you first selected as you'll have to SYS to it.

### Putting It into Action

When you wish to use Searchlight, SYS to the address you've chosen before you run your program. I usually put the SYS command in the first line of the program I'm testing so that I don't have to type it in each time. When you run a program and an error is encountered, type LIST. The line that contains the error will be listed with an arrow pointing to the appropriate place in the code. For example, it may be in a spot where some code is missing, to a place where you're trying to go to a nonexistent line number, and so on. If the arrow is placed in a statement that contains parentheses, check to see whether there are an equal number of open and closed parentheses. If there isn't an arrow in the statement listed, the error is at the end of the line. After Searchlight lists an erroneous statement and it is corrected, run the program again. If another error exists, this, too, will be listed. Keep running it until the program is free of errors.

Anytime you use RUN/STOP-RESTORE, Searchlight will be deactivated. This feature has been programmed for user convenience—you don't have to turn off the machine or remember any POKES. Also, you can use Searchlight as a TRACE function by using the RUN/STOP key. Pressing RUN/STOP when a program is running will cause BREAK IN <line number> to appear. Now, when you type LIST, the arrow will point to the statement that was executing when you pressed the RUN/STOP key. This can be useful in testing and debugging.

When you make a correction, be sure to delete the arrow before pressing RETURN. If you don't, the arrow will become part of the BASIC line in which it appears.

## CHAPTER 1

Certain errors, like those encountered during an INPUT statement (EXTRA IGNORED and REDO FROM START, for example) will not activate Searchlight since these are user errors and are not problems related to a program line. Also, the program will not indicate errors encountered during statements entered in direct mode.

This program can save you many hours of searching for errors—especially when you're dealing with long lines. You can use it for your own programs or for those you type in from books or magazines. It's especially helpful when you're typing in a program since it points to the place in the line where you need to look. You won't have to start from the beginning of that line when you compare your version with the printed copy.

### Searchlight

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
BF 10 INPUT "{CLR}STARTING ADDRESS";SA$:SA=VAL(SA$)
XC 20 IFSA<820OR(SA>9000ANDSA<2048)OR(SA>32643ANDSA
    <49152)ORSA>53122THEN10
MP 30 FORI=SATOSA+121:READA:POKEI,A:N=N+A:NEXT
EP 40 IFN<>16667THENPRINT"{CLR}ERROR IN DATA":STOP
SG 50 A=INT((SA+11)/256):B=SA+11-A*256:C=INT((SA+4
    6)/256):D=SA+46-C*256
EA 60 POKESA+1,B:POKESA+6,A:POKESA+34,D:POKESA+39,
    C
GG 70 PRINT"SYS"SA"TO USE":NEW
EA 80 DATA169,11,141,34,3,169,192,141,35,3,96,165
SE 90 DATA123,201,2,240,26,165,122,133,251,165,57,
    133
XB 100 DATA181,165,58,133,182,169,0,133,183,169,46
    ,141
CX 110 DATA6,3,169,192,141,7,3,76,51,243,133,252
XE 120 DATA165,183,208,18,230,183,169,145,32,210,2
    55,165
EK 130 DATA181,133,20,165,182,133,21,76,167,166,16
    5,183
MA 140 DATA201,1,208,9,230,183,56,165,251,229,95,1
    33
XK 150 DATA253,132,254,196,253,240,13,200,200,177,
    95,240
EF 160 DATA9,164,254,165,252,76,26,167,198,254,169
    ,95
PH 170 DATA133,252,169,26,141,6,3,169,167,141,7,3,
    208,231
```

# Triple 64

---

---

*Feeman Ng*

*Three computers in your Commodore 64? This seven-line program creates three independent 12K blocks which can be accessed very simply. An excellent tool for program development and comparison. For the Commodore 64 and 128 in 64 mode.*

Have you ever wished you could work on two or three programs at once and compare them? Or view a disk directory without erasing a program in memory? This short machine language program lets you do just that.

“Triple 64” is a machine language program (in the form of a BASIC loader) that divides the 64’s memory into three independent 12K workspaces. You can work in any one of the areas without disturbing the other two. You can even save and load from any of the three work areas without affecting the others. The program starts at 40004 (\$9C44) and uses only 71 bytes. Also, a favorite area of many machine language programmers, 49152 (\$C000), is unaffected.

## **Accessing Three Computers**

After you’ve entered and saved Triple 64, type RUN. To access any of the three areas, enter SYS 40004. Notice that the cursor disappears immediately after you press RETURN. Now press 1, 2, or 3—the identification numbers of the three independent work areas—and you’re ready to begin programming. If you’ve found that you don’t recall which area you’re in, enter PRINT PEEK(40061). This will return a 1, 2, or 3.

## **Techniques and Applications**

The most obvious use of Triple 64 is to partition the computer to hold three BASIC programs. These could be games, utilities, or applications—or any combination. And switching between them involves only a SYS and a single keypress. Each work area holds up to 12K, space enough for a fairly sophisticated program.

## CHAPTER 1

---

Triple 64 may prove even more useful, however, in the development of your own programs. The three workspaces are truly separate; for example, one could hold a working version of your program, another might contain a test version you're enhancing, and the third could provide a scratch-pad area, where you could try out new ideas and write short programs to test them. These testing routines could even examine the other two memory areas for the effects on the programs residing there. When you've got something working well, you can transfer it to another area with this simple procedure:

1. List it to the screen.
2. Select the desired Triple 64 workspace.
3. Cursor up to the lines you want to transfer, and press RETURN over each of them. They'll immediately be inserted into the BASIC program in the new workspace.

Triple 64 offers a wide range of possibilities—it's almost like having three instant 12K disk drives at your disposal. And if you have a disk drive as well, you can maintain its directory in one workspace while you work in the others. This is a very useful feature if your programs will be using files on the disk currently in your drive.

### Triple 64

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
MX 10 FORY=40004TO40071:READA:POKEY,A:NEXT
RD 20 FORY=14336TO14338:POKEY,0:NEXT
SQ 30 FORY=26624TO26626:POKEY,0:NEXT:NEW
PX 40 DATA174,125,156,165,45,157,129,156,165,46,15
    7,132,156,32,228,255,41,15,240
ES 50 DATA249,201,4,176,245,170,142,125,156,189,12
    5,156,133,44,189,126,156,133,56
CH 60 DATA189,129,156,133,45,133,47,133,49,189,132
    ,156,133,46,133,48,133,50,96,1
BE 70 DATA8,56,104,152,3,3,3,8,56,104
```



# String Search

---

---

*Glen Colbert*

*Use "String Search" to search through string arrays looking for a match. This utility is much faster than its BASIC equivalent. For the Commodore 64 and 128 in 64 mode.*

Although machine language is fast, BASIC is generally preferable when you're writing a program to handle lots of strings: names, addresses, recipes, lists in general. BASIC has built-in string and array functions that make it easy to handle large volumes of information.

It's frustrating, however, to have to wait while the program searches through a few hundred entries looking for a match. The longer the list, the slower BASIC becomes.

String Search is fast because it's written in machine language (ML), although you don't have to understand machine language to use it.

## **Special Instructions**

There are two things you must do before using the program:

1. The first and second variables defined in the target program must be strings. To be safe, put them in the first few lines. And the second string must be the "match" you're looking for.
2. The string array to be searched must be the first array DIMensioned. An integer array containing the same number of elements must be the second array DIMensioned. The integer array will contain flags that indicate a match has been found.

Program 1 is the BASIC loader for String Search. When you run it, the ML routine is located to the top-of-BASIC memory and the pointers are reset, protecting it from BASIC. Program 1 can be incorporated into your own programs or loaded and run as a separate program before you load your own data-management program.

## CHAPTER 1

---

---

To access the search routine, enter

**SYS (PEEK(55)+256\*PEEK(56))**

The ML routine is relocatable. If you prefer, you can put it up at \$C000 (49152).

Program 2 is a test of String Search. After you have entered, saved, and run Program 1, run Program 2. First, an array containing 300 elements is set up. BASIC then searches for a match, and you see how many jiffies it takes. (A jiffy is 1/60 second.) Next, the ML routine is used. You may be surprised at how much faster you get the results.

### How It Works

The search method used in this routine is quite simple. When it's called, the first operation is to swap out a portion of the zero page (\$D9-\$E9) into the cassette buffer. The length of the string to be checked for is put into \$D9, and the address of the string is set into \$DA-\$DB. Next, addresses \$DC-\$DD are set to point to the zero element of the integer array. Addresses \$E0-\$E1 are set to point to the three bytes of string array information (length, low byte of address, and high byte of address) for the zero element of the string array. Things are now in order for the processing loop.

The first step in the processing loop is to increment the pointers for the arrays that are being worked to the next element. For this reason, the zero element is not searched. The information for the string array element being worked is moved to \$E5-\$E7. Then \$E5 is checked for a null (string = " "); if it is null, the zero page information is put back in and returned to BASIC. A counter for the search string (\$E2) and one for the searched string (\$E3) are set to zero, and the search begins.

These counters determine whether or not there is a match. If the search string counter is equal to the length of the search string, there has been a match. If the searched string counter is equal to the length of the searched string, there has been no match. In either case, the routine sets the value in the integer array and returns to the main loop to try the next element of the array.

If the counters do not match, the accumulator is loaded with the first character of the search string. This is compared against each element of the searched string until a match is

## BASIC PROGRAMMING UTILITIES

---

found. Then the second character of the search string is compared against the next character in the searched string, and so on, until the counter equals the length of the search string. If a match is not found, the search string counter is reset (but the searched string counter isn't), and the program loops back.

### Program 1. String Search—BASIC Loader

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
PG 10 REM*****{3 SPACES}BASIC LOADER F
    OR **{5 SPACES}STRING SEARCH{2 SPACES}*****
    *****
AB 100 PRINT"{CLR}{4 DOWN}{3 SPACES}STRING SEARCHE
    R"
CJ 110 PRINT"{2 DOWN}ONE MOMENT PLEASE"
BH 120 TP=PEEK(55)+256*PEEK(56)
SR 130 TP=TP-186:H=INT((TP)/256):L=TP-H*256:POKE55
    ,L:POKE56,H
JH 140 IN=PEEK(55)+256*PEEK(56):FORC=IN TO IN+185:
    READI:POKEC,I:CK=CK+I:NEXT:
RR 150 IFCK<>26449 THEN PRINT"ERROR IN DATA":END
KH 160 REM***** STRING SEARCH DATA**
    *****
PK 180 DATA 160,17,185,216,0,153,60,3,136,208
KH 190 DATA 247,160,9,177,45,133,217,200,177,45
GF 200 DATA 133,218,200,177,45,133,219,24,160,2
BF 210 DATA 177,47,101,47,105,7,133,220,200,177
SK 220 DATA 47,101,48,133,221,160,0,24,165,47
FF 230 DATA 105,7,133,224,165,48,105,0,133,225
RB 240 DATA 169,0,240,12,160,17,185,60,3,153
BA 250 DATA 216,0,136,208,247,96,24,165,224,105
RC 260 DATA 3,133,224,165,225,105,0,133,225,160
HK 270 DATA 0,177,224,153,229,0,200,192,3,208
GD 280 DATA 246,24,165,220,105,2,133,220,165,221
QD 290 DATA 105,0,133,221,165,229,240,202,208,2
BP 300 DATA 240,210,162,0,134,227,134,226,24,165
XK 310 DATA 217,197,226,240,31,24,165,229,197,227
MS 320 DATA 144,37,164,226,177,218,164,227,209,230
DC 330 DATA 208,6,230,227,230,226,208,226,230,227
PQ 340 DATA 169,0,133,226,240,218,160,0,169,1
KE 350 DATA 145,220,200,169,0,145,220,240,197,160
EJ 360 DATA 0,152,145,220,240,242
RX 999 PRINT"DONE":NEW
```

## CHAPTER 1

### Program 2. String Search—Demo Program

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
PK 10 REM STRING SEARCH DEMO{9 SPACES}PROGRAM
EJ 20 A$="DUMMY DATA":REM**MUST BE A STRING**
PP 30 Q$="":REM THIS IS TO BE USED AS THE SEARCH S
    TRING *****
JC 40 DIMA$(300),Q$(300):REM SEARCHED STRING AND F
    LAG ARRAY
ES 45 ML=PEEK(55)+256*PEEK(56):REM START ADDRESS
GB 50 PRINT"{CLR}{2 DOWN}{2 SPACES}STRING SEARCH D
    EMO"
QE 100 PRINT"BUILDING ARRAY"
QG 110 Q$="GOOD"
DX 120 FORL=1TO299
DK 130 :
QQ 140 :A$(L)="ABCDEFGHIJKLMNPOQRSTUVWXYZ"
PM 150 :
BB 160 NEXTL
BP 170 A$(1)="GARBAGE GOOD MORE GARBAGE"
DM 180 A$(10)="GARB GOOD MORE GARB"
SM 185 A$(70)="GOOD GARBAGE"
DB 190 A$(100)="GARBAGE GOOD"
MM 195 A$(250)="GARBAGE GOOD MORE GARBAGE"
XE 200 PRINT"ARRAY FINISHED"
JQ 300 REM*****{8 SPACES}BASIC SEARCH
    {2 SPACES}*****
BH 310 PRINT"BASIC SEARCH":TI$="000000"
JM 320 FORL=1TO299
BJ 330 :FORJ=1TOLEN(A$(L))-LEN(Q$)+1
KB 340 ::IFMID$(A$(L),J,LEN(Q$))=Q$THENQ$(L)=1:NEX
    TL
HD 350 :NEXTJ
XS 360 NEXTL
MK 370 PRINTTI;"JIFFIES"
AS 380 FORL=1TO299
RG 390 :IFQ$(L)<>0THENPRINTA$(L)
AB 395 NEXTL
SJ 400 REM*****{10 SPACES}ML SEARCH
    {3 SPACES}*****
DD 410 PRINT"ML SEARCH":TI$="000000"
CJ 420 SYS(ML)
XS 430 PRINTTI;"JIFFIES"
QC 440 FORL=1TO299
JF 450 :IFQ$(L)<>0THENPRINTA$(L)
FE 460 NEXTL
GG 999 END
```

# Step Lister

---

---

*E. A. Cottrell*

*“Step Lister” lets you look at your BASIC program lines without repeatedly typing LIST. This is a machine language routine, but it requires no special knowledge to use it. For Commodore 64 and 128 in 64 mode.*

“Step Lister” is a machine language *wedge* (explained below) that allows you to step through a BASIC listing one line at a time. To see the first line of your program, just enter

@0

Entering any other number after the *at* sign (@) will start the listing at that line. There should be no spaces between the @ and the line number, and the @ must be on the left margin.

Then press any key, and the next line will be displayed. Press the space bar and hold it down; the listing will continue scrolling until the space bar is released. If you wish to stop Step Lister, press RUN/STOP.

Be sure to save the program before you run it, because, if there are any undetected errors, the computer may crash.

## **What Is a Wedge?**

To understand a wedge, you must first have some knowledge of how BASIC works. When you press RETURN, one of two things happens. If the entered line has a number as the first character, the computer assumes that a BASIC line is being entered. This line is then converted to BASIC *tokens* and is put into its proper place in memory. (Tokens are single-byte symbols that represent BASIC commands. To save space and time, the computer stores PRINT, for example, as 153.)

No interpretation of the characters following the line number is made until the program is run. If the first character is not numeric, the line is tokenized and placed in the BASIC input buffer at locations 512–600 (\$0200–\$0258). The interpreter then calls the CHRGET subroutine to get the characters from the buffer and return them for interpretation.

## CHAPTER 1

---

To implement a wedge, the CHRGET subroutine located at 115-138 (\$73-\$8A) must be altered to go to your machine language program before returning to the interpreter. At the entry point of the wedge, a check is made to see whether the special character (in this case, @) has been entered. If it has been, the special routine is executed. Otherwise, the character is sent to the interpreter for normal BASIC interpretation and execution.

### Using ROM Routines

Step Lister uses many of the subroutines that are part of the BASIC ROM in the 64. Analyzing some of the subroutines already in the machine can prove useful.

The wedge can be a powerful tool. If you decide to write a wedge program of your own, heed one word of caution: Do not try to alter the CHRGET subroutine with BASIC. You will be changing the way BASIC gets its instructions in the middle of a BASIC program, and this will crash your computer.

### Step Lister

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
AB 10 TM=49152
EH 20 FOR I = TM TO TM + 241
MJ 30 READ A: POKE I,A: CHK = CHK + A: NEXT I
AB 40 X = 828: FOR I = X TO X + 23
EM 50 READ A: POKE I,A: CHK = CHK + A: NEXT I
AX 60 IF CHK <> 32456 THEN PRINT "DATA ERROR": END
FX 70 SYS49152
PS 100 DATA162,0,189,60,3,149,115,232,224,23,208,2
    46,0,201,64,240,22,201
PF 120 DATA58,176,10,201,32,240,11,56,233,48,56,23
    3,208,96,76,116,164,234
SJ 140 DATA76,115,0,160,0,185,0,2,201,64,208,243,2
    00,185,0,2,201,0,240,9,201
FQ 160 DATA45,208,244,169,171,153,0,2,169,1,133,12
    2,160,1,24,185,0,2,32,107
FM 180 DATA169,32,19,166,160,0,32,121,0,32,107,169
    ,165,20,5,21,208,6,169
DK 200 DATA255,133,20,133,21,160,1,132,198,160,1,1
    32,15,177,95,240,175,32
DF 220 DATA44,168,32,215,170,134,25,132,26,173,198
    ,0,240,251,169,0,141,198
MJ 230 DATA0,166,25,164,26,200,177,95,170,200,177,
    95,197,21,208,4,228,20
```

## BASIC PROGRAMMING UTILITIES

---

QX 250 DATA240,2,176,44,132,73,32,205,189,169,32,1  
64,73,41,127,32,71,171  
SR 270 DATA201,34,208,6,165,15,73,255,133,15,200,2  
40,17,177,95,208,16,168  
RM 290 DATA177,95,170,200,177,95,134,95,133,96,208  
,163,108,6,3,16,218,201  
GA 310 DATA255,240,214,36,15,48,210,56,233,127,170  
,132,73,160,255,202,240,8  
QH 320 DATA200,185,158,160,16,250,48,245,200,185,1  
58,160,48,181,32,71,171,208  
CC 350 DATA245,0,230,122,208,2,230,123,173,0,2,201  
,58,240,10,201,32,240,239  
QF 370 DATA76,13,192,234,234,234,96

# ASCII/POKE Printer

---

---

*Todd Heimarck*

*Sometimes it would be handy to be able to calculate ASCII and POKE values. This utility lets you automatically calculate ASCII and POKE values as you're writing a BASIC program. For the Commodore 64 and 128 in 64 mode.*

Chances are, PRINTing to the screen was one of the first things you learned to do in BASIC. You probably also learned how to control where the computer prints by putting cursor commands within strings or by using SPC and TAB functions. The PRINT statement is common, primarily because it is so easy to use. But in certain situations, you may need to find out a character's ASCII number. And sometimes it is quicker to simply POKE a character onto the screen.

But before you can POKE, you have to know the character number. Let's put a row of hearts at the top of the screen. So, we need to POKE a bunch of 81's. Wait, those are solid circles. What's the number for hearts? I know that list is here somewhere in the reference book.

If you use POKes or ASCII values in programming, you know how annoying it is to flip back and forth through the reference book, losing time and patience. Even worse, you could lose the book and end up typing the character and PEEKing screen memory to get the POKE value.

## **Let the Computer Do the Work**

Your Commodore already knows the POKE values and ASCII numbers, so why not let it do the work?

This short machine language program, "ASCII/POKE Printer," does not use any BASIC memory. Its 52 bytes remain in the cassette buffer, ready to convert letters and graphics characters to POKE and ASCII numbers whenever you want.

Note that if you write a program that POKes any of the address locations of the cassette buffer (828-1019), you may lose ASCII/POKE Printer. Also, if you use a cassette player for saves, loads, or tape files, you will erase the machine language program. Fortunately, it is entirely relocatable, so if you



## BASIC PROGRAMMING UTILITIES

---

want to use the cassette buffer, you can change line 10 to move it to another part of memory. On the 64, it is usually safe to use any of the memory locations from 49152 through 53247.

### LOADing and Using the Program

Type in the program exactly as it appears. Make sure the DATA statements are exactly as printed. Save the program to tape or disk and VERIFY (if you have a cassette drive). Then run it and type NEW. The program is now in your cassette buffer. BASIC memory was cleared when you typed NEW, but it did not touch the cassette buffer.

Anytime you want to use ASCII/POKE Printer, type SYS 828. The computer will wait for you to type a character and then will display that character in the upper left corner with the ASCII value to the right and the POKE value below. Type another character, and you get two new values.

To exit back to BASIC, hold down SHIFT and press RETURN. This returns you to your program. SYS 828 will send you back to ASCII/POKE Printer, and so on. You can toggle back and forth as the need arises.

### Special Cases

There are some ASCII numbers that have no equivalent POKES. For example, adding CHR\$(13) to a string will force a RETURN after the string is printed. But ASCII 13 cannot be POKED to the screen. (What would a RETURN look like?) ASCII/POKE Printer will give you the correct ASCII numbers, but for certain characters, like RETURN, it will print a blank space and list a POKE of 32 (the number for a blank space). In the case of function keys, CLR/HOME, INST/DEL, and color commands, it will print a reverse-video character, as if in quote mode, and the correct ASCII number. But the POKE number will be wrong. Keys that perform a function—clearing the screen, for example—are not characters that can be POKED to the screen.

Also note that you cannot get values for reverse-video characters, which do not have separate ASCII numbers. To program a reverse character, precede it with a CHR\$(18). To POKE a reverse-video character, add 128 to the POKE value of the regular character.

This machine language utility will be most helpful when you are writing BASIC programs. By letting the computer tell

## CHAPTER 1

---

---

you ASCII and POKE values, you can really save time. The program was written to be short and simple, but if you are familiar with machine language, you can modify it to do much more.

### ASCII/POKE Printer

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
FR 10 FORJ=828T0879:READK:POKEJ,K:NEXT
DD 15 READY:IFY<>999THENSTOP
GM 20 DATA32,228,255,240,251,170,201,141,208,1,96,
    169,147
QG 21 DATA32,210,255,169,255,133,212,138,32,210,25
    5,169,32
BF 22 DATA32,210,255,169,0,32,205,189,169,13,32,21
    0,255
SH 23 DATA169,0,133,212,174,0,4,32,205,189,232,208
    ,204
QA 25 DATA999
```

# Auto Line Numbering

---

---

*Jeff Young*

*Programmers will find this short program to be a very handy, time-saving utility. For the Commodore 64 and 128 in 64 mode.*

“Auto Line Numbering” is a utility that automatically generates a line number for the current BASIC program statement being entered. As written, the program begins with line 100 and increments by tens (100, 110, 120, and so forth). You can modify this as described below.

## **How to Use the Program**

Auto Line Numbering is a BASIC program that loads a machine language subroutine into a free block of memory. The program puts the subroutine at memory location 49152 (\$C000). This area of memory will not be used by BASIC, so the program should be safe.

Type in the program and save it. After loading, type RUN, press RETURN, type NEW, press RETURN, type SYS 49152, and press RETURN. If you wish to leave the program for any reason, just press RETURN immediately after you see a new line number. To return to the program, type SYS 49160 and press RETURN. This will continue generating line numbers from where you left off.

Although the program will always begin numbering with 100 and increment by tens, you can modify either of these numbers if you wish. If you want to begin with a number other than 100, determine the number with which you want to start and then subtract ten. Next, POKE this number in low-byte/high-byte format into 251 and 252; then SYS 49160.

For example, if you want to begin with line 1000, subtract 10. The number you're now working with is 990. To determine low-byte/high-byte, divide 990 by 256. The result, 3, is the number you POKE into location 252—POKE252,3. The remainder of the division is 222. Now, POKE 251,222. The low byte is location 251, and the high byte, 252.

## CHAPTER 1

---

The lines you would type, then, if you wish to begin the line numbering with 1000 are

```
POKE 251,222:POKE 252,3
SYS 49160
```

To change the increment from ten, POKE the desired number into location 49179. If you want to increment by fives, for example, enter

```
POKE 49179,5
```

This utility program can save you a lot of time when you're programming, and it provides a neat, structured sequence for program line numbers.

### Auto Line Numbering

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
JM 1 X=49152
QD 2 READY:IFY=-1THEN4
KQ 3 POKE X,Y:X=X+1:Z=Z+Y:GOTO2
CB 4 IFZ<>12374THENPRINT"ERROR IN DATA STATEMENTS"
:END
FG 100 DATA169,90,133,251,169,0,133,252,169,19,141
,2,3,169,192,141,3,3,96,32,25
AR 110 DATA192,76,134,164,24,169,10,101,251,133,25
1,144,2,230,252,165,251,133,99
HC 120 DATA165,252,133,98,162,144,56,32,73,188,32,
221,189,162,0,189,1,1,240,9,32
XG 130 DATA210,255,157,0,2,232,208,242,32,18,225,2
01,13,240,3,76,105,165,56,165
SP 140 DATA251,233,20,176,2,198,252,169,131,141,2,
3,169,164,141,3,3,76,118,165,-1
```

CHAPTER 2

---

---

Screen Control

●  
●  
●  
●  
●  
●

●  
●  
●  
●  
●  
●

# Help Screens

---

---

*Jaffer Siddiqui*

*Create up to eight help screens with your BASIC and machine language programs. Disk drive required. For the Commodore 64 and 128.*

Many powerful programs include a lot of commands and options. In some cases, it's difficult or impossible to remember them all. And paging through a magazine article or a manual for the documentation can be awkward and inconvenient. "Help Screens" offers an elegant solution. By following a few simple steps, you can create up to eight help screens for a program.

Begin by typing in and saving a copy of Help Screens. Although most of the program is written in machine language, you can save, load, and run the program as if it were BASIC. The machine language is stored in the form of DATA statements. Program 1 is for the Commodore 64; Program 2 is for the 128.

When you type RUN, the machine language routines are POKEd into memory and the screen is cleared. When the flashing cursor returns, you're ready to begin creating your help screens.

Press the CLR key (SHIFT-CLR/HOME) to erase the screen; then type in the information you want displayed on the first help screen. (*Do not* press RETURN while you're working on this screen. If it is pressed, a SYNTAX ERROR will result, and your screen will be destroyed.) Use the cursor keys to move the cursor to any position on the screen. Be careful to avoid scrolling the top line off the screen.

When you've finished the help screen, 64 users should press CTRL-F (press F while holding down the CTRL key). If you're using the 128 version, press the ENTER key on the numeric keypad. Help Screens then stores the page in memory. The 64 version flashes the border color to signal that the page has been saved. The 128 version changes the border color. After a help screen has been saved, you can clear the screen and design the next one.

## CHAPTER 2

---

By following these steps, you can create up to eight screens. After the eighth one has been finished, a new program—HELPER.EXE—is created on your disk, and your help screens are saved into a file called HS. You may find that you need fewer than eight help screens. In the 64 version, just press CTRL-D after completing your last screen. In the 128 version, press SHIFT-ENTER.

It's a good idea to save your help screens on the same disk as the program they'll be used to support. For example, if you've created help screens for a program named "File-a-way," insert the disk that contains "File-a-way" and run "Help Screens." The "HELPER.EXE" and "HS" files will then be on the same disk.

### **In Action**

To use Help Screens, Commodore 64 users should enter

**LOAD "HELPER.EXE",8,1**

Commodore 128 users should enter

**BLOAD "HELPER.EXE"**

and then type **SYS 5632**.

Help Screens works with all BASIC programs and most machine language programs. Always load Help Screens (HELPER.EXE) first. After HELPER.EXE has been loaded, it loads the HS file that contains your help screens. Enter NEW; then load the program for which you created the help screens. Your help screens are now available—and only a keypress away.

If you're using a 64, press CTRL-H. If you're using a 128, press HELP. Page 1 (the first one created) is displayed when you first access the help screens. Commodore 64 users can flip through the pages by pressing CTRL-cursor right to page forward or CTRL-cursor up to go to the previous help screen. In the 128 version, use the up and down arrow keys to page through the help screens. To exit the help screens, press CTRL-DEL (64 version). In the 128 version, press the HELP key again. When you return to the help screens, you'll see the most recently viewed screen.

Running certain machine language programs—or pressing RUN/STOP-RESTORE—will prevent Help Screens from operating. If this happens, type SYS 679 (64 version) or SYS 5728 (128 version) to reactivate Help Screens.



# SCREEN CONTROL

---

---

## Program 1. Help Screens—64 Version

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
DJ 10 FORS=679TO766:READA:POKES,A:NEXT:FORS=820TO1
Ø16:READA:POKES,A:NEXT
ES 20 FORS=49152TO49397:READA:POKES,A:NEXT:PRINT"
{CLR}{WHT}PROGRAM ACTIVATED"
KC 30 PRINT"CTRL/F{2 SPACES}SCREEN FINISHED":PRINT
"CTRL/D{2 SPACES}DONE WITH SCREENS":SYS49152
:NEW
RC 40 DATA 173,245,3,208,18,173,20
QH 50 DATA 3,141,212,2,173,21,3
PR 60 DATA 141,213,2,238,245,3,32
AX 70 DATA 223,3,160,2,169,214,32
GM 80 DATA 160,3,88,169,131,141,2
XJ 90 DATA 3,169,164,141,3,3,76
FQ 100 DATA 116,164,76,0,0,173,141
FH 110 DATA 2,201,4,208,246,165,197
CE 120 DATA 201,29,208,240,160,234,169
AP 130 DATA 49,32,160,3,88,169,193
MJ 140 DATA 141,24,3,173,2,221,9
EH 150 DATA 3,141,2,221,173,21,208,72,76
HR 160 DATA 52,3,169,0,141,21,208,169,197
CP 170 DATA 141,0,221,173,32,208,72
SX 180 DATA 173,134,2,32,205,3,76
AA 190 DATA 79,3,24,32,168,3,32
KM 200 DATA 193,3,76,89,3,56,76
GP 210 DATA 76,3,238,32,208,165,197
HF 220 DATA 201,64,208,247,238,32,208
BF 230 DATA 173,141,2,201,4,208,246
FR 240 DATA 165,197,201,2,240,217,201
QA 250 DATA 7,240,223,201,0,208,232
MH 260 DATA 169,63,141,2,221,169,199
JK 270 DATA 141,0,221,169,21,141,24
KH 280 DATA 208,104,141,32,208,104,141
RC 290 DATA 21,208,160,2,169,214,32
AP 300 DATA 160,3,169,71,141,24,3
CQ 310 DATA 76,211,2,120,141,20,3
CQ 320 DATA 140,21,3,96,173,246,3
AE 330 DATA 176,10,105,16,208,2,169
QD 340 DATA 128,141,246,3,96,233,16
RH 350 DATA 201,128,176,246,169,240,144
MA 360 DATA 242,173,24,208,41,15,13
HX 370 DATA 246,3,141,24,208,96,162
BR 380 DATA 0,157,0,216,157,255,216
HF 390 DATA 157,254,217,157,253,218,232
MH 400 DATA 208,241,96,169,1,162,8
RF 410 DATA 168,32,186,255,169,2,162
XJ 420 DATA 247,160,3,32,189,255,169
SX 430 DATA 0,76,213,255,0,128,72,83
```

## CHAPTER 2

---

QB 440 DATA 169,17,160,192,32,201,192  
QX 450 DATA 88,169,193,141,24,3,96  
SS 460 DATA 76,49,234,173,141,2,201  
XF 470 DATA 4,208,246,165,197,201,18  
RR 480 DATA 240,71,201,21,208,236,32  
JX 490 DATA 209,192,32,197,192,88,169  
RG 500 DATA 0,133,95,169,4,133,96  
MC 510 DATA 169,232,133,90,169,7,133  
HC 520 DATA 91,173,244,192,133,88,173  
GP 530 DATA 245,192,133,89,32,191,163  
PF 540 DATA 238,32,208,162,0,32,179  
QG 550 DATA 238,232,208,250,206,32,208  
MS 560 DATA 173,231,192,201,8,240,10  
HK 570 DATA 169,17,160,192,32,201,192  
EG 580 DATA 76,14,192,32,197,192,88  
QS 590 DATA 169,167,141,2,3,133,253  
GE 600 DATA 169,2,141,3,3,133,254  
QJ 610 DATA 32,222,192,169,10,162,232  
QD 620 DATA 160,192,32,189,255,162,250  
EX 630 DATA 160,3,169,253,32,216,255  
CF 640 DATA 32,222,192,169,2,162,242  
PE 650 DATA 160,192,32,189,255,169,0  
MF 660 DATA 133,253,169,160,133,254,165  
HQ 670 DATA 1,41,254,133,1,169,253  
QE 680 DATA 162,255,160,191,32,216,255  
SD 690 DATA 169,71,141,24,3,169,131  
KR 700 DATA 141,2,3,169,164,141,3  
RP 710 DATA 3,165,1,9,1,133,1  
MB 720 DATA 96,169,49,160,234,120,141  
ED 730 DATA 20,3,140,21,3,96,173  
DF 740 DATA 245,192,24,105,4,141,245  
DP 750 DATA 192,238,231,192,96,169,1  
QJ 760 DATA 162,8,160,1,76,186,255  
SH 770 DATA 0,72,69,76,80,69,82  
CF 780 DATA 46,69,88,69,72,83,232,159

### Program 2. Help Screens—128 Version

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
FE 10 PRINT "LOADING DATA"  
AF 20 FOR I=3072 TO 3249:READ A:POKE I,A:NEXT  
BG 30 FOR I=5632 TO 6045:READ A:POKE I,A:NEXT  
JH 40 BSAVE "HELPER.EXE",P5632 TO P6046  
EM 50 PRINT "INSTALLING HELP SCREEN MAKER":SYS3072  
KC 60 END  
BR 3072 DATA173,60,3,141,33,12,173,61  
XD 3080 DATA3,141,34,12,169,28,141,60  
XD 3088 DATA3,169,12,141,61,3,169,192
```

## SCREEN CONTROL

---

KE 3096 DATA141,76,12,96,192,76,240,3  
KA 3104 DATA76,255,255,224,1,240,79,224  
BD 3112 DATA0,208,245,173,41,10,164,236  
QD 3120 DATA145,224,169,4,141,73,12,162  
SF 3128 DATA0,120,173,6,213,9,3,141  
BP 3136 DATA6,213,169,127,141,0,255,189  
JG 3144 DATA0,4,157,0,192,232,208,247  
XQ 3152 DATA238,73,12,238,76,12,173,73  
XE 3160 DATA12,201,8,208,234,169,0,141  
MD 3168 DATA0,255,173,6,213,41,252,141  
SK 3176 DATA6,213,88,238,32,208,173,76  
HR 3184 DATA12,201,224,240,1,96,169,1  
QM 3192 DATA162,0,32,104,255,169,0,162  
SH 3200 DATA8,168,32,186,255,169,4,162  
CM 3208 DATA174,160,12,32,189,255,169,0  
JS 3216 DATA133,253,169,192,133,254,169,253  
XK 3224 DATA162,255,160,223,32,216,255,120  
QD 3232 DATA173,33,12,141,60,3,173,34  
GX 3240 DATA12,141,61,3,88,96,48,58  
XQ 3248 DATA72,83  
KH 5632 DATA169,0,141,0,255,173,6,213  
BJ 5640 DATA9,3,141,6,213,169,79,141  
MJ 5648 DATA0,255,162,0,189,0,208,157  
AC 5656 DATA0,224,232,208,247,238,22,22  
QA 5664 DATA238,25,22,173,22,22,201,224  
JE 5672 DATA208,234,169,0,141,0,255,173  
RR 5680 DATA6,213,41,252,141,6,213,169  
CJ 5688 DATA1,162,0,32,104,255,169,1  
CC 5696 DATA162,8,160,0,32,186,255,169  
MH 5704 DATA2,162,119,160,22,32,189,255  
XS 5712 DATA169,0,162,0,160,192,32,213  
EP 5720 DATA255,169,192,133,58,32,129,175  
MB 5728 DATA173,60,3,141,131,22,173,61  
PC 5736 DATA3,141,132,22,169,126,141,60  
FM 5744 DATA3,169,22,141,61,3,96,72  
SM 5752 DATA83,8,0,216,0,240,201,132  
QX 5760 DATA240,3,76,255,255,169,118,141  
XH 5768 DATA60,3,120,173,24,3,72,169  
EH 5776 DATA51,141,24,3,173,25,3,72  
RG 5784 DATA169,255,141,25,3,165,216,72  
GG 5792 DATA169,0,133,216,165,217,72,169  
KB 5800 DATA4,133,217,173,44,10,72,173  
PR 5808 DATA121,22,141,44,10,173,21,208  
SX 5816 DATA72,169,0,141,21,208,173,33  
HF 5824 DATA208,72,169,0,141,33,208,173  
BR 5832 DATA2,221,9,3,141,2,221,173  
MX 5840 DATA0,221,72,41,252,9,0,141  
HS 5848 DATA0,221,173,6,213,72,41,63  
HX 5856 DATA9,67,141,6,213,32,108,23  
KG 5864 DATA88,165,212,201,88,208,250,165  
AX 5872 DATA212,201,64,240,24,201,83,240

## CHAPTER 2

---

MQ 5880 DATA7,201,84,208,242,169,240,44  
XE 5888 DATA169,16,24,109,44,10,41,127  
MK 5896 DATA141,44,10,16,220,165,212,201  
KC 5904 DATA88,208,250,120,169,22,141,7  
EB 5912 DATA213,173,6,213,9,3,141,6  
GR 5920 DATA213,169,112,141,0,255,160,0  
PG 5928 DATA198,123,198,125,177,124,145,122  
FG 5936 DATA200,208,249,165,123,201,216,208  
HR 5944 DATA239,169,0,141,0,255,141,7  
QD 5952 DATA213,104,141,6,213,104,141,0  
CP 5960 DATA221,104,141,33,208,104,141,21  
RP 5968 DATA208,173,44,10,141,121,22,104  
CF 5976 DATA141,44,10,104,133,217,104,133  
RA 5984 DATA216,104,141,25,3,104,141,24  
KG 5992 DATA3,76,108,22,169,112,141,0  
BM 6000 DATA255,169,22,141,7,213,160,0  
PA 6008 DATA177,122,145,124,169,13,145,122  
FR 6016 DATA200,208,245,230,123,230,125,165  
AC 6024 DATA123,201,220,208,235,169,0,141  
EX 6032 DATA0,255,141,7,213,173,6,213  
HG 6040 DATA41,252,141,6,213,96

# Blick

---

---

## *Plummer Hensley*

*This short utility spices up your programs by adding a blink and a click to the PRINT statement. Anytime you type a character to the screen, you'll see an underline cursor accompanied by a brief sound. For the Commodore 64 and 128.*

If you don't think sound is important, try playing your favorite action game with the volume turned all the way down. It's just not as much fun without the explosions, zaps, and other noises.

Sounds help to liven up games—so why not make PRINT statements a little more interesting? This program gives you a blink and a click (a *blick*) every time a character is printed.

### **Typing It In**

Enter the version written for your computer, and save it to tape or disk before proceeding. Saving is important because the last command in line 120 is a NEW, which erases the program currently in memory.

“Blick” is written in machine language (ML), but you don't need to know machine language to use it. It is presented in the form of a BASIC loader that reads DATA statements and POKES the routine into memory. After running it, you should see the message BLICK ENABLED.

Once Blick is in memory, try printing a message—PRINT “THIS IS BLICK”, for example. Or load a program and list it. See the table for ideas on customizing the program.

If you should accidentally disable Blick by pressing RUN/STOP-RESTORE or RUN/STOP-RESET, enter the appropriate SYS from the table to reenable Blick. To turn it off, enter the two POKES listed. (Note: enter them on the same line, separated by a colon.) To change the cursor character, POKE the appropriate ASCII value into the location listed. Finally, you can modify the blinking speed with a POKE to the address specified in the table.

## CHAPTER 2

### Important Blick Locations

	64	128
Enable	SYS 679	SYS 3072
Disable	POKE 806,202: POKE 807,241	POKE 806,121: POKE 807,239
Change cursor (POKE location with any ASCII value ( <i>x</i> ))	POKE 728, <i>x</i>	POKE 3128, <i>x</i>
Change blinking speed (POKE location with 0–255 ( <i>y</i> ); numbers greater than 234 speed up cursor)	POKE 733, <i>y</i>	POKE 3133, <i>y</i>

### How It Works

Blick is a *wedge* that temporarily diverts the PRINT statement into a routine that prints an underline character, makes a sound, and erases the underline. When it's finished, it goes on to the main PRINT statement.

PRINT is a common, easy-to-use statement in BASIC. But at the machine language level, PRINT is more complex; it has to do a lot of work. First, the computer looks ahead to see whether it will be printing a variable, a number, a string, or maybe even a long calculation. Once that's straightened out and BASIC knows the sequence of characters to be printed, it goes through the Kernal routine for printing characters (always at location \$FFD2 on the 128 and 64). The Kernal routine looks at locations 806–807 to find the actual ROM routine for printing a character.

This pointer was deliberately designed to be the weak link in the process. If we change the address there, anytime the computer wants to print a character, it runs into a detour we have set up. This detour handles the blink and the click before jumping back to the main PRINT routine.

**A word of caution:** The 64 version of Blick is subject to the infamous lockup bug which affects version 2 of ROM. To see if your 64 has version 2 of the operating system, start with a newly powered up 64 and put the cursor at the bottom of the screen. Hold down the space bar until it travels across two complete screen lines. After the cursor has wrapped around to

## SCREEN CONTROL

---

the beginning of the next line exactly twice, use the DEL key to move it back to the end of the previous line. If the screen says LOAD and the computer locks up, you've got version 2 ROM. This lockup happens only when the cursor color is red, cyan, blue, yellow, light red, dark gray, light blue, and light gray. If you limit character colors to black, white, purple, green, orange, brown, medium gray, and light green, you'll be safe. It's also a good idea to limit your printing to strings of 79 characters or fewer.

### Program 1. Blick—64 Version

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
DP 100 FORI=679TO758:READA:POKEI,A:CK=CK+A:NEXT
BP 110 IFCK<>11167THENPRINT"ERROR IN DATA STATEMEN
TS.":STOP
BD 120 SYS679:PRINT"BLICK ENABLED":NEW
SF 130 DATA 169,15,141,24,212,141,19,212,169,120
SM 140 DATA 141,15,212,169,1,141,14,212,169,0
JX 150 DATA 141,20,212,162,201,160,2,142,38,3
MC 160 DATA 140,39,3,96,32,202,241,133,251,134
MD 170 DATA 252,132,253,169,233,141,18,212,169,175
MP 180 DATA 32,202,241,162,234,160,0,200,208,253
BH 190 DATA 232,208,250,169,32,141,18,212,169,20
RM 200 DATA 32,202,241,165,251,166,252,164,253,96
```

### Program 2. Blick—128 Version

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
BA 100 FORI=3072TO3158:READA:POKEI,A:CK=CK+A:NEXT
XD 110 IFCK<>10998THENPRINT"ERROR IN DATA STATEMEN
TS.":STOP
FE 120 SYS3072:PRINT"BLICK ENABLED":NEW
SF 130 DATA 169,15,141,24,212,141,19,212,169,120
SM 140 DATA 141,15,212,169,1,141,14,212,169,0
RH 150 DATA 141,20,212,162,34,160,12,142,38,3
QG 160 DATA 140,39,3,96,72,169,0,141,0,255
SD 170 DATA 104,32,121,239,133,167,134,168,132,169
FG 180 DATA 169,33,141,18,212,169,175,32,121,239
PS 190 DATA 162,234,160,0,200,208,253,232,208,250
SS 200 DATA 169,32,141,18,212,169,20,32,121,239
EF 210 DATA 165,167,166,168,164,169,96
```

# Input Windows

---

---

*Thorpe Thompson*

*This machine language routine can give your BASIC programs a highly professional look. It adds screen windowing capability—you can choose the window size—for user input. For the Commodore 64 and 128 in 64 mode.*

When you're programming, it's important to maintain tight control over input. You can use the INPUT statement, but it's often susceptible to unwanted results. "Input Windows," a machine language utility in the form of a BASIC program, functions just like an INPUT statement, but it gives you more control over the process by creating a window for inputting a response from a user. The window, which can easily be positioned anywhere on the screen, defines the size of the input field and the active area of the editing keys (CRSR-right/CRSR-left and INST/DEL). When the RETURN key is pressed, the input data is placed in T\$ or T1, depending on whether you require string or numeric data from the user.

## Using the Routine

Type in and save Program 1. Type the DATA statements carefully—one incorrect digit can make a big difference in machine language. The program keeps track of a checksum value, so it will not write an executable file to the disk unless all the data items are correct. When you have a good file, you can load it into your BASIC program with the following line:

```
5 IF A=0 THEN A=1: LOAD "INPUT  
.OBJ",8,1
```

If you're using tape, change the 8 to 1. Next, you need to add this subroutine to your program:

```
10000 POKE 142,LNG: POKE 143,TYP  
10010 SYS49152: IF (1 AND ST) THEN  
T$=" ": T1=0  
10020 RETURN
```



## SCREEN CONTROL

---

You set LNG to the field size (in characters) and TYP to the data type (0=string/1=numeric) prior to calling the subroutine.

The left edge of the input window will be placed at the current cursor position—you can position the window with PRINT statements. For example, if you want the window to start at the fifth row from the top and in the tenth column, you could use this line:

```
100 PRINT "{HOME}{5 DOWN}  
{10 RIGHT}";
```

Don't forget to put a semicolon on the end of the line, or the window will be placed one row below the one you want.

Suppose you want the window to start next to a screen prompt. Since the position is determined by the current cursor position, you can use the prompt PRINT statement to position the window like this:

```
100 PRINT "ENTER YOUR NAME- ";
```

Here, again, the semicolon must not be forgotten. The trailing space on the screen prompt separates the window from the prompt.

Use the parameters LNG and TYP to control the input data. LNG is set to the maximum size of the input field in characters. If you wanted to input a string of ten characters in length, you would set LNG to 10 before calling the subroutine. (The data need not be ten characters in length, but it can be no greater than ten.) You must also set TYP to the type of data to be input. If TYP=0, the machine language routine treats the data as string input. A TYP of 1 causes the data to be treated as numeric input.

Let's set up the code to input a name with a maximum length of 20 characters:

```
100 PRINT "ENTER YOUR NAME- ";  
110 LNG=20:TYP=0:GOSUB 10000
```

After the program returns from the GOSUB call, the data will be in the variable T\$. You must transfer the value to another variable before calling the subroutine again or the data will be lost. As an example of numeric input, this code could

## CHAPTER 2

---

be used to input a dollars-and-cents amount in the range of \$0.00 to \$99.99:

```
100 PRINT "ENTER THE PRICE-$";  
110 LNG=5:TYP=1:GOSUB 10000
```

This time you have to set TYP to 1. Note also that LNG was set to 5. This is necessary because the decimal point counts as one character in the field size. It's possible to enter an amount as large as \$99999 by omitting the decimal point, so you have to check the data after the GOSUB call to insure that it's valid data.

### A Demonstration

To see how Input Windows works, type in Program 2, "Demo," and save a copy. Change the 8 in line 100 to 1 if you're using tape. Be sure Program 1 is on the disk (or immediately following Program 2 if you're using tape). Load Demo and type RUN. The machine language file created by Program 1 will automatically be loaded into memory.

### How the Routine Works

When the BASIC subroutine at line 210 is called, the values of LNG and TYP are stored in zero page for access by the machine language routine. Next, the SYS statement causes the program to execute the machine language code at 49152 (\$C000). The machine language waits for a key to be pressed. When it reads a key, it first checks the key against a table of values to see whether it needs to execute a function (such as INSerT, CRSR right, and so on). If the keypress is not a function, the value is tested to insure that it's in the range of printable characters. If the key is out of range, the routine goes back to fetch another keypress. If the value is within range, the routine displays the keypress on the screen and stores the ASCII value of the key in the input buffer. This process continues until the RETURN key is pressed.

Now the routine transfers the data to a special buffer and sets up a "fake" BASIC line in high memory. The CHRGET routine is vectored to point to the pseudo-BASIC statement, and the LET routine in BASIC ROM is executed, equating the variable with the input data. Finally, the CHRGET routine is revectored to its original address in the BASIC program, and the program returns from the SYS.

## SCREEN CONTROL

---

Execution continues with the IF statement. If no data has been entered before the RETURN key is pressed, the status variable (ST) will be set to 1. Otherwise, ST will have a value of 0. If ST is set to 1, both variables are cleared, and the program returns from the GOSUB call empty-handed. When the condition is false, the proper variable will hold the input data.

### Wrapping It Up

This routine behaves differently, depending on which character set you're using. When set 1 is in use, the routine accepts numbers, punctuation characters, and uppercase letters as valid characters. This prevents the user from entering graphics characters as input data. If set 2 is being used, the valid characters are numbers, punctuation characters, and uppercase or lowercase letters. The field size can be from 1 to 75 characters. Characters which would be interpreted as delimiters by the INPUT statement (such as commas) are accepted as valid data in the string input mode. The sign characters, negative and positive, are accepted as valid data in numeric input mode. CRSR-up/CRSR-down and CLR/HOME are not active during either input mode.

You can use the routine with any screen unless it's located under BASIC or Kernal ROM. The screen address is figured by the routine each time it's called, so you can switch screens in your program without any problem. By using this controlled input routine, you can prevent unwanted results from occurring at input points and make your programs less reliant on the user "playing by the rules."

### Program 1. Controlled Keyboard Input

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
HR 100 PRINT "{CLR}READING DATA STATEMENTS..."
KR 110 FORB=49152TO49604:READD:POKEB,D:CK=CK+D:NEXT
      T
KG 120 IFCK<>57716THEN PRINT"ERROR IN DATA STATEME
      NTS":END
JG 130 PN$="INPUT.OBJ":FORJ=1TOLEN(PN$):POKE704+J,
      ASC(MID$(PN$,J,1)):NEXTJ
ME 140 PRINT"{DOWN}{RVS}D{OFF}ISK OR {RVS}T{OFF}AP
      E? "":DEVICE=8
JJ 150 GETA$:IFA$="T"THENDEVICE=1:GOTO170
RE 160 IFA$<>"D"THEN150
DK 170 PRINTA$:POKE780,15:POKE781,DEVICE:POKE782,2
      55:SYS65466
```

## CHAPTER 2

---

HP 180 POKE780,LEN(PN\$):POKE781,193:POKE782,2:SYS6  
5469  
QS 190 BA=49152:HI=INT(BA/256):LO=BA-HI\*256:POKE25  
1,LO:POKE252,HI  
RK 200 EA=49604:HI=INT(EA/256):LO=EA-HI\*256+1:POKE  
780,251:POKE781,LO:POKE782,HI  
XH 210 PRINT"SAVING ML VERSION OF "PN\$:SYS65496  
PG 220 DATA 56,32,240,255,132,139,173,136  
QS 230 DATA 2,168,169,0,202,48,8,24  
BJ 240 DATA 105,40,144,248,200,208,245,24  
ES 250 DATA 101,139,144,1,200,133,139,133  
FJ 260 DATA 167,132,140,152,24,105,212,133  
RK 270 DATA 168,173,24,208,41,2,141,195  
XK 280 DATA 193,240,4,169,127,208,2,169  
JK 290 DATA 63,141,240,192,162,87,169,32  
DG 300 DATA 157,0,2,202,16,250,232,134  
EK 310 DATA 141,164,141,177,139,9,128,145  
FP 320 DATA 139,32,228,255,240,251,72,164  
GR 330 DATA 141,177,139,41,127,145,139,104  
DQ 340 DATA 162,4,221,29,193,240,5,202  
AF 350 DATA 16,248,48,29,224,4,208,3  
MH 360 DATA 76,42,193,138,10,170,189,34  
HS 370 DATA 193,141,132,192,232,189,34,193  
AD 380 DATA 141,133,192,32,0,16,76,73  
CC 390 DATA 192,201,32,144,188,201,96,144  
DX 400 DATA 8,201,193,144,180,201,219,176  
AG 410 DATA 176,164,141,196,142,240,170,174  
MF 420 DATA 195,193,208,7,201,96,144,3  
JX 430 DATA 56,233,128,153,0,2,32,235  
KD 440 DATA 192,173,134,2,145,167,230,141  
GS 450 DATA 76,73,192,166,141,228,142,240  
GD 460 DATA 136,230,141,96,166,166,141,240,129  
FS 470 DATA 198,141,96,164,141,208,1,96  
XR 480 DATA 198,141,185,0,2,136,153,0  
AQ 490 DATA 2,32,235,192,200,200,196,142  
JA 500 DATA 144,240,169,32,153,0,2,32  
AP 510 DATA 235,192,96,201,193,144,5,41  
DB 520 DATA 127,76,250,192,201,65,144,2  
AC 530 DATA 41,63,145,139,96,164,141,196  
DR 540 DATA 142,208,1,96,169,32,72,185  
XF 550 DATA 0,2,170,104,153,0,2,32  
PC 560 DATA 235,192,200,196,142,240,5,138  
JS 570 DATA 72,76,7,193,96,29,157,148  
PA 580 DATA 20,13,187,192,196,192,253,192  
QM 590 DATA 203,192,169,0,133,144,162,79  
MQ 600 DATA 189,0,2,201,32,208,6,202  
CE 610 DATA 16,246,230,144,96,232,134,142  
XA 620 DATA 165,143,208,13,162,183,160,193  
HE 630 DATA 142,94,193,140,95,193,76,91  
JJ 640 DATA 193,162,188,160,193,142,94,193  
SM 650 DATA 140,95,193,160,0,185,0,16

## SCREEN CONTROL

---

```
EC 660 DATA 240,7,153,52,3,200,76,93
FR 670 DATA 193,162,0,189,0,2,153,52
SR 680 DATA 3,200,232,228,142,208,244,165
KM 690 DATA 143,208,9,169,34,153,52,3
XG 700 DATA 200,76,144,193,169,34,153,52
GH 710 DATA 3,200,169,41,153,52,3,200
EF 720 DATA 169,0,153,52,3,165,122,141
QS 730 DATA 181,193,165,123,141,182,193,169
QR 740 DATA 52,133,122,169,3,133,123,32
RB 750 DATA 165,169,173,181,193,133,122,173
HS 760 DATA 182,193,133,123,96,0,0,84
QA 770 DATA 36,178,34,0,84,49,178,197
XJ 780 DATA 40,34,0,0,170
```

### Program 2. Demo

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
QX 100 IFA=0 THEN A=1:LOAD"INPUT.OBJ",8,1
CM 110 PRINT"[CLR]"{3 DOWN}ENTER YOUR NAME- ";
HQ 120 LNG=22:TYP=0:GOSUB210
DA 130 PRINT:PRINTT$:PRINT"{2 DOWN}ENTER THE PRICE
-$";
BF 140 LNG=5:TYP=1:GOSUB210
GK 150 IFT1>99.99 THEN 140
CC 160 PRINT:PRINTT1:PRINT:PRINT"{DOWN}MORE (Y/N)?
"
RC 170 WAIT198,1:GETK$
JX 180 IFK$="Y" THEN 110
SH 190 IFK$<>"N" THEN 170
MB 200 END
ES 210 POKE142,LNG:POKE143,TYP:SYS49152:IF(ST AND
{SPACE}1) THEN T$="":T1=0
CD 220 RETURN
```

# Blink Mode on Commodore Machines

---

*David Sanner*

*Here's a way to bring your color screen to life. Two demonstration programs are included along with a good tutorial and lots of programming explanations and tips. For the Commodore 64 and 128 in 64 mode.*

Have you ever seen the Apple II's "video blink mode" and wished for a similar feature on your Commodore computer? Here's a program that provides that feature for the Commodore 64 and 128 in 64 mode. Let's look at what "Blink Mode" can do.

The basic purpose of the program is to allow text characters to blink (switch) between two different colors at varying speeds. For instance, you can tell Blink Mode to blink the color white to the color black. Thereafter, any characters printed to the text screen in white will alternate between white and black. You aren't, of course, restricted to just white and black; you can get 8 colors blinking to 8 different colors, and, in special cases, you can get all 16 colors blinking to different colors.

## **The USR Interface**

In this program, the USR function controls what Blink Mode does for you. If you're not familiar with this powerful, but little used instruction, here's a bit of background. Two bytes in memory are reserved for use with the USR function. These bytes hold the address (in standard low-byte/high-byte format) of the location in memory to which the computer will go when USR is executed. You have to make this address point to the location of Blink Mode with the following POKES:

**POKE 785,0: REM LOW BYTE OF \$C000**  
**POKE 786,192: REM HIGH BYTE \$C000**

As you can see, Blink Mode is located at \$C000. The first POKE sets up the low byte of this address, which is \$00. The

## SCREEN CONTROL

---

second POKE sets up the high byte of the address (\$C0 is 192 in decimal, \$3C is 60, and \$1C is 28). Once these values are in place, the computer will jump to Blink Mode every time you use the USR function.

Controlling an interrupt-driven routine like Blink Mode can often be tricky business. I have tried to minimize the amount of interaction you need to get it to do interesting things—which is why I chose to use the USR function. It not only calls up a machine language routine (like the SYS statement), but it also lets you send a value to the routine quite easily (on the 64, SYS doesn't allow you to send a value to the routine).

Additionally, the USR function returns a value to the program. This means that Blink Mode can tell you whether your command is okay by simply passing back different numbers. If everything has gone according to plan, Blink Mode will pass back a value of 1. If something has gone awry, it will return a 0. This is helpful should you ever want to test whether the command worked in your program.

Because the USR function returns a value, you must assign the value to a variable. This means you must use the USR function in the following manner:

**X = USR(*data*)**

In the above example, the important part is *data*, the number sent to Blink Mode. The data to be passed to the routine via a USR function must be enclosed within the parentheses, as in X=USR(16384). Once Blink Mode has set things up, it returns a value to your BASIC program; the variable X will hold the value that the USR function passes back.

### **How to Use Blink Mode**

There are seven commands that you can pass to Blink Mode via the USR function. In order to pass a command to Blink Mode, you must send a specific number via the USR function. The number depends on what command you send. For instance, to send the Enable Blink Mode command, you must call Blink Mode like this: X=USR(4096). Whenever I use Blink Mode in a BASIC program, I set variables to equal the command numbers, as with EB=4096. Then I can use them in the USR function: X=USR(EB). Some of the commands require that you send data as well as the command number.

## CHAPTER 2

---

Here are the commands for Blink Mode.

**Enable Blink Mode (4096).** Every 1/60 second, the computer interrupts itself, saves what it was doing, and jumps to a special location in memory. Normally, this location is the start of some operating system routines that perform necessary tasks like checking the keyboard and advancing the jiffy clock. When Blink Mode has been enabled, however, this location points to the part of Blink Mode that takes care of blinking the colors. After Blink Mode has finished running, it jumps to the normal interrupt location. This kind of program is often known as an *interrupt wedge*. The Enable command must be sent if you want to see the colors blink. You can send it with  $X=USR(4096)$ .

**Disable Blink Mode (8192).** This command instantly stops Blink Mode from being an interrupt wedge. You can send this command by  $X=USR(8192)$ .

**Blink Color (1024).** Use this command when you want to make a specific color blink to another color. Just add the number of the color you want to blink to the command number (1024). Then add 16 times the number of the color you want to blink to. For example, if you wanted the color white (color 1) to blink to black (color 0), you would send this command:  
 $X=USR(1024+(16*0)+1)$

(Note that color numbers range from 0 through 15.) The original color and the color you want it to blink to should always be different.

**Stop Color From Blinking (2048).** If you want to stop a specific color from blinking, send this command. Just add the number of the color you want to stop blinking to the command number (2048). For instance, if you wanted to stop cyan from blinking to some other color, you would send  $X=USR(2048+3)$ . Three is the color number for cyan.

It's important to note that you should turn off only a color that has been set to blink, not the color it is blinking to. In other words, if you tell cyan to blink to red, you should tell cyan to stop blinking. This command will wait until all characters typed in the requested color have returned to that color before stopping them from blinking. For example, if green is told to blink to black, and a "stop green from blinking" command is sent, Blink Mode will wait until black characters have turned back to green before stopping green from blinking.



## SCREEN CONTROL

---

**Set Delay (16384).** This command allows you to set the time it takes to blink between two colors. This time ranges from 1/60 second to about four and a half minutes. (Note that all blinking colors will blink at the same rate.) You send this command by passing the command number (16384) plus the time value. Time values range from 1, which sets the time delay to about 1/60 second, to 16383, which sets the delay to about 16383/60 seconds (four and a half minutes). Every time you add one to the time value, you add 1/60 second to the delay. The number 60 will make the blink time about one second.

You send the command like this:  $X=USR(16384+2)$ . This call will set the time between color changes to 2/60 second. Note that the lower this number, the faster the blink, but the slower everything else—like your BASIC program. The reason is that Blink Mode must update the entire color screen whenever the time value you have set runs out. If the time value is very small, you'll notice things slowing down because Blink Mode is running more frequently.

**Return Value (512).** Normally, when you're using Blink Mode, the function  $X=USR$  will make  $X$  either 1 or 0, depending on whether or not Blink Mode understands the instruction. However, you can also pass back other values. Let's say, at some point in your program, that you want to know which color blue has been blinking to. By using the Return Value command, you can determine this. Simply send the command number (512), plus the number of the color to be checked. For instance, the function  $X=USR(512+6)$  will assign to  $X$  the color number that blue has been told to blink to. Note that if a color is not blinking to another color it will return itself. For example, if green is not blinking to another color,  $X=USR(512+5)$  should return 5, the number for green.

**Reset All Colors (256).** There will be times when you'll be experimenting with colors, and things just get out of hand. If that happens, the Reset All Colors command will reset all colors to blink to themselves, thus making it appear that none of them is blinking. To use this command, send the command number (256) to Blink Mode. Like the Stop Blinking command, this command will wait until the colors have returned to their original values before resetting them.

### Using Blink Mode in Your Own Programs

There are a few important items to remember when you write programs to use with Blink Mode. First, each color can blink to only one other color. Second, each color must blink to a color that no other color is blinking to. For instance, if you have green blinking to red, you should not try to have white blinking to red also. If you do this, you'll get strange (but often interesting) results.

Always reset a color from blinking before you make it blink to another color. Because Blink Mode waits until the color has returned to its original value, however, this command may take awhile, depending on the time delay between blinks. One way to get around this problem is to use the following lines:

```
10 X =USR (2048+6): REM STOP BLUE FROM BLINKNG
20 X =USR (16384+1): REM SET DELAY TO MIN.
30 X =USR (512+6): IF X<>6 THEN 30: REM WAIT
40 X =USR (16384+600): REM RESET ORIGINAL BLINK
  DELAY
```

Let's assume that you've set the time delay to a very high value (say, ten seconds). When you send the "stop blue from blinking" command, it could take up to ten seconds for blue to stop blinking. (For example, if Blink Mode has just changed all the blue to red and you send this command, it will be ten seconds before Blink Mode can change all the red back to blue. Once it has done that, it will stop blue from blinking to red.) So, in order to speed things up a bit, I set the time delay to a minimum; this stops blue from blinking almost immediately. Line 30 checks the value of the color that blue is blinking to. If blue has been reset, it will be blinking to blue. If it hasn't been reset, we simply keep running line 30 until it has been. Line 40 resets the time delay to its original value. (You can also use these BASIC lines with the Reset All Colors command.)

If you don't care whether the color gets reset (to its original color) before it stops blinking, you can use the following lines:

```
10 X =USR (1024+(16*6)+6): REM BLINK BLUE TO BLUE
20 X =USR (1024+(16*3)+3): REM BLINK CYAN TO CYAN
```

In this example, we assume that blue has been told to blink to cyan. What these lines do is to reset the colors man-

## SCREEN CONTROL

---

ually by telling them to blink to themselves. The benefit of this sequence of commands is that the action happens immediately: there's no waiting for the time delay. Unfortunately, there's no guarantee that the cyan will have blinked back to blue when you send this command.

### **A Little More Advanced**

This section has been saved for last because it deals with more complex ideas and involves a little more programming. Normally, you can get 8 colors on the screen blinking to 8 other colors with no problem. This may not always work right if you try to blink all 16 colors or try to blink a color to a color that has been told to blink (for instance, if red has been told to blink to white, and you tell yellow to blink to red).

Since there's no way for Blink Mode to know whether the characters on the screen are blinking to a color or whether they started out as that color, Blink Mode will treat them as the same color. Using the example from the start of this section, you may try to type something in red. As Blink Mode runs in the background, it has no way of knowing that the red you have typed on the screen is not the red that it has just changed the yellow to. So, it will just change all red back to yellow, and you'll never see white.

There is, however, a way to get around this problem. It involves more planning and, as you'll see, cannot be used "on the fly." Here's what you need to do: Disable Blink Mode to prevent it from trying to change any colors. Set up any colors that you want to blink. Then, any text that you have should be written to the screen in the original color. Finally, simply enable Blink Mode. The colors will begin to blink back and forth.

I call this technique "double-blink" because two text lines will trade colors back and forth. Using this method can provide quite interesting effects. As an example, let's say you want to have two lines of text blink back and forth between white and black. You follow the above guidelines, and write one text line in black and the other in white. You set up everything else and then enable Blink Mode. For one second (or whatever time delay you have prescribed) the first line is black and the second is white; during the next second the colors switch, and so on. This is a nice effect, but it must be controlled. For example, if you try to type something in white

## CHAPTER 2

now, some of the text you type will be out of sync with the rest that you type.

If all this seems very complex, don't worry. As with most other things, experimentation is the best way to discover. The two programs included here demonstrate some of the things you can do with Blink Mode. Program 2 is a basic demonstration of Blink Mode. Program 3 demonstrates the technique of double-blink. Each of these demos automatically loads and runs the machine language portion of the main Blink Mode program (Program 1)—provided that the main program has been saved as BLINKOBJ on the same disk. If you've saved the main program with another filename, be sure to change it in line 99 of Programs 2 and 3. If you're using tape, change the load instruction in this same line to LOAD "BLINKOBJ",1,1.

Incidentally, I've used Blink Mode with a 300-baud modem and terminal program with no problem. The time delay was set to about 10.

### Program 1. Blink Mode

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
XA 100 FORI=49152TO49506:READA:POKEI,A:X=X+A:NEXT
XG 110 IFX<>36438THENPRINT"ERROR IN DATA.":STOP
CG 120 PN$="BLINKOBJ":FORJ=1TOLEN(PN$):POKE704+J,A
SC(MID$(PN$,J,1)):NEXTJ
SP 130 PRINT"{DOWN}{RVS}D{OFF}ISK OR {RVS}T{OFF}AP
E? ";:DEVICE=8
JP 140 GETA$:IFA$="T"THENDEVICE=1:GOTO160
XG 150 IFA$<>"D"THEN140
QG 160 PRINTA$:POKE780,15:POKE781,DEVICE:POKE782,2
55:SYS65466
PE 170 POKE780,LEN(PN$):POKE781,193:POKE782,2:SYS6
5469
XP 180 BA=49152:HI=INT(BA/256):LO=BA-(HI*256):POKE
251,LO:POKE252,HI
QQ 190 EA=49507:HI=INT(EA/256):LO=EA-(HI*256)+1:PO
KE780,251:POKE781,LO:POKE782,HI
MK 200 PRINT"{DOWN}SAVING ML VERSION OF ";PN$:SYS6
5496
CS 210 DATA 32,170,177,120,42,42,176,21,42,176,32
QD 220 DATA 42,176,42,42,176,87,42,176,54,42,176
MA 230 DATA 101,42,176,108,168,144,121,74,74,141
GM 240 DATA 94,193,140,93,193,238,97,193,56,176
QJ 250 DATA 105,169,234,141,21,3,169,49,141,20,3
KF 260 DATA 56,176,92,169,192,141,21,3,169,156,141
RX 270 DATA 20,3,169,26,141,90,193,56,176,74,152
SP 280 DATA 41,15,170,152,41,240,74,74,74,157
```

## SCREEN CONTROL

---

```
MF 290 DATA 42,193,157,74,193,170,152,41,15,157,26
DF 300 DATA 193,157,58,193,56,176,44,152,41,15,170
MK 310 DATA 168,189,42,193,170,152,153,74,193,138
GJ 320 DATA 157,58,193,56,176,22,152,41,15,170,188
PS 330 DATA 42,193,56,176,16,160,15,152,153,74,193
GF 340 DATA 153,58,193,136,16,246,230,254,160,1
QJ 350 DATA 169,0,88,108,5,0,173,97,193,208,10,206
JG 360 DATA 95,193,208,113,206,96,193,16,108,48,3
FM 370 DATA 206,97,193,173,94,193,141,96,193,173
PS 380 DATA 93,193,141,95,193,169,0,133,251,169
KR 390 DATA 219,133,252,169,26,77,90,193,141,90
HX 400 DATA 193,208,2,169,42,141,250,192,141,10
KP 410 DATA 193,173,90,193,208,23,165,254,240,19
JQ 420 DATA 198,254,160,15,185,58,193,153,26,193
CK 430 DATA 185,74,193,153,42,193,136,16,241,160,0
FR 440 DATA 177,251,41,15,170,189,26,193,145,251
RE 450 DATA 169,4,133,253,160,232,177,251,41,15,17
0
BG 460 DATA 189,26,193,145,251,136,208,243,198,252
MM 470 DATA 198,253,208,237,76,49,234,0,1,2,3,4,5,
6
AS 480 DATA 7,8,9,10,11,12,13,14,15,0,1,2,3,4,5,6,
7
JB 490 DATA 8,9,10,11,12,13,14,15,0,1,2,3,4,5,6,7,
8
CE 500 DATA 9,10,11,12,13,14,15,0,1,2,3,4,5,6,7,8,
9
FM 510 DATA 10,11,12,13,14,15,0,0,1,0,0,0,0,0,0
```

### Program 2. Blink Mode—Demo 1

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
BA 99 IFA=0THENA=1:LOAD"BLINKOBJ",8,1
GA 110 V1=785:V2=786:CO=646:HI=192
JA 130 POKEV1,0:POKEV2,HI
CC 140 REM
KX 150 SD=16384: REM ** SET DELAY FLAG
FF 160 DB=8192 : REM ** DISABLE BLINK
XG 170 EB=4096 : REM ** ENABLE BLINK
SM 180 SB=2048 : REM ** STOP COLOR FROM BLINKING
RD 190 MB=1024 : REM ** MAKE COLOR START BLINKING
EQ 200 VB=512{2 SPACES}: REM ** RETURN VALUE OF BL
INKING COLOR
RF 210 RC=256{2 SPACES}: REM ** RESET ALL COLORS-N
O BLINKING
PJ 220 REM *** NOW WE DEMONSTRATE BLINK
EC 230 RD$="-{BLK}**** RED ****{BLU}-":GR$="-{BLK}
*** GREEN ***{BLU}-":PRINT"{CLR}"
PG 240 POKE 53281,0:POKE 53280,0:REM MAKE BACKGROU
ND BLACK
```

## CHAPTER 2

```
PA 270 X=USR(EB):IF X=0 THEN530: REM ** ENABLE BLI
NKMODE
BB 280 C$=RD$
KF 290 GOSUB450
QF 300 X=USR(MB+(16*2)+0):REM ** BLACK BLINKS TO R
ED
PD 310 X=USR(MB+(16*3)+1):REM * WHITE BLINKS TO CY
AN
SR 320 X=USR(MB+(16*7)+4):REM ** MAKE PURPLE BLINK
TO YELLOW
HD 330 X=USR(SD+14): REM ** SET BLINK SPEED
RD 340 PRINT:PRINT"{PUR}{3 SPACES}*** WARNING ***
{SPACE}"
RK 350 PRINT:PRINT"{WHT}HIT A KEY TO CHANGE"
XG 360 GOSUB530
FP 370 X=USR(SB+0):REM **{2 SPACES}STOP BLACK FROM
BLINKING TO RED
DG 380 X=USR(VB+0):IF X<>0 THEN380:REM ** WAIT UNT
IL BLACK RESET
QR 390 X=USR(MB+(16*5)+0):REM ** NOW BLINK BLACK T
O GREEN
HM 400 C$=GR$:GOSUB450
GP 410 GOSUB530
AQ 420 X=USR(SB+0):REM **{2 SPACES}STOP BLACK FROM
BLINKING TO GREEN
XC 430 X=USR(VB+0):IF X<>0 THEN430:REM ** WAIT UNT
IL BLACK RESET
PB 440 GOTO280
CB 450 PRINT"{HOME}{WHT}ATTENTION - ATTENTION":PRI
NT
HC 460 PRINT"{BLK}{3 SPACES}CONDITION{2 SPACES}COD
E "
KE 470 PRINT"{BLU}{3 SPACES}U*****I "
RC 480 PRINT"{BLU}{3 SPACES}_-[13 SPACES]_"
RF 490 PRINT"{3 SPACES}"C$
AD 500 PRINT"{BLU}{3 SPACES}_-[13 SPACES]_"
DX 510 PRINT"{BLU}{3 SPACES}J*****K "
EG 520 RETURN
SM 530 TX=10:TY=20
CJ 540 GET A$:TY=TY-1:IF TY<0 THEN TY=20:X=USR(SD+
TX):TX=TX-1:IF TX=0 THEN TX=1
FG 550 CT$=" ":IF TX/2=INT(TX/2)THEN CT$="Q"
EQ 560 PRINT"{HOME}{16 DOWN}"
CG 570 X=USR(VB+0):REM ** NOW FIND THE COLOR BLACK
IS BLINKING TO
AR 580 POKECO,X:REM ** CHANGE CURRENT CHARACTER CO
LOR TO BLINKING COLOR
AR 590 PRINTTAB(TY);CT$;
KB 600 IF A$="" THEN540
QD 610 X=USR(SD+14):RETURN
```

## SCREEN CONTROL

---

### Program 3. Blink Mode—Demo 2

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
BA 99 IFA=0THENA=1:LOAD"BLINKOBJ",8,1
HF 110 NU=15:REM NU=7 FOR VIC
MB 120 V1=785:V2=786:CO=646:HI=192
QH 140 POKEV1,0:POKEV2,HI
DR 150 REM ** CHANGE THESE VALUES TO CHANGE THE BL
      INK COLORS
AJ 160 DIMCT(NU)
JF 170 FORI=0TONU:CT(I)=NU-I:NEXT
KQ 190 SD=16384: REM ** SET DELAY FLAG
MD 200 DB=8192 : REM ** DISABLE BLINK
QE 210 EB=4096 : REM ** ENABLE BLINK
PE 220 SB=2048 : REM ** STOP COLOR FROM BLINKING
ME 230 MB=1024 : REM ** MAKE COLOR START BLINKING
EK 240 VB=512{2 SPACES}: REM ** RETURN VALUE OF BL
      INKING COLOR
DP 250 RC=256{2 SPACES}: REM ** RESET ALL COLORS-N
      NO BLINKING
PP 260 REM *** NOW WE DEMONSTRATE BLINK
MF 270 X=USR(DB): REM ** MUST DISABLE BLINKMODE FI
      RST
GB 280 PRINT"{CLR}"
DD 290 POKE53280,0:POKE53281,0:REM MAKE BACKGROUND
      BLACK
KB 320 FOR L=0 TO NU: REM ** SET ALL THE COLORS
AB 330 CL=CT(L)
QE 340 X=USR(MB+(CL*16)+L): REM ** SET THE COLOR T
      O BLINK TO
KA 350 POKECO,L:REM ** CHANGE CURRENT COLOR TO X
RM 360 PRINT"BLINKING FROM"L"TO"CL
PX 370 NEXT L
JB 380 X=USR(SD+60): REM * SET BLINK FOR 1 SECOND
HM 390 PRINT:PRINT"HIT A KEY TO START"
BK 400 GETA$:IFA$="" THEN400
FB 410 X=USR(EB)
JX 420 END
```

# The Printmaker

---

---

*Manu Gambhir*

*This clever program converts a screen you design with keyboard graphics into a BASIC routine—and appends it to your program. For the Commodore 64 and 128 in 64 mode.*

Wouldn't it be nice if you could spend your time designing a screen and not have to worry about writing the program to produce it? "Printmaker" lets you do just that. It automatically creates code in the form of PRINT statements from whatever is on the screen and appends these lines to the program in memory. The PRINT statements include color control codes and REVERSE ON/OFF codes to reproduce the screen exactly as it was created.

Printmaker is very easy to use. It's written in machine language, but as a BASIC loader. There is only one rule to follow: The top line of the screen may not be used.

## **Designing a Screen**

Type in Printmaker; be sure to save a copy before running it the first time because the BASIC loader erases itself from memory. To use it, just load it and run. The program is POKed into a safe location (49152), out of the way of BASIC. Now you can begin writing your BASIC program, or you can load a BASIC program to which you wish to append your screen.

At this point, you're ready to create your design on the screen by using keyboard characters. All characters—numbers, letters, graphics—are legal. Colors are available, too. To move about the screen, use the cursor keys. (If you mistakenly hit the RETURN key, the computer will attempt to enter the current line as a BASIC statement.)

The entire screen, apart from the first line, will be encoded in PRINT statements. Since the last character position on the screen, the bottom right location, is included, the screen (and your display) will scroll up one line when you run the BASIC program. If you wish to avoid this effect, delete the last character (even if it's a space) in the final PRINT statement created by Printmaker. If your screen design calls for a



## SCREEN CONTROL

---

character in this position, it can be POKEd there in the BASIC program following the final PRINT statement.

When you've completed your design, press the CLR/HOME key to move the cursor to the upper left corner of the screen. Then type

**SYS 49152,I**

where *I* is the increment by which you want the lines numbered. Any number from 1 through 255 is allowed. Printmaker will append the new lines automatically to your program. For example, if your BASIC program ends with line 850, and you design a screen with Printmaker and SYS with an increment of 10, the appended code will begin with line 860 and proceed with 870, 880, and so on.

When you have entered the SYS, the cursor reappears and the screen (minus the top line) is appended to your program in the form of PRINT statements. Type LIST to see the results.

### Printmaker

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
PQ 100 FORI=49152TO49649:READA:X=X+A:POKEI,A:NEXT
MG 110 IFX<>55761THENPRINT"ERROR IN DATA STATEMENT
S. ":STOP
SD 120 NEW
KE 130 DATA 32,155,183,142,85,2,32,228,193,32
HK 140 DATA 228,193,32,228,193,32,140,193,160,193
RQ 150 DATA 169,109,32,30,171,169,0,141,84,2
HF 160 DATA 169,40,133,253,169,216,133,254,169,255
BM 170 DATA 141,88,2,169,40,133,251,169,4,133
PK 180 DATA 252,32,0,193,160,0,169,1,145,45
RF 190 DATA 32,0,193,145,45,32,0,193,165,20
GF 200 DATA 145,45,32,0,193,165,21,145,45,24
XJ 210 DATA 165,20,109,85,2,133,20,144,2,230
KJ 220 DATA 21,32,0,193,160,0,140,86,2,169
RH 230 DATA 153,145,45,32,0,193,169,34,145,45
XH 240 DATA 160,0,177,251,201,32,240,25,201,96
QC 250 DATA 240,21,177,253,41,15,205,88,2,240
RM 260 DATA 12,141,88,2,170,189,93,193,32,0
JJ 270 DATA 193,145,45,177,251,32,7,193,32,0
EH 280 DATA 193,145,45,230,251,208,2,230,252,230
SA 290 DATA 253,208,2,230,254,165,252,201,7,208
HQ 300 DATA 6,165,251,201,232,240,37,238,86,2
RD 310 DATA 173,86,2,201,25,208,179,32,0,193
JK 320 DATA 169,34,160,0,145,45,32,0,193,169
FD 330 DATA 59,145,45,32,0,193,169,0,168,145
```

## CHAPTER 2

---

---

SR 340 DATA 45,76,51,192,160,0,32,0,193,169  
SS 350 DATA 34,145,45,32,0,193,169,59,145,45  
BX 360 DATA 32,0,193,169,0,145,45,32,0,193  
AE 370 DATA 145,45,32,0,193,145,45,32,0,193  
PC 380 DATA 32,51,165,108,2,3,230,45,208,2  
GE 390 DATA 230,46,96,72,16,22,173,84,2,208  
QS 400 DATA 36,169,1,141,84,2,32,0,193,169  
RS 410 DATA 18,160,0,145,45,76,51,193,173,84  
DG 420 DATA 2,240,14,169,0,141,84,2,32,0  
EJ 430 DATA 193,169,146,160,0,145,45,104,41,127  
KE 440 DATA 201,34,208,3,169,39,96,201,32,176  
AR 450 DATA 4,24,105,64,96,201,64,176,1,96  
XH 460 DATA 201,96,176,4,24,105,32,96,201,128  
GE 470 DATA 176,4,24,105,64,96,169,63,96,144  
CM 480 DATA 5,28,159,156,30,31,158,129,149,150  
RS 490 DATA 151,152,153,154,155,19,18,80,82,73  
EP 500 DATA 78,84,77,65,75,69,82,146,32,32  
QF 510 DATA 32,32,32,13,0,234,234,234,234,234  
XM 520 DATA 234,234,234,234,234,234,160,0,177,43  
PG 530 DATA 208,15,200,177,43,208,10,173,85,2  
BM 540 DATA 133,20,169,0,133,21,96,165,43,133  
KF 550 DATA 251,165,44,133,252,160,0,177,251,133  
RQ 560 DATA 253,200,177,251,133,254,160,0,177,253  
GC 570 DATA 208,31,200,177,253,208,26,160,2,177  
KQ 580 DATA 251,133,20,200,177,251,133,21,24,165  
SF 590 DATA 20,109,85,2,133,20,165,21,105,0  
KH 600 DATA 133,21,96,165,253,133,251,165,254,133  
DF 610 DATA 252,76,169,193,56,165,45,233,1,133  
EJ 620 DATA 45,165,46,233,0,133,46,96

# Screen Customizer

---

---

*Christian Elfers*

*If you prefer certain screen colors and use various machine language programs during programming sessions, here's a way to keep the colors from being disabled. For the Commodore 64 and 128 in 64 mode.*

Many programmers have individual preferences for certain background and border colors and use them while programming. Unfortunately, whenever RUN/STOP-RESTORE is pressed, the computer resets these colors. Most of the time it's too much of a nuisance to rePOKE the values for these colors. Also, RUN/STOP-RESTORE disables some machine language programs, such as "The Automatic Proofreader." In order to reenale the program, the appropriate SYS must be entered. During a programming session, pressing RUN/STOP-RESTORE and repeatedly resetting screen colors and reenabling programs gets tiresome.

## **One-Key Default**

"Screen Customizer" will fix these problems. Type in the program and save it to tape or disk. From now on, before you begin a programming session, load and run Screen Customizer. Whenever you press the RESTORE key, the screen colors will be set to a black border, gray background, white cursor, and white characters. These colors can be changed by POKEing 1020 with the desired border color, 1021 with the screen color, 1022 with the cursor color, and 1023 with the character color. (Colors are numbered 0-15 and correspond to the colors available on the top row of the keyboard.) You can turn off Screen Customizer by typing **POKE32776,0**. To turn it back on, type **POKE32776,48**.

Screen Customizer also prevents most machine language programs from becoming disabled. Note to tape users: You must disable the Automatic Proofreader before a tape save by typing **SYS8**; typing RUN/STOP-RESTORE will no longer disable the Proofreader. Location 8 is used because it almost always contains a zero. A SYS to any location containing a

zero resets the computer like RUN/STOP-RESTORE without Screen Customizer activated.

Remember that every time you load in a new machine language program, such as the Proofreader or the DOS Wedge, you must tell Screen Customizer you did so by typing **SYS 32785**. Otherwise, the program will be disabled when RUN/STOP-RESTORE is pressed. This is necessary only for programs that are normally disabled by RUN/STOP-RESTORE.

### How It Works

Here's how the program works. It takes advantage of the autostart feature normally used for cartridges. When the computer is first turned on, it checks for a cartridge by looking for specific data at locations 32772-32776. If these locations contain the ASCII values of the characters *CBM80*, the computer will jump to whatever address is pointed to by locations 32768-32769 in low-byte/high-byte format. Locations 32770-32771 must also contain the same address. The computer also checks for this combination during a warm start (SYS64738) or when you press RESTORE. I changed these locations so that when RESTORE is pressed the computer jumps to Screen Customizer.

When Screen Customizer is activated for the first time, it transfers the values in locations 768-819 into a temporary storage area. These locations contain important vectors. They tell the computer what to do every 1/60 second, or when RETURN is pressed, or when the commands CLOSE, LIST, LOAD, OPEN, and SAVE are entered.

Some machine language programs change the values in these locations to point to a customized routine. Normally, whenever RUN/STOP-RESTORE is pressed, these locations are reset to their normal values, thus disabling the customized routines. But Screen Customizer transfers the values from the temporary storage area back to locations 768-819, thus preventing most machine language programs from being disabled. The program then sets the screen colors and returns to BASIC.

In addition, since the RESTORE key causes a non-maskable interrupt (NMI), pressing RESTORE with Screen Customizer activated enables you to recover from keyboard lock-up bugs without having to turn the computer off and on to regain control.

## SCREEN CONTROL

---

### Screen Customizer

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
ME 10 POKE56,128:CLR
XA 20 FORG=32768TO32876:READA:C=C+A:POKEG,A:NEXT
EP 30 IFC<>13765THENPRINT"{CLR}TYPING ERROR IN LIN
ES 100-150.":STOP
PM 40 A$="{8 SPACES}":PRINT"{CLR}{DOWN}{6 SPACES}P
RESS {RVS}RESTORE{OFF} TO ACTIVATE."
EC 50 PRINT"{2 DOWN}"A$"POKE1020,<BORDER COLOR>"
XS 60 PRINT A$"POKE1021,<SCREEN COLOR>"
RB 70 PRINT A$"POKE1022,<CURSOR COLOR>"
GE 90 PRINT"{2 DOWN}TYPE {RVS}SYS 32785{OFF} TO RE
AD NEW VECTOR VALUES"
RB 100 DATA 11,128,11,128,195,194,205,56,48,161,12
8,32,17,128,76,29,128,160
GC 110 DATA 204,185,52,2,153,128,128,200,208,247,9
6,169,0,141,252,3,169,11
RD 120 DATA 141,253,3,169,1,141,254,3,141,255,3,16
9,58,141,0,128,141,2,128
HM 130 DATA 108,20,3,32,129,255,32,138,255,160,204
,185,128,128,153,52,2,200
RJ 140 DATA 208,247,173,255,3,141,33,208,173,252,3
,141,32,208,173,254,3,141
XD 150 DATA 134,2,169,147,32,210,255,173,253,3,141
,33,208,162,255,76,139,227
```

# QuickScan

---

---

*Daan Deenik*

*If your eyes get tired while you're checking long listings on the screen, this program will be a real boon. It highlights the current screen line, making it easier for you to keep your place while scanning the program. For the Commodore 64 and 128 in 64 mode.*

Everyone who has written a program or typed one in from a magazine or book knows the sinking feeling you get when you realize you've made a mistake and you'll have to go back and check your work. Programs that contain long lists of DATA statements are especially annoying; it's easy to accidentally check a line twice or miss a line here or there.

Have you ever wanted a ruler that would automatically move up and down the screen? "QuickScan" is just that, a bar that highlights screen lines. Just use the cursor keys to control the highlighter's location.

## Using the Highlighter

The instructions aren't complicated. Type in QuickScan and save it to tape or disk. When you run it, a short machine language program is POKed into memory, and a message (describing how to start it) is printed on the screen. To enable it, type **SYS49152**. The machine language program is loaded into RAM by a BASIC loader. Although there is a built-in checksum to help in entering the program, accurate typing is still required since any mistake can crash the computer.

## Modifying QuickScan

QuickScan uses seven multicolor sprites to create the highlighting bar. Here are a few ideas for modifications. If you change the 0's in lines 1001 and 1003 to 255's, and run the program again, you'll see a bar three lines high. The upper and lower parts of the bar are the same color as the characters on the screen, so you won't be able to see them. But the middle part is visible.

## SCREEN CONTROL

---

---

You can change the color of the middle part by POKEing 53285 with a number from 0 through 15. You can split the bar by giving the first four sprites low priority and the other three high priority (POKE 53275,15). This might come in handy with question-and-answer programs.

### QuickScan

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
BQ 100 PRINTCHR$(147);CHR$(144)
AF 110 FORAD=704TO766:READDA:POKEAD,DA:NEXT: REM S
    PRITE
EX 120 FORAD=49152TO49251:READDA:POKEAD,DA:
    {4 SPACES}CS=CS+DA:NEXT:REM M-L PROGRAM
GQ 130 IFCS<>11879THENPRINT"ERROR IN DATA STATEMEN
    TS":END
DB 140 PRINT"ENTER SYS 49152 TO ACTIVATE SPRITE-BA
    R"
XG 1000 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
FS 1001 DATA0,0,0,0,0,85,85,85,85,85,85,85,85,85,85
HS 1002 DATA85,85,85,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
ES 1003 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
MG 2000 DATA11,160,7,153,247,7,136,208,250,169,24,
    160,0,24,153,0,208,105,48
HJ 2001 DATA200,200,144,247,169,96,141,16,208,169,
    8,141,10,208,169,56,141,12
QC 2002 DATA208,169,127,141,28,208,141,27,208,141,
    29,208,141,23,208,141,21
PM 2003 DATA208,169,1,141,37,208,120,169,74,160,19
    2,141,20,3,140,21,3,88,96
QP 2004 DATA24,173,134,2,141,38,208,165,214,10,10,
    10,105,34,160,14,153,255
JG 2005 DATA207,136,136,208,249,76,49,234
```

# Quick Character Transfer

---

---

*Fabio Coronel*

*Setting up a custom character set can be painfully slow. This machine language routine will greatly speed up the process. For the Commodore 64 and 128 in 64 mode.*

Well-designed graphics add a lot to almost any program. Sometimes pictures can communicate better than words. The easiest type of graphics to use is character graphics, those odd shapes shown on the front of the keys. Another type of graphics is achieved with redefined characters which allow you to create shapes in much greater detail. But they're also more difficult to use.

First, you must reserve space for the new character set in RAM by lowering the top-of-BASIC pointer. Next, you change the character-set pointer to point to the location of the new characters. You then transfer the character patterns from ROM to RAM and change selected characters to their new shapes.

The major drawback is the time it takes to transfer the character patterns. It may take BASIC almost a minute to PEEK the bytes from ROM and POKE them to RAM, depending on the number of characters transferred.

"Quick Character Transfer" creates a machine language routine to transfer the character patterns from ROM to RAM instantly. The ML routine is POKEd into the cassette buffer, but it's completely relocatable. You can put it elsewhere by setting variable AD in line 100 to the start of the new location.

## **Adding It to Your Programs**

To use Quick Character Transfer, you must add it to your program. You can place it at the end as a subroutine or at the start as part of the initialization, as long as it's executed before you redefine your characters. You may have to change the line numbers to make it fit.



## SCREEN CONTROL

---

Quick Character Transfer can also be used by itself as a demonstration. Type in the program and save it before running. The characters on the screen will momentarily appear as a random pattern of dots as the pointer to the start of the character set is changed to point to the random bytes in RAM. Then the characters quickly return to normal as the patterns in ROM are transferred to RAM.

Now you can change any character to look like a spaceship or a bird or a foreign language character. Just POKE the character pattern into the RAM area used by the new character set. To show that the characters are indeed in RAM, lines 1000-1010 change the @ character into a happy face. Type the @ key to see it. To return to the normal character set, press RUN/STOP-RESTORE.

The program transfers the entire uppercase/graphics character set (256 characters) from ROM to RAM. You can transfer the uppercase/lowercase character set instead by changing the 208 to 216 in line 170, thus altering the checksum in line 120. The location of the character set in RAM is determined by the two 14's in line 90. these values represent the number of kilobytes from the start of the video bank.

When you're selecting a place for the character set, remember that it must be placed above your BASIC program on a 2K boundary in the same video bank as the screen. If you were frightened by that last sentence, just leave the values at 14. This puts the start of the character set at 14336, leaving 12K free for your BASIC program.

### Quick Character Transfer

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
MR 90 POKE56,14*4:CLR:POKE53272,(PEEK(53272)AND240
)OR14
XB 100 AD=828:REM STARTING ADDRESS
QB 110 FORI=ADTOAD+81:READA:X=X+A:POKEI,A:NEXTI
SM 120 IFX<>9923THENPRINT"ERROR IN DATA STATEMENTS
.":STOP
BJ 130 SYSAD
SE 140 DATA 173,14,220,41,254,141,14
XP 150 DATA 220,173,24,208,41,14,10
MX 160 DATA 10,133,167,169
XB 170 DATA 208:REM CHANGE TO 216 TO MOVE LOWER CA
SE
QP 180 DATA 133,252,173,0,221,41,3,73,3
DK 190 DATA 10,10,10,10,10,10,5
```

## CHAPTER 2

---

AG 200 DATA 167,133,254,165,1,41,251  
PS 210 DATA 133,1,169,0,133,251,133  
QM 220 DATA 253,168,162,8,177,251,145  
HH 230 DATA 253,200,208,249,230,252,230  
FC 240 DATA 254,202,208,242,165,1,9  
GG 250 DATA 4,133,1,173,14,220,9  
MK 260 DATA 1,141,14,220,96  
KS 1000 FORI=14336TO14343:READA:POKEI,A:NEXT  
QG 1010 DATA{2 SPACES}60,66,165,129,165,153,66,60

# Color Swap

---

*Lee Noel, Jr.*

*Frustrated by the time and trouble involved in making color changes to high-resolution program displays? "Color Swap" makes them instantly, and it works for the other graphics modes of your computer, too. For the Commodore 64 and 128 in 64 mode.*

"Color Swap" is a machine language graphics aid for your BASIC programs. Operating at high speed, it enables you to change colors at will—without LISTING and editing your program or without waiting for the screen to rebuild. It works with any type of display—text, hi-res, multicolor, sprites, and so on—and automatically adjusts itself to the screen configuration you've chosen.

To start, type in the program. Be sure to save a copy of the program before you run it. When you run the program, if you have a correct version, you'll see a message with program instructions. If this is the first time you've used a machine language program, the SYS command in the final screen message may be unfamiliar to you. SYS XXXXX, a BASIC statement that can be used in either direct or program mode, transfers control from the BASIC environment to the machine language program at address XXXXX.

Note where it tells you to SYS; then type NEW and press RETURN. Although you have just erased the program from BASIC memory, it resides in a safe location (see below for details). You can load, save, and NEW lots of programs, but Color Swap remains ready to be activated at any time by the SYS call. The program can be erased only by turning off your computer or by putting something else into its memory area.

## **A Pair of POKEs**

Once in place, the program is simple to use. It creates two new (pseudo) registers at addresses 700 and 701. Think of 700 as the Old Color Register, and 701 as the New Color Register. If you POKE these locations with two different color codes and SYS to Color Swap, any displayed color that matches the

## CHAPTER 2

---

value in 700 will be changed to the color indicated at 701.

To make this clearer, here's an example. After loading and running Color Swap, load the program you want to experiment with and bring up the desired screen display. Working in direct mode, you might enter

```
POKE700,6:POKE701,7:SYS XXXXX
```

where XXXXX is the Color Swap address.

If you enter this line and press RETURN, the program will find your display, look for blue (color code 6), and alter any blue it finds to yellow (code 7). This could, for example, change an entire blue sky to yellow or cause the same transformation in a tiny redefined character. And the exchange of color takes place immediately—there's no waiting. If you don't like the effect, reverse the codes and SYS to Color Swap again.

As you go, make notes of the color codes that give the best results. Later on you'll want to plug these tested values into the program you're working with. Color Swap acts directly on your computer, *not* on your programs, so you'll need either notes or a perfect memory.

Until you become thoroughly familiar with Color Swap, there are a number of points to keep in mind. The program is designed to change *every* occurrence of a particular color—which can be disconcerting at times. If you alter a blue sky and the current text color is also blue, both will be changed accordingly. (For a way around this problem, read on.) Moreover, it can be difficult to use Color Swap with extended background color. The program works fine, but you must know exactly what you're doing. Finally, for safety, stick to the standard Commodore color code numbers, 0–15. Note that Color Swap will not generally work in conjunction with other machine language programs, including utility cartridges and BASIC extensions.

Color Swap is most effective with displays and programs that are fairly satisfactory apart from their colors. If you're in the early stages of developing a program and are using Color Swap concurrently, you'll need to take extra care to keep track of which program is supplying color codes.

A little practice will make things clear. You can even start without a program by experimenting with the colors of your computer's standard display. If you ever do run into trouble, just reload your program and start over.

## SCREEN CONTROL

---

### **One-Way Trip**

Blocks of machine language work like subroutines in BASIC. Color Swap is just simple chunks of machine language subroutines linked together in a particular sequence.

In this case, the arrangement is like towns (the subroutines) connected by a one-way street. If you want to avoid the town that changes text color, just get on the road at the next town. (With Color Swap, once you get on the one-way street, you're carried along to the very end—where control is returned to BASIC.)

The accompanying diagrams show how the program is laid out. The first few sections are changes that can be made from BASIC, usually by a POKE to one memory location. You can skip over some of these early portions, thus avoiding unwanted color changes.

To make such a detour, just SYS to the address of the first needed town along the Color Swap route, having first POKED 700 and 701 with the desired values, of course. This multiple-entry-points feature of Color Swap is obviously useful, and you may enjoy experimenting with it.

### **Relocating the Program**

Color Swap is relocatable. If the current location is inconvenient, the bulk of the program can be shifted anywhere with free memory. Line 120 of the program contains the variable RA (Relocatable Address), which can be set to any desired value. However, if Color Swap is moved into the normal program area, you'll need to take steps to prevent it from being overwritten by BASIC.

Remember that in any relocation the target addresses of all the SYS calls have to be changed as well. The parenthetical expressions in the diagram show you how to do this. (RA is the value of the number that appears after SYS in the final message from your loader. If you move RA outside the values described above, make provisions to protect the program.)

### **Extending Its Usefulness**

After a little experimentation with Color Swap, you'll find many uses for it. Although it was devised mainly as a direct-mode method of editing displayed colors, it can be extended to many other applications.

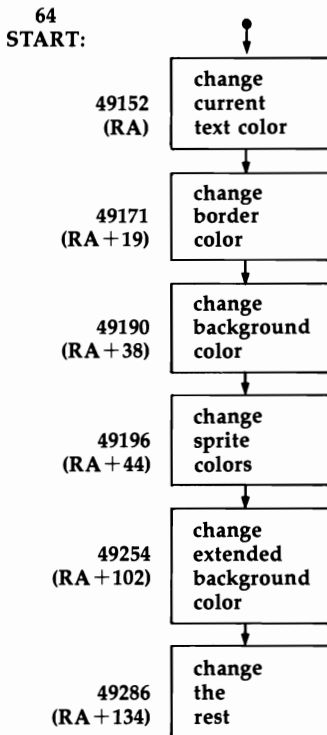
## CHAPTER 2

For example, Commodore users know that you can't POKE character codes directly to screen memory and see any effect unless you make corresponding POKES to color memory. As long as you know what code is in color memory, you can easily change it to contrast with the background by a SYS to Color Swap. With Color Swap in your computer and the default display on screen, enter this line in direct mode:

**POKE700,6:POKE701,1:SYS49286:POKE1524,83**

You should see a white heart appear near the center of your screen. POKE a few more character codes directly to screen memory. Convenient, isn't it?

### Color Swap Locations



To swap colors:

POKE 700, old color: POKE 701, new color: SYS 49152. SYS to one of the later addresses shown above to avoid unwanted changes.

Note: If Color Swap is relocated, use the expressions given above to work out the new addresses—including RA, the new starting address.

## SCREEN CONTROL

---

You can also add Color Swap to programs as a subroutine. Although doing this may take a little thought, it can be worthwhile. As part of a larger program, Color Swap's speed enables it to flash windows of text and background on and off, make displays appear and disappear, and alter characters and sprites instantly.

### Color Swap

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
MS 10 REM *COLOR-SWAP 64*{3 SPACES}N IS ML CODE, A
    IS ADDRESS, AND I DOES THE COUNTING.
CA 20 N=74:FORA=685TO690STEP5:FORI=0TO3:POKEA+I,N:
    NEXTI:POKEA+I,96:N=N-64:NEXTA
MB 30 CK=0:FORI=0TO9:X=PEEK(685+I):CK=CK+X:NEXT
FB 40 REM CK IS CHECKSUM AND X ONE OF ITS COMPONENTS.
JX 50 IFCK<>528THENPRINT"{CLR} {DOWN}CHECK LINE 20
    FOR ERRORS." :END
KM 100 CK=0:FORA=702TO750:READN:POKEA,N:CK=CK+N:NEX
    XTA
AP 110 IFCK<>5158THENPRINT"{CLR} {DOWN}CHECK LINES
    702-750 FOR ERRORS." :END
QF 120 RA=49152:RA$=STR$(RA):L=LEN(RA$):RA$=RIGHT$(
    RA$,L-1):REM RA IS RELOC. ADR.
CC 130 CK=0:FORA=RATORA+392:READN:POKEA,N:CK=CK+N:
    NEXTA
CD 140 IFCK<>46713THENPRINT"{CLR} {DOWN}CHECK LINE
    S 49152-49542 FOR ERRORS." :END
CG 150 PRINT"{CLR} {DOWN}NOW POKE700,OC:POKE701,NC
    :SYS"RA$" TO{3 SPACES}SWAP COLORS"
JD 160 PRINT:PRINT"{2 SPACES}WHERE OC IS THE COLOR
    CODE FOR THE OLD COLOR,"
BG 170 PRINT"{2 SPACES}AND NC IS THE CODE FOR THE
    {SPACE}NEW COLOR." :END
XC 702 DATA 216,173,188,2,41,15
DG 708 DATA 141,188,2,173,189,2
GH 714 DATA 41,15,141,189,2,169
AX 720 DATA 0,141,172,2,141,171
SG 726 DATA 2,141,169,2,141,170
MR 732 DATA 2,96,173,33,208,41
HK 738 DATA 15,205,188,2,208,6
QP 744 DATA 173,189,2,141,33,208
QS 750 DATA 96
FX 49152 DATA 32,190,2,173,134,2
ME 49158 DATA 41,15,205,188,2,208
SE 49164 DATA 9,173,189,2,141,134
JS 49170 DATA 2,32,190,2,173,32
```

## CHAPTER 2

---

SF 49176 DATA 208,41,15,205,188,2  
BM 49182 DATA 208,9,173,189,2,141  
JD 49188 DATA 32,208,32,190,2,32  
DF 49194 DATA 222,2,32,190,2,173  
BH 49200 DATA 37,208,41,15,205,188  
MC 49206 DATA 2,208,6,173,189,2  
KG 49212 DATA 141,37,208,173,38,208  
PK 49218 DATA 41,15,205,188,2,208  
HH 49224 DATA 6,173,189,2,141,38  
EG 49230 DATA 208,160,0,185,39,208  
PJ 49236 DATA 41,15,205,188,2,208  
BS 49242 DATA 6,173,189,2,153,39  
XF 49248 DATA 208,200,192,8,208,235  
BJ 49254 DATA 32,190,2,173,17,208  
MG 49260 DATA 41,64,240,22,32,222  
FX 49266 DATA 2,173,36,208,41,15  
KB 49272 DATA 205,188,2,208,6,173  
BQ 49278 DATA 189,2,141,36,208,24  
EG 49284 DATA 144,7,173,22,208,41  
XP 49290 DATA 16,240,37,169,255,141  
KG 49296 DATA 170,2,173,34,208,41  
EG 49302 DATA 15,205,188,2,208,6  
HJ 49308 DATA 173,189,2,141,34,208  
XS 49314 DATA 173,35,208,41,15,205  
FA 49320 DATA 188,2,208,6,173,189  
JX 49326 DATA 2,141,35,208,173,17  
KF 49332 DATA 208,41,32,208,5,169  
MJ 49338 DATA 255,141,169,2,173,0  
MF 49344 DATA 221,41,3,201,3,208  
FX 49350 DATA 4,162,0,134,254,201  
QJ 49356 DATA 2,208,4,162,64,134  
RC 49362 DATA 254,201,1,208,4,162  
BP 49368 DATA 128,134,254,201,0,208  
JS 49374 DATA 4,162,192,134,254,173  
JD 49380 DATA 24,208,41,240,32,173  
RQ 49386 DATA 2,32,180,2,101,254  
SK 49392 DATA 133,254,169,0,133,253  
AD 49398 DATA 169,0,133,251,169,216  
KF 49404 DATA 133,252,162,0,142,167  
GS 49410 DATA 2,162,0,142,168,2  
CB 49416 DATA 162,234,142,183,2,160  
XJ 49422 DATA 0,177,251,41,15,205  
MJ 49428 DATA 188,2,208,5,173,189  
AK 49434 DATA 2,145,251,173,169,2  
KH 49440 DATA 201,255,208,2,240,55  
EG 49446 DATA 177,253,41,15,141,171  
FQ 49452 DATA 2,205,188,2,208,6  
MA 49458 DATA 173,189,2,141,171,2  
RF 49464 DATA 177,253,41,240,141,172  
RD 49470 DATA 2,32,173,2,205,188



## SCREEN CONTROL

---

---

SG 49476 DATA 2,208,13,173,189,2  
AK 49482 DATA 32,178,2,24,109,171  
MF 49488 DATA 2,24,144,7,173,172  
MQ 49494 DATA 2,24,109,171,2,145  
AS 49500 DATA 253,200,204,168,2,240  
PG 49506 DATA 2,208,170,174,183,2  
PP 49512 DATA 224,96,208,1,96,238  
GJ 49518 DATA 167,2,230,252,230,254  
JF 49524 DATA 162,3,236,167,2,240  
FG 49530 DATA 2,208,144,162,232,142  
DA 49536 DATA 168,2,162,96,142,183  
RX 49542 DATA 2,208,132

# Time Clock

---

---

*David W. Martin*

*Put a digital clock on your computer screen with this machine language program. For the Commodore 64 and 128 in 64 mode.*

There's a clock inside your computer. It starts ticking immediately when you flip the on switch and continues until you turn your computer off.

It's called the jiffy clock, a three-byte section of memory that ticks every 1/60 second (jiffy). You can read the time in jiffies with `PRINT TI`, or you can find how many seconds it's been since you turned on the computer with `PRINT TI/60`. To get a more readable time, `PRINT TI$` gives you hours, minutes, and seconds. (For example, 131500 is 13 hours, 15 minutes, 0 seconds, or a quarter past one in the afternoon.)

`TI` and `TI$` are called reserved variables. They are reserved for timekeeping only; you can't use them in your programs unless you're checking the time. To set the clock, you can define `TI$`, using the *HHMMSS* (Hours, Minutes, Seconds) format inside quotation marks. For example, enter `TI$="063000"` to set the clock to 6:30 a.m. You can't set `TI` directly; you have to set `TI$`, which affects both time variables.

## **The Stopwatch Function**

The jiffy clock sometimes comes in handy. When you're running benchmark tests, you can set the clock to 000000 just before running the routine being checked. When the routine is finished, `PRINT TI` to see how much time the program took to run. In this way, you find the fastest ways of doing such things as alphabetizing. It's like using a stopwatch on a programming technique.

You can also use the clock as an alternative to `FOR-NEXT` delay loops. Define a variable as `TI+60` (for a one-second delay) and keep looping around until `TI` is greater than or equal to the variable.

If you need to keep time in a program like a racing game or a touch-typing program, you simply read the jiffy clock.

## SCREEN CONTROL

---

Commodore owners have two time-of-day (TOD) clocks in addition to the jiffy clock. They're built into one of the interface chips. The TOD clocks count in tenths of seconds rather than sixtieths.

### A Time-Display Window

You could use a one-line program to continually display the time, for example,

```
1 PRINT"{CLR/HOME}";TI$:GOTO1
```

except for one problem. To keep the time updated, the program would have to be running all the time, which would mean you couldn't use the computer for anything else.

"Time Clock" eliminates this problem. It prints the time in the upper right corner of the screen and leaves the computer available for other tasks. A machine language interrupt drives the program. In other words, it runs in the period when the operating system does its housekeeping (like updating the screen).

To use Time Clock, type it in and save it before running. If the internal checksum does not match up, check the DATA statements. (Remember to save your corrected version.)

Next, you'll be asked to set the clock. The Commodore uses a 12-hour clock, with a.m. and p.m. designations. It wraps around and resets at midnight.

After the clock has been set, the time will appear in the upper corner. You can press RUN/STOP and type NEW without affecting the clock. RUN/STOP-RESTORE will erase the clock. To regain it, you'll have to type SYS49171.

Since the program runs independently from the jiffy clock, you can use TI and TI\$ as clocks that are separate from the time displayed on the screen. Because the jiffy clock does not keep correct time when information is being written to or read from the cassette, Time Clock reads a TOD clock, which is not affected by tape.

### Time Clock

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
GE 100 Y=PEEK(49269):IFY<>141THENGOSUB270
BK 110 POKE56335,0:PRINT"{CLR}{WHT}":INPUT"AM OR P
      M ({RVS}A{OFF}/{RVS}P{OFF})";A$
RC 120 IF A$<>"A" AND A$<>"P" THEN 110
```

## CHAPTER 2

---

```
SG 130 B=0:IF A$="P" THEN B=128
RA 140 INPUT "{3 DOWN}TIME (HHMMSS FORMAT)";T$
KD 150 IF LEN(T$)<>6 THEN 140
QG 160 IF VAL(LEFT$(T$,2))>12 THEN 250
FB 170 IF VAL(MID$(T$,3,2))>59 OR VAL(MID$(T$,5,2)
)>59 THEN 250
XG 180 FORA=1TO5STEP2:D=VAL(MID$(T$,A,1)):D=D*16+V
AL(MID$(T$,A+1,1))
CJ 190 POKE49154-(A-1)/2,D:NEXTA
PJ 200 IF LEFT$(T$,2)="12" THEN B=128-B
RP 210 POKE49154,PEEK(49154)AND127:POKE49154,PEEK(
49154)ORB
KD 220 PRINT "{3 DOWN}{6 SPACES}PRESS ANY KEY TO ST
ART CLOCK"
DA 230 GET A$:IF A$="" THEN 230
DF 240 PRINT "{CLR}":SYS49155:END
FG 250 PRINT "{DOWN}ERROR IN INPUT.":FORI=1TO1000:N
EXT:GOTO140
JQ 260 REM ML LOADER
RR 270 I=49155
FF 280 READ A:IF A=256 THEN 300
QF 290 POKE I,A:I=I+1:X=X+A:GOTO 280
QR 300 IF X<>13794 THEN PRINT"ERROR IN DATA STATEM
ENTS.":END
CH 310 DATA 162,3,189,255
SC 320 DATA 191,157,8,220,202,208,247
KS 330 DATA 169,0,141,8,220,120,169
JK 340 DATA 32,141,20,3,169,192,141
XF 350 DATA 21,3,88,96,169,58,141
MR 360 DATA 29,4,173,134,2,141,29
KM 370 DATA 216,162,3,160,0,189,8
KF 380 DATA 220,41,112,74,74,74,74
EM 390 DATA 24,105,176,153,30,4,173
FG 400 DATA 134,2,153,30,216,200,189
SG 410 DATA 8,220,41,15,24,105,176
EJ 420 DATA 153,30,4,173,134,2,153
RK 430 DATA 30,216,200,202,208,213,173
RD 440 DATA 8,220,24,105,176,141,37
AA 450 DATA 4,173,134,2,141,37,216
DM 460 DATA 173,11,220,48,5,169,1
DQ 470 DATA 76,117,192,169,16,141,39
MX 480 DATA 4,173,134,2,141,39,216
KC 490 DATA 76,49,234,0,256
QF 500 RETURN
```

# Slowpoke

---

---

*Daniel R. Widyono*

*This very short machine language program, from a 12-year-old author and programmer, allows you to control the speed of the PRINT statement. We've added some suggestions for using this unique routine. For the Commodore 64 and 128 in 64 mode.*

When a 64 or 128 is printing something or listing a program, you can slow it down by pressing the CTRL key. You have a choice of two speeds, regular and slow. If you've ever used an Apple II, you probably know that it gives you a wider choice; it has 256 printing speeds.

Can a similar feature be added to the 64 and 128? You could use a BASIC subroutine, although it would be rather complicated. You would have to define a string, do a FOR-NEXT loop for the length of the string, use MID\$ to pull out a character, print the character, use another FOR-NEXT loop for the delay, and then continue until the message is done. The subroutine would work only as part of a program. You'd still have to press CTRL to slow down program listings.

## **A Better Way**

There's a short, simple, and effective machine language alternative to the BASIC subroutine. Enter and save "Slowpoke."

Now run the program and type LIST. It lists as you would expect. Enter

**POKE 251,255:LIST**

and you'll see a very slow listing. The program takes more than a minute to list (compared to less than a second at the regular speed). POKE a zero into 251, and the computer returns to normal.

You now have 256 different printing speeds, from 0, the fastest, to 255, the slowest. There's no need to use a BASIC subroutine or to SYS. Just POKE the speed you want into memory location 251. It works within programs and also in immediate mode. It even slows down error messages and the

READY prompt. If you use a printer while "Slowpoke" is on, it will wait between lines, but not between characters. You'll notice, however, that your typing is not affected; the computer still reads the keyboard at the regular speed.

Pressing RUN/STOP-RESTORE disables Slowpoke. To get it back, you'll have to run the loader program again or type **POKE 806,167: POKE 807,2**

If you use these POKes, make sure you put them on the same line, separated by a colon.

### How It Works

I was looking through a memory map and discovered a vector at 806-807 that points to the character-out (CHROUT) routine in the operating system. The Kernal CHROUT vector in ROM (which cannot be POKed) uses this vector in RAM (where it can be changed) to find the routine that sends a character to screen memory, to a printer, or to a tape or disk file.

When the computer gets an instruction to PRINT, it checks the Kernal vector, goes to this vector, and finally ends up at the instructions for printing. The computer has to look up the address for every character it prints, even if the characters are part of a long string. The address is stored in the usual low-byte/high-byte format.

Vectors are like highway signs, pointing the way to a destination (in this case, the CHROUT routine). By changing the numbers in memory locations 806 and 807, you can change the route, adding a slight detour, a machine language delay loop. When the computer tries to PRINT or LIST, the vector at 806-807 sends it to the delay loop, which is followed immediately by a jump to the usual CHROUT part of the operating system.

Since machine language is so fast, you need two delay loops, one inside the other. First, the X register is loaded from location 251. If 251 holds a zero, the rest of the loop is skipped (using BEQ, Branch if Equal to zero). Then the Y register is loaded with the number 255. Y is decremented (DEY) until it becomes zero. Next, X is decremented and a BNE instruction (Branch if Not Equal) loops back to the DEY loop. The higher the number in 251, the longer the loops take. At the slowest printing speed, X has to be decremented 255 times, while Y is decremented  $255 \times 255$  times (over 65,000 times).

## SCREEN CONTROL

---

The values of X and Y are always zero after the loop has been done. So, to be safe, the values of the A, X, and Y registers, as well as the processor status, are saved. They are pushed onto the stack before the routine is executed and pulled off after the delay loop.

The delay loop occupies a small area of memory at locations 679–767. If you need this RAM for other machine language programs, Slowpoke is relocatable. You can move it to the cassette buffer, beginning at 828, or other free sections of memory. If you have a program which uses zero-page location 251, change the 251 in line 30 to another available zero-page location.

### Practical Ideas

One useful application of Slowpoke is to slow down program listings. Load and run Slowpoke; then load the program you're working on. You can control the speed of the listing with a single POKE.

If you are working on a game and the rules take up a couple of screens, you can use Slowpoke to make them scroll up slowly, rather than the usual method of printing a screen and waiting for the user to press a key to get the next page. This same idea could be adapted to a story program: At the end of the story, roll the credits (just like at the movies). Many programs that put a lot on the screen could use a printing speed control.

The program is a good debugging tool if you are having problems getting the screen graphics just right. You should note, however, that only the PRINT statement is affected; POKEing to screen memory is no slower than usual.

Slowpoke can be very useful with a program that examines tape or disk files. You would open the file, read an item, print it to the screen, and then use a GET statement to see whether a key has been pressed. If not, then continue with the next item. If f1 has been pressed, speed up the printing; if f7 has been pressed, make it slower. You might even adapt it to a speed-reading program.

Finally, if you're familiar with machine language programming, you could make a slight modification to add sound effects. Just before the delay loop, turn on the noise generator and turn it off at the end of the loop. Each time you print

## CHAPTER 2

---

something, you will hear, say, the clicking of a typewriter. If you want to get really fancy, you can add a bell sound at the end of each line.

### Slowpoke

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
AX 10 CK=0:POKE251,0
FK 20 FORJ=679TO703:READA:CK=CK+A:POKEJ,A:NEXT
HD 25 IFCK<>3615THENPRINT"CHECK DATA STATEMENTS":E
    ND
PS 30 DATA72,138,72,152,72,8,166,251,240,8,160,255
    ,136,208,253,202,208,248,40
GG 40 DATA104,168,104,170,104,76
HD 50 IFPEEK(807)<>2THENPOKE704,PEEK(806):POKE705,
    PEEK(807)
GE 60 POKE806,167:POKE807,2
```



# Hi-Res Screen Dump

---

---

*Gregg Peele*

*Have you ever created a hi-res picture or graph and then tried to reproduce it on your printer? This program allows you to do just that. Compatible with the Commodore 1525 or MPS-801 printer (but not the 1526). For the Commodore 64 and 128 in 64 mode.*

The Commodore 64 and 128 allow you to create high-resolution graphics images on the video screen. With the 64 Super Expander cartridge or another hi-res program, it's easy to produce detailed artistic creations. However, most of these programs don't provide a method of printing out these artistic endeavors once you've finished them. Unless you leave your computer turned on indefinitely, your creation is short-lived.

"Hi-Res Screen Dump" works with a Commodore 1525 or compatible printer. (Note that the 1526 printer from Commodore is *not* compatible with the 1525 and will not work with this program.)

## **Bit Transfer**

Hi-Res Screen Dump is designed to transfer the bit information from screen memory to the printer. Since the 1525 printer can accept only seven bits of data at a time in graphics mode (the high bit must always be set), the eight-bit bytes in screen memory must be split into odd units before they are sent to the printer. Transferring the information from screen to printer is further complicated since the location of screen memory bytes must also be calculated, and hi-res screens for the 64 can be moved to several different areas of memory.

This program reads data from the screen one bit at a time starting from the lower left corner of the screen. After seven bits, the program moves to the leftmost bit of the next row up and prints seven more bits, continuing up the screen. After the leftmost seven-bit column has been printed, the program starts at the eighth bit over from the bottom left corner and continues cycling from bottom to top until the entire screen has been read. Each seven bits are combined to form the byte sent out

## CHAPTER 2

---

to the printer. Since the program reads from the left bottom side of the screen to the right top side, the printout is a 90-degree-turned reproduction of the screen image.

Hi-Res Screen Dump is written in machine language. A BASIC loader (the first several lines of the program) puts the machine language (in the form of DATA statements) into the appropriate locations in memory. The BASIC loader also prompts you for the width of the printout. To operate the program correctly, *you must load and run Hi-Res Screen Dump before you load the program that creates the hi-res image.*

### **Selecting a Width and Making a Printout**

You can select either a single-width or double-width printout by POKEing a 1 (for single width) or a 2 (for double width) into location 2 (in other words, POKE 2,1 or POKE 2,2). This location is changed by your selection of width when you are prompted in the BASIC program, but it can be changed at any time. A SYS to location 52224 will initiate a printout of the hi-res screen. You can issue this SYS in direct mode if you have a design on the screen or add it to a hi-res drawing program if you make sure the machine language is loaded into memory before the SYS is encountered. Also, be sure that the printer is turned on before giving the SYS.

### **Using the 64 Super Expander**

The machine language resides at the top of the computer's block of RAM above location 49152 (\$C000). This makes it compatible with the 64 Super Expander, but also means that it cannot be used with the 64 DOS Wedge program, since both occupy the same area of memory. The program is designed to print the hi-res screen that is currently visible. If you want a screen dump when you are not in hi-res mode, POKE location 900 with the high byte of the starting address of the hi-res screen and SYS to location 52224+32. This alternate SYS bypasses the routine that determines the location of the hi-res screen. For example, if your hi-res screen starts at location 57344 (\$E000), you would initiate the screen dump with

**POKE 900,(57344/256): SYS 52256**

# SCREEN CONTROL

---

## Hi-Res Screen Dump

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
HD 3 INPUT "{CLR}WIDTH 1 OR 2";WI$
RQ 4 IF VAL(WI$)<1OR VAL(WI$)>2 THEN3
BG 5 POKE2,VAL(WI$)
JG 10 I=52224
HG 20 READ A:IF A=256 THEN 40
XF 25 PRINT "{CLR}ENTERING DATA":POKE646,A
FC 30 POKE I,A:I=I+1:CH=CH+A:GOTO 20
FD 40 IF CH<>60660 THENPRINT"ERROR IN DATA STATEME
    NTS":END
EH 50 PRINT"SYS 52224 TO START DUMP":END
EP 52224 DATA 173,0,221,41,3,73
FD 52230 DATA 3,160,6,10,136,208
SD 52236 DATA 252,141,132,3,173,24
DH 52242 DATA 208,41,8,240,9,24
RB 52248 DATA 169,32,109,132,3,141
GC 52254 DATA 132,3,169,0,32,189
PD 52260 DATA 255,169,4,170,160,255
FP 52266 DATA 32,186,255,32,192,255
QR 52272 DATA 162,4,32,201,255,176
EX 52278 DATA 3,76,61,204,76,32
KC 52284 DATA 205,169,8,32,210,255
KM 52290 DATA 169,13,32,210,255,162
BM 52296 DATA 0,169,1,141,198,205
ER 52302 DATA 169,0,141,199,205,169
QK 52308 DATA 0,141,200,205,169,199
AG 52314 DATA 141,201,205,32,225,255
PJ 52320 DATA 208,3,76,32,205,138
QS 52326 DATA 72,152,72,32,41,205
SS 52332 DATA 104,168,104,170,173,205
RK 52338 DATA 205,45,203,205,240,12
PC 52344 DATA 173,202,205,13,198,205
QE 52350 DATA 141,202,205,76,143,204
MG 52356 DATA 173,198,205,73,255,45
XM 52362 DATA 202,205,141,202,205,14
GJ 52368 DATA 198,205,173,198,205,201
PQ 52374 DATA 128,240,20,24,173,199
AC 52380 DATA 205,105,1,141,199,205
SM 52386 DATA 173,200,205,105,0,141
BF 52392 DATA 200,205,76,93,204,173
JJ 52398 DATA 202,205,9,128,224,45
JC 52404 DATA 144,10,173,202,205,41
RM 52410 DATA 31,9,128,141,202,205
RS 52416 DATA 142,207,205,166,2,142
DC 52422 DATA 206,205,168,32,210,255
DJ 52428 DATA 152,206,206,205,208,246
HD 52434 DATA 174,207,205,169,1,141
EK 52440 DATA 198,205,169,0,141,202
```

## CHAPTER 2

---

---

BH 52446 DATA 205,56,173,199,205,233  
CA 52452 DATA 6,141,199,205,173,200  
PS 52458 DATA 205,233,0,141,200,205  
EE 52464 DATA 206,201,205,173,201,205  
DH 52470 DATA 201,255,240,3,76,93  
RP 52476 DATA 204,224,45,176,31,24  
SP 52482 DATA 173,199,205,105,7,141  
SB 52488 DATA 199,205,173,200,205,105  
PG 52494 DATA 0,141,200,205,232,169  
RF 52500 DATA 199,141,201,205,169,13  
PD 52506 DATA 32,210,255,76,93,204  
KB 52512 DATA 169,13,32,210,255,32  
DP 52518 DATA 231,255,96,173,201,205  
CF 52524 DATA 41,7,141,204,205,173  
SR 52530 DATA 201,205,74,74,74,168  
JE 52536 DATA 185,146,205,133,251,185  
MK 52542 DATA 172,205,133,252,24,165  
SG 52548 DATA 251,109,204,205,133,251  
AC 52554 DATA 165,252,105,0,133,252  
SQ 52560 DATA 24,173,132,3,101,252  
FF 52566 DATA 133,252,173,199,205,41  
GQ 52572 DATA 7,73,7,168,200,169  
AP 52578 DATA 0,56,42,136,208,252  
GK 52584 DATA 141,203,205,24,173,200  
BJ 52590 DATA 205,101,252,133,252,173  
FJ 52596 DATA 199,205,41,248,168,138  
SK 52602 DATA 72,120,162,52,134,1  
RR 52608 DATA 177,251,162,55,134,1  
DE 52614 DATA 88,168,104,170,152,45  
MR 52620 DATA 203,205,141,205,205,96  
RB 52626 DATA 0,64,128,192,0,64  
MX 52632 DATA 128,192,0,64,128,192  
HD 52638 DATA 0,64,128,192,0,64  
QR 52644 DATA 128,192,0,64,128,192  
QH 52650 DATA 0,64,0,1,2,3  
SH 52656 DATA 5,6,7,8,10,11  
KX 52662 DATA 12,13,15,16,17,18  
HG 52668 DATA 20,21,22,23,25,26  
SJ 52674 DATA 27,28,30,31,256

# Screen Headliner

---

---

*Todd Heimarck*

*This short machine language routine expands a letter to four times its normal size. The large character can then be used in a headline or for a variety of other purposes. The program is also compatible with Commodore printers. For the Commodore 64 and 128 in 64 mode.*

Oversized characters can be useful—on a title screen, in a children's alphabet or math program, or for visually impaired computer users. Finding the right combination of graphics characters usually takes time. You have to experiment. And creating a whole alphabet can use a lot of memory.

The simplest method for displaying huge letters without experimenting or wasting memory is to PEEK the character generator in ROM and print a solid block (reverse space) for each bit that is on. If the bit is off, you print a space. The one major disadvantage to this method is that each character expands to eight times its normal size. Very little space remains on the screen. But by keeping in mind the idea of reading character ROM, you can sidestep this problem with some special Commodore characters.

## **The Quarter-Square Solution**

Hold down the Commodore key and type IKBVDCF. These seven characters, plus a blank space, make up half the quarter-square graphics set. The other half is accessed by typing the same keys while reverse is turned on. There are 16 different characters, one for each combination of quarter squares turned on or off.

Quarter squares enable you to set up what amounts to a medium-resolution screen. It's less complicated to program than a hi-res screen and has better resolution than the usual low-resolution character set. Instead of making characters turn on and off, you control big pixels (each of which is one fourth of a character). A Commodore 64 has the capability of addressing  $80 \times 50$  big pixels.

## CHAPTER 2

---

The 16 characters are the starting point for the "Screen Headliner." The basic idea is to read the character ROM, translate each bit into a big pixel, and print the equivalent quarter-square graphics character. You can do this in BASIC with a lot of PEEKs and POKEs, but machine language is faster and more elegant.

The program is easy to use. After you've entered and saved the program, type RUN. A short machine language program is POKEd into memory. To make it work, you need two POKEs and a SYS:

**POKE 249,0: POKE 250,1: SYS 828**

You should see a large capital A, four characters wide and four deep. Now simultaneously press the Commodore key and SHIFT to switch to the uppercase/lowercase set. Cursor up to the POKEs, press RETURN, and you will see a large lowercase a. Now try putting a 129 into location 250; the result is the same character printed in reverse.

If you've saved a copy of Headliner, type NEW to erase the BASIC loader program. (It won't affect the machine language program, which is safely tucked into the cassette buffer.) Now type in these lines:

```
QG 2 MK=7:REM FOR VIC-20, USE 3
HH 5 PRINT"{CLR}";
RQ 10 FORX=0TO255
PS 20 Y=(XANDMK)*4:POKE249,Y
JM 25 IFXANDMKTHENPRINT"{4 UP}";
XP 30 POKE250,X:SYS828
QG 40 NEXT
```

(Note: Tape users should not save this example program; tape operations erase Headliner from the cassette buffer.) Type RUN, and the whole Commodore character set will parade down the screen.

The top of the large character is printed wherever the cursor happens to be when you SYS. The POKE to 249 determines how far the cursor spaces over before it begins. The number must be between 0 and 35.

Next, POKE the letter's screen code into 250. Ignore the ASCII value; you want the screen code—the number you use when POKeing a character to the screen. Numbers 1–26 are the letters A–Z, 48–57 are the characters 0–9, and so on. To get a reversed character, add 128 to the screen code.

## SCREEN CONTROL

---

After you've POKEd into 249 and 250, enter SYS 828. The oversize character will appear almost instantly.

### Some Bonuses and a Drawback

The original version of this routine figured out the shape of the large character and POKEd the appropriate quarter-square graphics to the screen. But Headliner now PRINTs (using the Kernal PRINT routine at \$FFD2) instead of POKeing. It's necessary to turn reverse on and off repeatedly to get all the quarter squares, which is a little cumbersome. But there are some major advantages to sending everything through \$FFD2.

One bonus is that you can send large characters to a Commodore printer, although you need to change one value to print spaces instead of cursor-rights (see line 951). Enter this to make a printout:

**OPEN 4,4: CMD4: POKE 249,xx: POKE 250,yy: SYS 828**

Remember to replace *xx* with the location where you want to print and to substitute the screen code for *yy*. If you can, adjust your printer's line spacing to zero—so there is no extra space between the characters.

When you've finished printing, PRINT#4:CLOSE4 properly closes the file to the printer. Unfortunately, printers do not allow cursor-up movements; you are limited to one large character per line. To get around this limitation, you could manually move the paper back or use a screen dump program. Or, if you feel ambitious, use CMD to send output to a tape or disk file and then read the data back into an array for dumping to the printer.

Finally, the flexibility of the PRINT statement is at your fingertips: You can print almost anywhere on the screen, in any color you like (just change the cursor color). You can even mix large uppercase, lowercase, and graphics characters on the same screen.

The image shows two columns of large, pixelated characters. The left column contains the uppercase letters 'A', 'B', and 'C' stacked vertically. The right column contains the lowercase letters 'a', 'b', and 'c' stacked vertically. Each character is composed of a grid of small squares, giving them a blocky, digital appearance.

*"Headliner" is compatible with Commodore printers. Uppercase and lowercase print are illustrated here.*

## CHAPTER 2

---

A slight drawback is that each line must be followed by a carriage return, which means you cannot put a character at the right edge of the screen. Nor can you print the large character at the bottom of the screen, which always scrolls up one line.

### How It Works

There are two sets of POKEs in the BASIC loader program. The first loop (688–703) contains the modified ASCII values of the quarter-square graphics characters. Since there is no such thing as an ASCII value of a reversed character, the reverse flag has to be turned on and off. Bit 6 of each character is used to signal whether or not the character is reversed; the number is then ANDed with \$BF (191) to turn off bit 6 before the character is printed.

The second loop is the machine language routine. It goes into the cassette buffer, but is written so that it can be relocated. If you need the cassette buffer for another ML program, or if you are using a Datassette, you can move the routine anywhere else in memory. (The first loop has to stay where it is, however). If you put it in BASIC RAM, you'll have to protect it from being overwritten.

If you're interested in machine language, here's a brief explanation of how Headliner works. The main routine first checks which character set is being used and sets a zero-page pointer accordingly. The screen code number is then multiplied by eight and added to the pointer. Once the pointer is set, the bytes from character ROM are loaded in two by two. By alternately shifting left the bytes (ASL) and rotating left the accumulator (ROL), a number from 0 through 15 is generated. This number is used as an offset to look up the appropriate quarter-square graphics character in the table at 688. Bit 6 is checked, and, if it's set, reverse is turned on. And, finally, a JSR to \$FFD2 prints the character. The program then loops back to get the next set of bits.

### Screen Headliner

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
BA 5 PRINT "{CLR}PLEASE WAIT A MOMENT"  
SE 10 T=0:FORJ=688TO703:READK:T=T+K:POKEJ,K:NEXT  
HX 15 IFT<>3078THENPRINT"ERROR IN DATA STATEMENTS"  
:STOP  
KK 20 T=0:FORJ=828TO1006:READK:T=T+K:POKEJ,K:NEXT
```



## SCREEN CONTROL

---

```
CP 25 IFT<>20306THENPRINT"ERROR IN DATA STATEMENTS
":STOP
HP 30 POKE249,0
QK 688 DATA32,188,190,226,172,225,191,251
GP 696 DATA187,255,161,236,162,254,252,96
SK 828 DATA 169,208,133,004,173,024
JR 834 DATA 208,041,002,240,004,169
GA 840 DATA 216,133,004,169,000,162
FF 846 DATA 003,006,250,042,202,208
PB 852 DATA 250,024,101,004,133,004
EA 858 DATA 165,250,133,003,173,014
HS 864 DATA 220,041,254,141,014,220
DD 870 DATA 165,001,041,251,133,001
DF 876 DATA 169,000,133,250,169,005
XP 882 DATA 133,002,160,000,177,003
RB 888 DATA 133,005,230,003,177,003
FC 894 DATA 133,006,230,003,198,002
DC 900 DATA 240,028,162,004,169,000
PE 906 DATA 006,006,042,006,006,042
EP 912 DATA 006,005,042,006,005,042
QR 918 DATA 164,250,153,048,002,230
HM 924 DATA 250,202,208,232,240,210
RX 930 DATA 165,001,009,004,133,001
GP 936 DATA 173,014,220,009,001,141
FB 942 DATA 014,220,160,000,166,249
CB 948 DATA 240,008,169
AG 951 DATA 029:REM 032 IF USING A PRINTER
EJ 952 DATA 032,210
MC 954 DATA 255,202,208,250,169,004
QH 960 DATA 133,006,185,048,002,170
KK 966 DATA 189,176,002,133,005,041
GD 972 DATA 064,240,005,169,018,032
DA 978 DATA 210,255,165,005,041,191
EK 984 DATA 032,210,255,169,146,032
BS 990 DATA 210,255,200,198,006,208
JE 996 DATA 221,169,013,032,210,255
RR 1002 DATA 192,016,208,196,096
```



## CHAPTER 3

---

---

# Input/Output Routines



# 64 RAM Disk

---

---

*MiAngelo Moore*

*This short utility is an excellent programming development tool—it's like having instant access to an 8K disk drive. For the Commodore 64 and 128 in 64 mode.*

Some computers have a very useful feature called a *RAM disk*—an area of memory used for temporary program or data storage. Although this area basically works like a tape or disk drive for program storage, it is not permanent. Anything stored there is erased when you turn the computer off. The advantage of a RAM disk is that you can have almost instant access to the information stored there—without waiting for data to be loaded from or saved to an external storage device. For this reason, it provides an excellent program development tool. The Commodore's operating system does not directly support a RAM disk, but this machine language program, "64 RAM Disk," provides one. A maximum of 8K is available for program storage. This RAM disk can hold only one program at a time, even if the program is less than 8K long.

## **A Development Tool**

There are several uses for a RAM disk. Suppose you need a quick disk directory, but you have a program in memory and no DOS Wedge. You can save your program to RAM disk, `LOAD"$0",8`, and then retrieve the program after viewing the directory. Or, if you want to append a subroutine to a program, you can save the subroutine to RAM disk, load the main program, change the start-of-BASIC program pointer, retrieve the program from RAM disk, and change the start-of-BASIC program pointer back to 2049, and your program will be appended. There are numerous uses for 64 RAM Disk as a development tool.

In order to make the program easy to use, three new commands are wedged into BASIC. Note that these commands should be used only in immediate mode.

## CHAPTER 3

---

- ← Save program currently in memory to RAM disk.
- ↑ Retrieve program in RAM disk and transfer to memory.
- < Clear RAM disk memory. *Be extremely careful when using this command.* It completely erases the contents of the RAM disk.

The RAM disk is designed for use with BASIC programs. The block of memory saved is the area between the address in the start-of-BASIC program pointer (contained in locations 43–44) and the address in the end-of-BASIC program pointer (contained in locations 45–46). Data from the RAM disk is always reloaded beginning at the address in the start-of-BASIC program pointer. Thus, the RAM disk is not suitable for storing machine language routines from other areas of memory.

Before you save a program to the RAM disk, be sure the program is no longer than 8K. If it's longer, it will not be placed in the RAM disk, and a message will appear. The reason for the 8K restriction is that the memory area used for 64 RAM Disk is located in the 8K RAM area under BASIC ROM. This means programs that copy BASIC to RAM for modification or programs that place high-resolution graphics under the BASIC ROM will not work with 64 RAM Disk. Also, 64 RAM Disk will not work with any program that uses memory locations 49152–49416, since this is where the machine language for 64 RAM Disk is located.

### Typing It In

As you're typing in the program, be especially careful with the numbers in the DATA statements. When you've finished, save a copy. To use it, load it and type RUN. A message will tell you to enter **SYS 49152** to activate and **SYS 64738** to deactivate. The latter SYS resets the computer just as if you had turned it on. (Unlike most wedge routines, 64 RAM Disk is *not* disabled when you press RUN/STOP-RESTORE.) The three commands discussed above are now at your disposal.

### Memory Locations Used

2–3 and 251–254	Temporary storage
40960–49151	64 RAM Disk storage area
49152–49416	64 RAM Disk machine language

## INPUT/OUTPUT ROUTINES

### 64 RAM Disk

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
RB 10 PRINT"{CLR}{4 DOWN}{WHT}"SPC(8)"LOADING MACH
    INE LANGUAGE"
EG 20 FORI=49152TO49416:READA:POKEI,A:X=X+A:NEXT
CS 30 IFX<>33629THENPRINT"ERROR IN DATA STATEMENTS
    .":STOP
FP 40 PRINT"{CLR}{2 DOWN}{RVS} SYS 49152 TO ACTIVA
    TE"
ER 50 PRINT"{DOWN}{RVS} SYS 64738 TO DEACTIVATE
    {OFF}{16 SPACES}(RESET COMPUTER)"
JP 60 DATA 169,76,133,115,169,13,133,116,169,192
BE 70 DATA 133,117,96,230,122,208,2,230,123,32
JG 80 DATA 121,0,201,95,208,83,165,43,133,251
GX 90 DATA 165,44,133,252,165,45,133,253,165,46
FJ 100 DATA 133,254,56,229,252,201,32,176,44,169
QR 110 DATA 0,133,2,169,160,133,3,160,0,177
XX 120 DATA 251,145,2,230,251,208,2,230,252,230
GK 130 DATA 2,208,2,230,3,165,251,197,253,208
DF 140 DATA 234,165,252,197,254,208,228,230,122,20
    8
EB 150 DATA 15,230,123,230,122,208,2,230,123,169
FK 160 DATA 226,160,192,32,30,171,76,121,0,201
AD 170 DATA 94,208,79,165,43,133,251,165,44,133
JG 180 DATA 252,169,54,133,1,165,2,133,253,165
XQ 190 DATA 3,133,254,169,0,133,2,169,160,133
KM 200 DATA 3,160,0,177,2,145,251,230,2,208
QD 210 DATA 2,230,3,230,251,208,2,230,252,165
PH 220 DATA 2,197,253,208,234,165,3,197,254,208
RG 230 DATA 228,165,251,133,45,165,252,133,46,230
QE 240 DATA 122,208,2,230,123,169,55,133,1,76
XS 250 DATA 121,0,201,60,208,166,169,160,133,3
AG 260 DATA 169,0,133,2,168,169,0,145,2,230
JB 270 DATA 2,165,2,201,3,144,244,230,122,208
EC 280 DATA 141,230,123,76,121,0,80,82,79,71
DF 290 DATA 82,65,77,32,73,83,32,84,79,79
HP 300 DATA 32,76,79,78,71,32,70,79,82,32
JG 310 DATA 84,72,69,32,82,65,77,32,68,73
PF 320 DATA 83,75,46,13,0
```

# Keyboard to Joystick Converter

---

---

*David A. Dunn*

*Did you ever play a good game that would have been even better if it offered joystick instead of keyboard controls? Here's a solution. For the Commodore 64 and 128 in 64 mode.*

Back in the days of the PET/CBM there was no way to hook up a joystick, so we all got pretty good at playing games using the keyboard. But it often became confusing: One game's controlling keys were different from another's. The addition of a joystick port on Commodore machines was more than welcome. But I still play some of the older games that require keyboard control and occasionally still find a program or two that insists that you use the keyboard.

I wrote "Keyboard to Joystick Converter" as a solution to this problem. It allows you to use a joystick with Commodore 64 and 128 programs that don't provide joystick control.

## **Plan A or Plan B?**

Type in Program 1 and save a copy before running it. Although it's written in BASIC, it includes a machine language routine in the form of DATA statements. When you run the program, it asks you to choose plan A or plan B. By making this selection you determine where in memory to put the machine language to avoid interference with the main program. Plan A uses the cassette buffer. Plan B uses the free area at 758-767. For most programs, plan A works fine, although you should avoid this option if you're using a Datassette. If plan A doesn't work, rerun the program and select plan B.

You're next asked to enter the keys corresponding to each direction and the fire button for each joystick port. Let's say a program requires you to press the J key to move left and the K key to move right. When Keyboard Converter asks you to



## INPUT/OUTPUT ROUTINES

---

---

press the key for left, press J. Next, you're asked to press the key for right. Press K. If there's no need for the joystick to move in a particular direction, just press any key.

### The Breaks

After you've done this for both joystick ports, the program asks for the maximum break length. Joysticks vary not only in size, shape, and price, but also in quality. Many joysticks make good, solid contact between the wires inside, but some do not. Some lesser quality joysticks will make contact one moment, break contact for a fraction of a second, then make good contact again. This makes the computer think the joystick has been moved twice in that direction. In some programs this causes no problem; in others it may.

Keyboard Converter can make up for this deficiency by distinguishing between a break in the contacts and the true release of the joystick. It does so by sensing how long the contacts have been broken. Generally, a period of three jiffies (one jiffy equals 1/60 second) or less is a result of a break in the contacts. Any longer than that usually indicates a true release of the joystick.

When Keyboard Converter asks for maximum break length, press RETURN to indicate three jiffies. If you have any problems with repeating movements, you can increase the break length by typing a higher number.

Now move the joystick around. You should see the corresponding letters on the screen. Now you can load and run a game, and use the joystick instead of the keyboard. Be sure not to press RUN/STOP-RESTORE after running Keyboard Converter—this will disable the joystick.

The program works by modifying the IRQ routine, which is executed 60 times per second. It normally scans the keyboard and increments the internal clock. When Keyboard Converter goes into effect, however, the computer scans not only the keyboard, but also the joystick ports. If the joystick has been moved, the code for the corresponding key is placed in the keyboard buffer to simulate a keypress.

### A Sample Program

For demonstration purposes, type in Program 2. It's a fast-action game (based on "Worm of Bemer," which appeared in

## CHAPTER 3

the April 1984 issue of *COMPUTE!* magazine) that uses keyboard control. After using Keyboard Converter, you'll be able to control it with a joystick.

After typing in "Worm," save a copy. Run Keyboard Converter and select plan A. Here are the controls:

I — up  
M — down  
J — left  
K — right

Now load and run Worm. In this game you control a worm and gain points by eating mushrooms that are placed randomly on the screen. The game ends when the worm runs over its own tail or into a wall. Sounds easy, but there's a catch. Your worm grows every time it eats a mushroom. This makes it increasingly difficult to keep from running over its own tail.

Lines 160-190 determine the direction keys. By modifying the number 25 in line 260, you can change the speed of the game.

### Program 1. Keyboard to Joystick Converter

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
KX 10 FORI=1TO168:READA:X=X+A:NEXTI
BC 20 IFX<>23067THENPRINT"{CLR}TYPING ERROR IN DAT
A LINES":STOP
CG 30 RESTORE:TT$=CHR$(142)+CHR$(8)+"{CLR}{RVS} KE
YBOARD TO JOYSTICK ":PRINTTT$
MK 40 FORI=828TO856:READA:POKEI,A:NEXT
XC 50 DATA120,169,49,141,20,3,169,234,141,21,3,88,
96
HD 60 DATA165,1,72,41,254,133,1,32
PK 70 DATA27,191,104,133,1,76,126,234
KC 80 FORI=758TO767:READA:POKEI,A:NEXT
SC 90 DATA198,1,32,27,191,230,1,76,126,234
PX 100 PRINT"{DOWN}(IF PLAN A DOESN'T WORK, TRY PL
AN B){2 UP}"
DM 110 PRINT"PLAN A OR B{2 SPACES}A{3 LEFT}";
QD 120 INPUTP$:I=758:IFP$="A"THENI=841
DC 130 POKE835,I/256:POKE830,I-PEEK(835)*256
HA 140 FORI=0TO73:POKEI+48923,PEEK(I+59953):NEXT
HS 150 FORI=48997TO49125:READA:POKEI,A:NEXT
JQ 160 DATA169,255,133,203,141,245,191,169,129
AG 170 DATA133,245,169,235,133,246,162,0,142,141
RG 180 DATA2,142,2,220,160,1,169,16,32,186,191
```

## INPUT/OUTPUT ROUTINES

---

```
XS 190 DATA232,74,144,249,136,16,244,140
PS 200 DATA2,220,173,245,191,201,16,240,35,165
ED 210 DATA203,162,3,157,230,191,189,230,191
EF 220 DATA201,255,208,6,202,16,246,76
EG 230 DATA135,234,133,203,162,0,189,231
HF 240 DATA191,157,230,191,232,224,3,144
DQ 250 DATA245,76,221,234,72,57
QR 260 DATA0,220,208,15,189,246,191,201
AE 270 DATA65,144,10,41,63,13,141,2
QM 280 DATA141,141,2,104,96,197,203,144
XQ 290 DATA7,133,203,104,141,245,191,96,72,165,203
DQ 300 DATA201,255,104,176,241,144,233
GF 310 FORI=49126TO49141:POKEI,255:NEXT:FORA=0TO4:
    READA$(A):NEXT
HJ 320 DATAUP,DOWN,LEFT,RIGHT,FIRE
EJ 330 FORF=1TO7:READF$(F):NEXT
GX 340 DATASHIFT,LOGO,SHIFT+LOGO,CTRL,SHIFT+CTRL,L
    OGO+CTRL,SHIFT+LOGO+CTRL
FA 350 POKE808,239:PRINTTT$(FORI=0TO1:PRINT"JOYSTI
    CK PORT";I+1
DJ 360 FORA=0TO4:PRINT"PRESS KEY FOR ";A$(A);"? ";
QJ 370 K=PEEK(197):F=PEEK(653):IFQTHENIFK<64ORFTHE
    NQ=0:GOTO400
CG 380 IFK=64ANDF=0THENQ=1
ES 390 GOTO370
KR 400 IFK=64THENK=K+F:A$=F$(F):POKE198,0
GA 410 POKE49142+I*5+4-A,K
JR 420 GETG$:IFG$>" "THENA$=G$:GOTO420
QB 430 IFA$=" "THENA$="SPACE"
CG 440 IFA$=CHR$(13)THENA$="RETURN"
MS 450 POKE216,1:PRINTA$:NEXTA:PRINT:NEXTI:POKE198
    ,0
QG 460 PRINT"MAXIMUM BREAK LENGTH? 3{LEFT}";:I=3
HP 470 GETA$:IFA$=CHR$(13)THENPRINT:PRINT:GOTO500
JG 480 IFA$<"0"ORA$>"9"THEN470
DS 490 I=VAL(A$):PRINTA$"{LEFT}";:GOTO470
RF 500 POKE808,237:POKE49047,I:POKE49076,I
QA 510 PRINT"{2 DOWN}ARE YOUR CHOICES SATISFACTORY
    ? (Y/N)"
JD 520 GETA$:IFA$="N"THEN350
XB 530 IFA$="Y"THENSYS828:PRINTCHR$(9);:NEW
EF 540 GOTO520
```

### Program 2. Worm (Demo)

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
HH 100 POKE53281,6:PRINT"{CLR}{CYN}"TAB(13)"*** HU
    NGRY LIKE A WORM ***":A=RND(0)
HB 110 POKE53280,0:POKE53281,0:FORI=0TO19:POKEI083
    -I,160:POKEI084+I,160:NEXTI
```

## CHAPTER 3

---

---

```
FR 120 FORI=0TO21:POKE1104+I*40,160:POKE1143+I*40,
160:NEXTI
FA 130 FORI=0TO19:POKE1984+I,160:POKE2023-I,160:NE
XTI
CP 140 DIMS(999):WH=292:WT=WH:D=1:POKEWH+1024,81:S
(WH)=D:GOSUB320:GOSUB350
EC 150 GETA$:D1=D
CG 160 IFA$="J"THENDE=-1
AC 170 IFA$="K"THENDE=1
HG 180 IFA$="I"THENDE=-40
BR 190 IFA$="M"THENDE=40
DQ 200 IFD=-D1THENDE=D1
BQ 210 S(WH)=D:WH=WH+D
AK 220 IFPEEK(WH+1024)=65THENSC=SC+10:GOSUB350:GOS
UB320:GOSUB320:L=5:GOTO240
CA 230 IFPEEK(WH+1024)<>32THEN270
PG 240 POKEWH+1024,81:IFLTHENL=L-1:GOTO260
HH 250 POKEWT+1024,32:WT=WT+S(WT)
HM 260 FORT=1TO25:NEXTT:GOTO150
XF 270 P=PEEK(WH+1024):POKEWH+1024,86:PRINT:PRINT:
PRINTTAB(2)CHR$(18)"YOU RAN ";
HC 280 IFP=81THENPRINT"OVER YOURSELF"
SX 290 IFP=160THENPRINT"INTO A WALL"
KK 300 PRINT:PRINTTAB(2)CHR$(18)"GAME OVER":END
KA 310 REM *** GROW MUSHROOMS ***
MK 320 R=INT(RND(1)*960)+1064:IFPEEK(R)<>32THEN320
FA 330 POKER,65:RETURN
CE 340 REM *** PRINT SCORE ***
RF 350 PRINTCHR$(19)"SCORE: "RIGHT$("0000"+MID$(ST
R$(SC),2),5):RETURN
```

# List Pager

---

**Robert A. Stoerrle**

*If you own a printer, you'll appreciate this short utility. It allows you to divide printouts into pages, insert headers, and print page numbers. For the Commodore 64 and 128 in 64 mode.*

No matter how much you adjust the paper on a printer, it seems that one line always prints smack dab on the perforation. When you separate the pages, the line is cut in half. Some printers have a skip-over-perf, or *paging*, option, accessed by setting a DIP switch or sending an escape code. On the 1526 or MPS-802, for example, you turn paging on with PRINT#4, CHR\$(147).

But this important feature is missing from the Commodore 1525, MPS-801, and MPS-803 printers. "List Pager" is a short machine language utility that offers a solution to the problem. It causes the printer to automatically skip to the top of the next page when it runs out of room on the current page. And, if you want, it will print both a header and the page number at the top of each page. Even if your printer already skips over perforations, you'll find the header and page-numbering options useful.

## **Special Loading Instructions**

Type in and save the List Pager loader program, which uses a special technique to store the machine language (ML) program in a string variable. The ML has to be saved to tape or disk as an object file (a pure machine language program) before you can activate it.

When you run the program, you're asked if you want to save it to tape (T) or disk (D). If you press D, you're prompted for a filename. If you're saving it to the same disk as the loader program, be sure to use a different program name. After you give it a name, the machine language portion of the program will be saved to disk. If you're using tape, be sure to have a blank tape in the Datassette, preferably positioned to

## CHAPTER 3

---

the beginning. Press T, and the computer will prompt you to press RECORD and PLAY (tape users don't name the program because the ML string becomes the name of the file).

To load the ML program from disk, type:

**LOAD "filename",8,1: POKE56, PEEK(56) -1:NEW**

The first time you load it from tape, type this line:

**OPEN 1:CLOSE 1:POKE 56,PEEK(56) -1:CLR**

In both cases, you should be loading the *object file* that was created, not the loader program you typed in. Besides loading the program, these procedures clear all variables.

### Protecting the Cassette Buffer

The cassette buffer is a section of memory that acts as a sort of pipeline between the cassette drive and the computer. It's a safe place to store ML programs on the 64 because BASIC doesn't use this area of memory.

If the List Pager ML is in the cassette buffer when a program is loaded from tape, it will be overwritten because the data on tape is temporarily stored in the buffer. So, tape users will have to take measures to protect the cassette buffer while using List Pager.

Tape users should *never* access tape files while List Pager is active. Before loading, saving, or opening, press RUN/STOP-RESTORE to turn off List Pager. After you have loaded or saved a program, List Pager will be gone, overwritten by the program data. To load it back in, put the cassette containing the ML part (the object code) of List Pager into the Datassette and enter

**OPEN1: CLOSE1**

(You should not enter the POKE to 56 unless this is the first time you're loading List Pager.) The program is built into the tape file header and will load directly into the cassette buffer. You'll have to enter the SYS given below to start it up.

The order of loading programs is just the opposite for disk owners. Tape users load the program to be listed first and then load List Pager (because of the cassette buffer). With a disk drive, you must load List Pager first (because of the NEW instruction after the LOAD and POKE). Once it's loaded, you

## INPUT/OUTPUT ROUTINES

---

can load the program or programs to be listed. Disk owners need to load the ML only once; disk access doesn't affect the cassette buffer.

### Telling It What to Print

Once you've loaded List Pager, following the instructions above, you must activate it and tell it what to do at the top of each page. The format for this command is

**SYS 833,"Header"**

If you want a header at the top of each page, put it inside quotation marks. If you want the page number to be printed, insert a number sign (#) into the header at the point where you want the page number to be printed. For example,

**SYS 833,"LIST PAGER.....PAGE #"**

The designated header will be printed at the top of each page. Pages will be numbered consecutively, starting at page 1. If you do not want a header at the top of each page, don't put anything between the quotation marks:

**SYS 833," "**

You must *always* use a comma and quotation marks, even if you do not want to print a header.

After you have entered the header line, position the paper so that the printhead will start printing a few lines down to allow for a top margin. If you've not already done so, load the program you want to list. (Remember, tape users should load the program to be listed before loading List Pager.) Type the following line:

**OPEN 1,4:CMD 1:LIST**

The printer should print the header, if you have specified one, and start to list the program. When it's finished, type

**PRINT #1:CLOSE 1**

List Pager will remain in effect until you press RUN/STOP-RESTORE. That is, it will continue to separate pages and print the header at the top of each page.

### Options

The program is written for standard 8½-by-11-inch paper, with 6 lines per inch—a total of 66 lines on each page. If you're using nonstandard paper, you can modify the parameters. Some European countries, for example, use slightly longer paper. The program defaults to 54 printed lines on a page of 66 lines. This leaves 2 lines for the header (one printed, one blank) and 10 lines between pages (five at the top, five at the bottom).

After you've loaded the program, you can change the number of printed lines per page with this POKE:

**POKE 926,x + 4**

where  $x$  is the desired number of printed lines per page. For example, if you want to print 38 lines per page, you would type **POKE 926, 38+4** after the program is loaded (but before you enter the SYS). The default number of printed lines per page is 54.

To change the total number of lines per page (printed and unprinted), enter

**POKE 934,x**

where  $x$  is the number of lines per page. The default number is 66.

### Not Just for Program Listings

List Pager is designed to work with both dot-matrix and letter-quality printers. However, it will work only with a printer with a device number of 4. Also, it will not affect listings on the screen. It can be used for a variety of purposes, not just program listings. For example, you can use it within BASIC programs to divide output into pages.

List Pager works by inserting a "wedge" into the routine that the computer uses to print a character. Every time the computer wants to print a character, it goes to the wedge program first. If the character is a carriage return and the computer is at the bottom margin of a page, it skips to the next page and prints the header. If the character is not a carriage return, it's sent to the normal print routine.

To make List Pager work, the cassette buffer is used to hold the program. Because of this, some conflicts may arise.



## INPUT/OUTPUT ROUTINES

Using the cassette drive while this routine is in effect will generally cause the system to lock up. In addition, other programs which reside in the cassette buffer, such as "The Automatic Proofreader," cannot be used at the same time as List Pager.

### List Pager

See special instructions in article before running.

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
PF 100 PRINT "{CLR} {3 DOWN} *** LIST PAGER ***":FOR
I=1TO185:READA:F$=F$+CHR$(A):NEXT
MB 110 PRINT "{2 DOWN}SAVE ON {RVS}T{OFF}APE OR
{RVS}D{OFF}ISK?"
FD 120 GETA$:IFA$<>"T"ANDA$<>"D"THEN120
DX 130 PRINT "{CLR}":IFA$="D"THEN150
FC 140 OPEN1,1,1,F$:CLOSE1:END
KG 150 INPUT "FILENAME";N$:OPEN1,8,4,N$+" ,P,W":PRIN
T#1,CHR$(65)CHR$(3)F$:CLOSE1
FM 160 DATA 169,1,133,251,173,39,3,201,3,240,9,141
XS 170 DATA 251,3,173,38,3,141,250,3,169,127,141,3
8
AA 180 DATA 3,169,3,141,39,3,32,115,0,32,115,0,160
BQ 190 DATA 0,132,252,177,122,240,9,201,34,240,6
SA 200 DATA 145,55,200,208,243,24,132,253,152,101
HM 210 DATA 122,133,122,96,132,254,72,165,154,201
FM 220 DATA 4,208,50,165,252,208,6,32,192,3,32,242
DG 230 DATA 3,104,72,201,13,208,34,32,242,3,164,25
2
GE 240 DATA 192,58,208,21,32,242,3,200,192,66,208
KA 250 DATA 248,169,0,133,252,165,251,248,24,105,1
PF 260 DATA 216,133,251,104,164,254,96,104,164,254
JD 270 DATA 76,249,3,165,253,240,23,160,0,196,253
AK 280 DATA 240,17,177,55,201,35,208,5,32,222,3,16
9
EJ 290 DATA 0,32,249,3,200,208,235,76,242,3,165,25
1
XS 300 DATA 74,74,74,74,9,48,32,249,3,165,251,41,1
5
DP 310 DATA 9,48,76,249,3,230,252,169,13,76,249,3,
76
```

# USR Joystick Reader

---

---

*Tim Gerchmez*

*Programming the joystick in BASIC may give disappointing results. It's just too slow. This program offers the speed of machine language for use in BASIC programs. For the Commodore 64.*

One of the most important decisions a computer manufacturer makes when designing a new computer is the number of features to include. The more features, the higher the cost. In order to keep the price of the 64 competitive while still maintaining superior sound and graphics, Commodore decided to take a few shortcuts with the BASIC language. Simply reading the joystick, for example, requires a complex series of POKEs and PEEKs. It's not only complicated but also slow.

"USR Joystick Reader" makes reading a joystick quicker and easier. It employs the USR function to simulate the JOY function found in the much larger versions of BASIC included with the Plus/4, 16, and 128.

## **The Mysterious USR**

USR (which stands for *user*) is not a fitting name for this function. It's hardly *used* at all because most people don't know what it does or how to make it work.

USR is like a cross between SYS and FN. Instead of the function being defined in BASIC by using the DEF FN statement, it's written in machine language (ML). First, you either POKE or load the ML into memory. Then you tell the computer where your ML routine is by POKEing locations 785 and 786 with the low byte and high byte of the starting address.

Now that you've defined the function, you're ready to put USR into your BASIC program. Like FN, USR is followed by a numeric expression inside parentheses. It can be a number as in USR(6), a variable like USR(X), or a complex expression such as USR(PEEK(X)+256\*PEEK(X+1)). When the USR function is executed, the computer evaluates the expression within the parentheses and puts that value into floating-point

## INPUT/OUTPUT ROUTINES

---

accumulator 1 (FAC1). (For more information on FAC1 and USR, refer to *Programming the 64*, published by COMPUTE! Books.)

USR then executes your ML routine, which takes the floating-point number in FAC1, processes it in some manner, and stores the result back into FAC1. If you end your routine with an RTS instruction, the computer returns to BASIC and makes USR equal to the new value in FAC1. USR can then be treated like any other value as in  $Y=USR(X)$  or `PRINT USR(3)`.

USR is easier to use than SYS because you can pass values between BASIC and ML, and you don't have to specify an address. It's faster than FN because the function is defined in ML. But it's more difficult to set up than either one.

### Putting USR to Work

Fortunately, you don't have to know anything about USR to add USR Joystick Reader to your own programs. Just type it in and save it on tape or disk. Now type RUN. The program is a BASIC loader that POKEs an ML joystick reader into memory and points the USR vector to it. If DATA ERROR is displayed, you've made a typing error in the DATA statements, and you should correct your mistake and resave the program.

Now, instead of typing a series of PEEKs and POKEs, just use  $A=USR(1)$  to read a joystick in port 1 or  $A=USR(2)$  to read port 2. The value of A will be a number from 1 through 8, corresponding to the eight directions, and 0 if the joystick is in the center position (see figure). To read the fire button, use  $B=USR(3)$  for port 1 or  $B=USR(4)$  for port 2. The value of B will be 1 if the button is pressed and 0 if it's not.

To use USR Joystick Reader in your own program, include it at the beginning. You can add the following lines to the loader program to check whether the joystick (port 1) is pointing to the right:

```
200 A=USR(1):IF A=2 THEN PRINT "RIGHT"  
220 GOTO 200
```

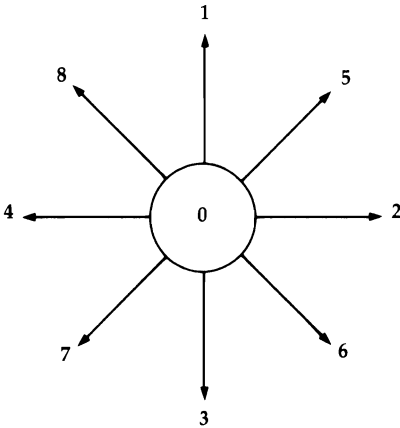
You can also check for the fire button by adding this line:

```
210 B=USR(3):IF B=1 THEN PRINT "FIRE"
```

## CHAPTER 3

---

### Joystick directions



### USR Joystick Reader

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
GE 10 FORA=679TO743:READB:CH=CH+B:POKEA,B:NEXT:IFC
H<>5899THENPRINT"DATA ERROR":END
EP 20 POKE785,167:POKE786,2:PRINT"{CLR}{DOWN}ACTIV
ATED"
BF 30 DATA 32,247,183,165,20,201
GD 40 DATA 3,176,26,73,3,170
CM 50 DATA 189,255,219,41,15,133
XF 60 DATA 2,169,15,56,229,2
HP 70 DATA 170,189,221,2,168,169
QC 80 DATA 0,32,145,179,96,160
GF 90 DATA 0,41,1,170,189,0
RS 100 DATA 220,41,16,208,1,200
XD 110 DATA 169,0,32,145,179,96
PS 120 DATA 0,1,3,0,4,8,7,0,2,5,6
```

# Disk Title Changer

---

*Michael Broussard*

*One step in organizing any growing disk library is renaming disks. If you have a 1541 disk drive, this eight-line BASIC program does the job efficiently and fast. The article also explains how to read from and write to disk sectors. For the Commodore 64 and 128 in 64 mode.*

Before you can do anything useful with a disk, you have to format it. And when you format a new disk, you have to choose a name for it. You may name it something ordinary, like DISK 15, if it's number 15 in your collection. Or you may give it an unusual name like UNICORN.

Many people name their disks according to the kind of programs that are on them: GAMES 1 or UTILITIES 3/87, for example. Usually, though, you're not sure what sort of programs or files will ultimately be stored on a new disk. As it fills up, you may wish you had named it something else. Although the Commodore disk operating system (DOS) provides an easy way to rename files, it's not such a simple task to rename a disk. This eight-line BASIC program does the job simply. You'll see how it works a little later, but first let's see how to use it.

**A word of caution:** The punctuation in the program is extremely important. When you enter the program, watch closely for quotation marks, parentheses, commas, colons, and semicolons. A typing mistake could potentially ruin one of your disks. It's a good idea to use "The Automatic Proof-reader" while you're entering the program.

Using the program is easy—load it (but don't type RUN yet) and then insert a disk whose name you want to change. Make sure the write-protect notch is not covered. Next, type RUN. The current name of the disk will be displayed, and you're prompted for a new one. Enter the new name and press RETURN; the disk directory will be updated and the name of the disk changed. That's all there is to it.

### Disk Organization

You don't have to understand how the program works to use it, but if you've been planning to learn some of the disk commands, this is a good time to begin. First, let's take a brief look at how the disk operating system (DOS, for short) stores information on the disk.

Data on a disk is organized into 35 concentric rings, or tracks. These tracks are numbered: Track 1 is the outermost track and track 35 is the last track, near the center of the disk. Each track, or "lap," around the disk is further divided into blocks, or sectors, which can store 256 bytes (characters) of data each.

Most of the space on a disk is available for storing programs or files. But a few sectors are used by DOS as a directory to store housekeeping information, such as the disk's name, the names of all the files on the disk, and what sort of files they are (program files, sequential files, and so forth). In addition, a block availability map (BAM) provides a chart telling which sectors are not being used so that DOS knows where it may put new files.

Whenever you save a program, a new file entry is placed in the directory, and the BAM is updated to reflect which blocks have been used. Conversely, when a file is scratched, the file entry is marked as free, and the BAM is changed so that the disk blocks that were used by the file are freed up.

But what does all this have to do with changing the name of the disk? Usually, the commands given to DOS are ones that manipulate files. These *high-level* commands cause the disk drive to execute fairly complex routines. When you load a program, DOS takes care of reading the directory to find out whether your program is on the disk. Then it finds out where all the blocks of your program are, and it transfers them from disk to the memory of the computer. You don't care where all the pieces are—all you know is that your file is on the disk. DOS does the rest.

From inside a program, however, it's possible to do more primitive, *low-level* disk operations. For example, by naming a specific track and sector, you can read or write specific bytes from a particular block of data on the disk, as opposed to reading or writing a whole file, which may consist of many blocks. By using this feature, you can change the name of a disk. Now let's examine the program, line by line.

## INPUT/OUTPUT ROUTINES

---

### A Close-Up Look at Program Operation

The first thing the program does (line 10) is close the error channel (15) and then reopen it, sending it the Initialize command ("I0:") to force the disk drive to read the BAM. This is done to make sure there are no side effects from either a previous disk or a previous program that may have opened files on the disk and not closed them. Note that you can close a file that's closed, but you'll get an error if you try to open an already-open file.

Next, channel 5 (an arbitrary choice) is opened as a buffer for reading from the disk. The number sign tells the drive to set aside one of its internal buffers. If you entered OPEN5,8,5,"#2", it would specify buffer 2. Without a number ("#"), it means "we'll accept any available buffer." In most cases, you don't need to worry about which buffer is used. A string (B\$) is then set to the null string (" "). The current name of the disk will be read into B\$.

Track 18, sector 0, contains directory header information, including the disk name. This is the block we're interested in changing, so we tell DOS to read the directory header block with the User-1 (U1:) block-read command in line 20. *Always read disk blocks with the U1: command; B-R (Block Read) is unreliable.* Notice the four numbers that follow U1: 5, 0, 18, and 0. The 5 is the channel number (from OPEN 5,8,5 in the previous line). The first 0 is the drive number—1541s are always drive 0 and (usually) device 8—and 18,0 means track 18, sector 0.

The U1: command reads a block from disk and puts it into a memory buffer inside the disk drive. Your computer doesn't have the information, however; it's still inside the drive. The second part of line 20 makes the drive set the buffer pointer ("B-P") to character number 144. B-P is followed by the channel number and character number. As you may have guessed, the name of the disk starts at 144 (hex \$90).

We could read the whole block from the buffer, except for a small problem. Each block contains 256 bytes, and Commodore BASIC allows a maximum of 255 characters in a string. We'd have to split the information into at least two strings to make it work correctly. The B-P command allows us to read only the disk name, and later only the name will be changed.

Line 30 of the program extracts the 16 characters of the disk name from the buffer inside the drive. The line that

## CHAPTER 3

---

makes A\$ into CHR\$(ASC(A\$ + CHR\$(0))) is not really necessary, but it's a good idea to include it if you plan to read other sectors from disk. A zero sometimes translates into a null string rather than a CHR\$(0). This conversion from ASCII to CHR\$ takes care of any potential problems.

Line 40 prints the current disk name, and line 50 then asks for a new name. A check is made to be sure it's 16 characters or fewer (16 is the maximum number of characters allowed for a filename). If necessary, line 60 pads the new name with shifted spaces to make it exactly 16 characters long.

In line 70, we use B-P again, to point the buffer to character 144. The new disk name, N\$, is printed to channel 5 and into the buffer. The semicolon following N\$ guarantees that a carriage return (CHR\$(13)) is not appended to the end of the disk name.

But we haven't changed the name yet. What has happened so far is that a sector has been read into a disk buffer, the disk name has been extracted, and a new name has been sent to the buffer. The buffer has been changed, but nothing has been written to the disk. We have to finish the job with the User-2, or U2: block-write command. *B-W (Block Write), like B-R, is unreliable. Always use U2: to write a block to disk.* Now we've successfully renamed the disk.

The last steps (line 80) are to read the error channel and initialize the disk. If everything worked properly, you should see DISK STATUS:0 OK.

Why initialize the disk again? Try this experiment: Remove the PRINT#15,"I0" from line 80. Now run the program and change the name of a disk. LOAD"\$",8 and LIST. Although the disk name has been changed, you'll see the old name. If you remove the disk, turn the drive off, then back on, and load and list the directory, you'll see that the disk name *has* been changed. When the 1541 was initialized in line 10, the block availability map and disk header were read into a buffer. After the name change, you loaded the directory, but the disk drive looked at the two-letter ID and concluded that it didn't need to read the header again because it was working with the same disk as before. Thus, you saw the old name. The disk drive recognizes disks by their two-letter IDs. If you have several disks with the same ID, you may run into problems. Initializing the drive helps you avoid the difficulties associated with duplicate IDs.



## INPUT/OUTPUT ROUTINES

---

---

These eight lines make renaming disks as easy as renaming files.

### Disk Title Changer

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
SX 10 CLOSE15:OPEN15,8,15:PRINT#15,"IØ":CLOSE5:OP
    EN5,8,5,"#":B$=""
QQ 20 PRINT#15,"U1:5,Ø,18,Ø":PRINT#15,"B-P:5,144"
HJ 30 FORJ=1TO16:GET#5,A$:A$=CHR$(ASC(A$+CHR$(Ø)))
    :B$=B$+A$:NEXT
ER 40 PRINT:PRINT"DISK NAME: ";B$
EA 50 INPUT" NEW NAME";N$:IFLEN(N$)>16THENPRINT"MA
    XIMUM LENGTH IS 16":GOTO40
DS 60 IFLEN(N$)<16THENN$=N$+CHR$(16Ø):GOTO60
PM 70 PRINT#15,"B-P:5,144":PRINT#5,N$;:PRINT#15,"U
    2:5,Ø,18,Ø":CLOSE5
PP 80 INPUT#15,ER,ER$:PRINT"DISK STATUS:"ER;ER$:PR
    INT#15,"IØ":CLOSE15:END
```

# Tape Program Rescue

---

*John R. Hampton*

*This short machine language utility reads a program from tape into memory, allowing you to recover programs that have become unloadable. For the Commodore 64 and 128 in 64 mode.*

The Commodore Datassette is an inexpensive and generally reliable device on which to store programs and data. But sooner or later, you'll be unfortunate enough to combine a very long program with a bad tape. After saving the program, you won't be able to load it back into the computer. This is one reason to keep backup copies of all important programs.

What do you do if you don't have a backup and don't feel up to retyping the entire program? "Tape Program Rescue" may be the answer.

## **Fixing a Bad SAVE**

A program on tape can be scrambled or destroyed by a number of things: magnetism, a faulty coating on the tape, or ripped or creased tape. You may not be able to rescue the entire program, but you should be able to recover at least a portion, saving a lot of retyping time.

Tape Program Rescue is written in machine language (ML), but uses BASIC to POKE the program into memory. When you run it, the problem program is loaded from tape into memory, overwriting the BASIC part of Tape Program Rescue. For this reason, you should save Tape Program Rescue before you attempt to use it, or your typing will be lost.

To use it, load (but don't run) Tape Program Rescue. Then put the problem tape in the cassette drive and fast-forward the tape to a spot just before the beginning of the program to be rescued.

Now type RUN. The short ML program-rescue routine will be POKEd into memory, and you should see the CUT PROGRAM OFF AT LINE NUMBER? prompt. Enter a line number from the program. Tape Program Rescue will read up to, *but not including*, that line. The remaining lines will not

## INPUT/OUTPUT ROUTINES

---

load into memory. (You can also use this utility to delete the last portion of a program on a good tape.)

You may have to experiment a bit. If you can't load up to line 1100, try cutting off the program at 1000 or 740 or some smaller number. If it works, part of the previously unloadable program will have been loaded into memory and you can save that part to a good tape. You may still have to retype the last part of the program.

### Setting Up Memory

Tape Program Rescue loads programs into the same section of memory they were saved from. This means you can't use a 64 to rescue 128 programs, nor can you use a 128 to recover 64 programs. The memory configuration when you rescue must be the same as when you saved.

Also, the line number used for cutting off the program must be a real line number. If a program has lines 400 and 410, and you try to cut it off at 405, Tape Program Rescue will not operate properly.

### Tape Program Rescue

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
GA 100 A=681
KX 110 READ H$:IF H$="**" THEN 180
XH 120 FOR I=1 TO 2:J=ASC(MID$(H$,I,1))-48
SJ 130 IF J>9 THEN J=J-7
MH 140 IF I=1 THEN D=J*16
AR 150 IF I=2 THEN D=D+J
GP 160 NEXT:POKE A,D
SR 170 A=A+1:GOTO 110
XK 180 PRINT"CUT PROGRAM OFF AT":PRINT"LINE NUMBER
";:INPUT LN
FE 190 H=INT(LN/256):L=LN-H*256
BG 200 POKE 679,L:POKE 680,H
XS 210 SYS 681
CQ 220 DATA A9,00,AA,A8,20,BD,FF,E8,8A,20,BA,FF,98
,A6,2B,A4,2C,20,D5,FF,A6,2B,A4
MK 230 DATA 2C,86,FB,84,FC,A0,00,B1,FB,85,FD,C8,B1
,FB,85,FE,C8,B1,FB,CD,A7,02,D0
AK 240 DATA 08,C8,B1,FB,CD,A8,02,F0,07,A6,FD,A4,FE
,4C,C1,02,A0,00,98,91,FB,C8,91
JE 250 DATA FB,A5,FB,18,69,02,85,2D,A5,FC,69,00,85
,2E,00,**
```

# Stop and Go

---

*Jim Pejsa*

*When a program is running and the telephone rings, what do you do? This short machine language routine provides a pause button that temporarily halts the program. It also works on program listings. For the Commodore 64 and 128 in 64 mode.*

A very useful function found on many other computers is missing from Commodore computers. Many computers allow a programmer to stop and restart a program listing or run by using CTRL-S and CTRL-Q.

Pressing the CTRL key on the 64 or the Commodore key on the 128 will slow the listing down, but it's often necessary to stop the listing if you want to study a program carefully. Having to continue the listing by retyping LIST followed by some line numbers (if the program is long) is an inconvenience. Additionally, there are times when you want to stop a program (maybe to study some output) and restart it at will.

Since I was eager to try some machine language programming, I decided to write a program to add this feature. CTRL-S (for stop) and CTRL-Q (for restart) seem to be standard for these functions on many computers. The program is designed in such a way that the real time clock will continue to be updated while processing is stopped.

The machine language for the program is in the DATA statements and is loaded by the BASIC program. Simply type in and save the program and then run it to load the machine language. The program loads, beginning at location 679 (\$02A7), in some unused locations below the BASIC program area, so it will not interfere with any BASIC programs. The SYS statement in line 30 starts the machine language program. Unfortunately, if RUN/STOP-RESTORE is ever used to stop a program, you will find that the CTRL-S and CTRL-Q functions become inoperative. You can get them back by typing SYS 679.

Briefly, here's how the program works. When it is initially started with SYS 679 in BASIC, the hardware interrupt (IRQ) vector is changed to point to this program. The program is

## INPUT/OUTPUT ROUTINES

---

accessed each time the hardware interrupt occurs (every 1/60 second). The program checks for a CTRL-S keypress. If one is detected, it stops the listing or run and then checks for CTRL-Q. When CTRL-Q is pressed, the listing or run is restarted.

### Stop and Go

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
SC 10 FORI=679TO743:READ A:X=X+A:POKEI,A:NEXT
SX 20 IFX<>7291THENPRINT"ERROR IN DATA STATEMENTS.
      ":STOP
DD 30 SYS679
PD 40 DATA 120,169,180,141,20,3,169
DA 50 DATA 2,141,21,3,88,96,173
MX 60 DATA 141,2,201,4,208,42,165
FP 70 DATA 197,201,13,208,36,120,169
EX 80 DATA 49,141,20,3,169,234,141
XP 90 DATA 21,3,88,173,141,2,201
GM 100 DATA 4,208,249,165,197,201,62
DQ 110 DATA 208,250,120,169,180,141,20
RJ 120 DATA 3,169,2,141,21,3,76
AF 130 DATA 49,234
```

# Numeric Keypad

---

---

*Charles Kluepfel*

*Turn your keyboard into a "Numeric Keypad" for more efficient numeric input than the standard keyboard allows.*

*The program lets you toggle to standard keyboard or numeric keypad. For the Commodore 64.*

If your Commodore 64 had a numeric keypad, you could type in numbers much faster and with fewer errors than by using the standard number keys. Here's a program that offers this handy feature by redefining a set of keys to represent numbers instead of letters.

When you run "Numeric Keypad," your computer will behave normally until you press CTRL-N. Then, the cursor will disappear until you press another key. The M, J, K, L, U, I, and O keys will become, respectively, 0, 1, 2, 3, 4, 5, and 6. By using these along with the numeric keys 7, 8, and 9, you will have a numeric keypad. Pressing CTRL-N toggles the keyboard back into its normal mode (again causing the cursor to disappear until you press a key).

You can put press-apply transfer numbers on the affected keys to help you remember which number each key represents. Use very small ones so that they won't interfere with the normal identification of the keys. (Transfer letters and numbers are available at art supply stores.)

## **Using Numeric Keypad in a Program**

You can also activate and deactivate the numeric keypad from a program, in anticipation of numeric or non-numeric input, by POKEing location 50216 with 255 or 0, respectively. The user can always override this with CTRL-N. (CTRL-N is never passed to the program, but serves only the toggle function.) Just don't POKE any value other than 0 or 255, as that would prevent you from toggling with CTRL-N.

If you want the keypad to start out activated, change the next-to-last DATA item in line 520 from 0 to 255.

## INPUT/OUTPUT ROUTINES

---

### Redefining the Keys

To redefine the keys, we transfer the Kernal from ROM into RAM, change it to intercept the M, J, K, L, U, I, and O keys, and convert the data to the appropriate numbers.

Lines 3 and 4 POKE the machine language into an unused area of memory from the DATA statements in lines 500–560.

Lines 10 and 20 transfer the BASIC interpreter *and* the Kernal from ROM to RAM with the same addresses so that we can modify them. The *Commodore 64 Programmer's Reference Guide*, page 261, states that turning off bit 1 in location 1 switches only the Kernal addresses to RAM; actually, it affects both the Kernal and BASIC address ranges.

Line 25 merely signals that the transfer is complete. (It takes about a minute).

### The Intercept Routine

Line 30 sets up the routine that intercepts keyboard characters. It is put at the end of the routine that pulls a character from the keyboard buffer.

Finally, line 40 activates the modified Kernal by turning off bit 1 of location 1 (changing the value in location 1 from 55 to 53). Once this has been done, the change is made, and pressing CTRL-N toggles between a numeric keypad and the normal usage of the M, J, K, L, U, I, and O keys.

### A Color Memory Bonus

A couple of bonuses are included in lines 31 and 32. Line 31 changes the portion of the Kernal, on newer 64s, that puts the background color into the color memory for screen locations being cleared. Instead of putting the background color there, it will now put 1 (for white), so if addresses 1024–2023 (decimal) are POKEd, a character will appear.

### Choose a Color

In the normal mode, printed characters are light blue on a dark blue background, while POKEd characters are white. Change the POKE to location 58587 in line 31 to some other number if you would like a color different from white for POKEd screen characters. Of course, if you have an older 64

## CHAPTER 3

---

which does not clear color memory to the background color, leave out this patch (line 31).

Line 32 eliminates the printing of a question mark and space in an INPUT statement prompt. This makes it possible to write

```
100 INPUT "TITLE: ";T$
```

and have the resulting screen look like this:

```
TITLE:COMPUTE!'s GAZETTE
```

In any place where you really want the question mark and the space, you can put them inside the quotation marks.

### Numeric Keypad

*For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.*

```
EK 3 FORI=50176TO50261:READX:POKEI,X
DM 4 NEXT
FS 10 FORI=40960TO49151:POKEI,PEEK(I):NEXT
PG 20 FORI=57344TO65535:POKEI,PEEK(I):NEXT
CG 25 PRINT"TRANSFERRED"
AQ 30 POKE 58823,76:POKE58824,0:POKE58825,196
KJ 31 POKE58586,169:POKE58587,1:POKE58588,234
AM 32 FORI=44029TO44034:POKEI,234:NEXT
DJ 40 POKE 1,53
GA 500 DATA201, 14, 240, 65, 44, 40, 196, 240, 28,
      201, 85, 240, 40, 201
KA 510 DATA73, 240, 40, 201, 79, 240, 40, 201, 74,
      240, 16, 201, 75, 240
FR 520 DATA16, 201, 76, 240, 16, 201, 77, 240, 28,
      88, 24, 96, 0, 169
HK 530 DATA 49, 208, 248, 169, 50, 208, 244, 169,
      {SPACE}51, 208, 240, 169, 52, 208
EK 540 DATA 236, 169, 53, 208, 232, 169, 54, 208,
      {SPACE}228, 169, 48, 208, 224, 169
QP 550 DATA 255, 77, 40, 196, 141, 40, 196, 88, 16
      5, 198, 240, 252, 120, 76
BM 560 DATA 180, 229
```



---

---

# Appendices



## Appendix A

# How to Type In Programs

---

---

To make it easy for you to know exactly what to type when you're entering one of these programs into your computer, we have established the following listing conventions.

Generally, program listings contain words within braces that spell out any special characters: {DOWN}, for example, means to press the cursor-down key; {5 SPACES} means to press the space bar five times.

To indicate that a key should be *shifted* (hold down the SHIFT key while pressing the other key), the key is underlined in our listings. For example, S means to type the S key while holding the SHIFT key. This will appear on your screen as a heart symbol. If you find an underlined key enclosed within braces—for example, {10 N}—you should type the key as many times as indicated. (In this example, you would enter ten shifted N's).

If a key is enclosed within special brackets, [<>], you should hold down the *Commodore key* while pressing the key inside the special brackets. (The Commodore key is the key in the lower left corner of the keyboard.) Again, if the key is preceded by a number, you should press the key as many times as necessary.

Rarely, you'll see a solitary letter of the alphabet enclosed within braces. You can enter these characters by holding down the CTRL key while typing the letter inside the braces. For example, {A} indicates that you should press CTRL-A.

### Quote Mode

You know that you can move the cursor around the screen with the CRSR keys. Sometimes a programmer will want to move the cursor under program control. That's why you see all the {LEFT}'s, {HOME}'s, and {BLU}'s in our programs. The only way the computer can tell the difference between direct and programmed cursor control is the quote mode.

Once you have pressed the quote key (the double quotation mark, SHIFT-2), you are in the quote mode. If you type something and then try to change it by moving the cursor left, you'll only get a bunch of reverse-video lines. These are the

## APPENDIX A

symbols for cursor left. The only editing key that isn't programmable is the INST/DEL key; you can still use INST/DEL to back up and edit the line. Once you type another quote, you are out of quote mode.

You also go into quote mode when you insert spaces into a line. In any case, the easiest way to get out of quote mode is just to press RETURN. You'll then be out of quote mode and you can cursor up to the mistyped line and fix it.

In order to insure accurate entry of each program line, we have included an aid to typing in programs. Please use "The Automatic Proofreader" (Appendix B) for entering programs.

Refer to the following table when you're entering cursor and color control keys:

When You Read:	Press:	See:	When You Read:	Press:	See:
{CLR}	SHIFT CLR/HOME		{ F1 }	COMMODEORE 1	
{HOME}	CLR/HOME		{ F2 }	COMMODEORE 2	
{UP}	SHIFT ↑ CRSR ↓		{ F3 }	COMMODEORE 3	
{DOWN}	↑ CRSR ↓		{ F4 }	COMMODEORE 4	
{LEFT}	SHIFT ← CRSR →		{ F5 }	COMMODEORE 5	
{RIGHT}	← CRSR →		{ F6 }	COMMODEORE 6	
{RVS}	CTRL 9		{ F7 }	COMMODEORE 7	
{OFF}	CTRL 0		{ F8 }	COMMODEORE 8	
{BLK}	CTRL 1		{ F1 }	f1	
{WHT}	CTRL 2		{ F2 }	SHIFT f1	
{RED}	CTRL 3		{ F3 }	f3	
{CYN}	CTRL 4		{ F4 }	SHIFT f3	
{PUR}	CTRL 5		{ F5 }	f5	
{GRN}	CTRL 6		{ F6 }	SHIFT f5	
{BLU}	CTRL 7		{ F7 }	f7	
{YEL}	CTRL 8		{ F8 }	SHIFT f7	
			←		
			↑	SHIFT	

## Appendix B

# The Automatic Proofreader

---

---

*Philip I. Nelson*

“The Automatic Proofreader” helps you type in program listings and prevents nearly every kind of typing mistake.

Type in the Proofreader *exactly* as it’s listed. Since the program can’t check itself, type carefully to avoid mistakes. Don’t omit any lines, even if they contain unfamiliar commands. After finishing, save a copy or two on disk or tape before you run it. This is important because the Proofreader erases the BASIC portion of itself when you run it, leaving only the machine language portion in memory.

Next, type RUN and press RETURN. After announcing which computer it’s running on, the Proofreader displays the message “Proofreader Active.” Now you’re ready to type in a BASIC program.

### Using the Proofreader

Once the Proofreader is active, you can begin typing in a BASIC program as usual. Every time you finish typing a line and press RETURN, the Proofreader displays a two-letter checksum in the upper left corner of the screen. Compare this result with the two-letter checksum printed to the left of the corresponding line in the program listing. If the letters match, you can be almost certain the line has been typed correctly. If the letters don’t match, check for your mistake and correct the line.

The Proofreader ignores spaces that aren’t enclosed within quotation marks, so you can omit or add spaces between keywords and still see a matching checksum. However, since spaces inside quotation marks are almost always significant, the Proofreader pays attention to them. For example,

```
10 PRINT"THIS IS BASIC"
```

will generate a different checksum than does

```
10 PRINT"THIS ISBA SIC"
```

A common typing error is transposition—typing two successive characters in the wrong order, like PIRNT instead

## APPENDIX B

---

---

of PRINT or 64378 instead of 64738. The Proofreader is sensitive to the *position* of each character within the line and thus catches transposition errors.

The Proofreader does *not* accept keyword abbreviations (for example, ? instead of PRINT). If you prefer to use abbreviations, you can still check the line: LIST it after you've typed it in, move the cursor back to the line, and press RETURN. LISTing the line substitutes the full keyword for the abbreviation and allows the Proofreader to work properly. The same technique works for rechecking programs you've already typed in.

If you're using the Proofreader on the 128, *do not perform any GRAPHIC commands while the Proofreader is active*. When you perform a command like GRAPHIC 1, the computer moves everything at the start of BASIC program space—including the Proofreader—to another memory area, causing the Proofreader to crash. The same thing happens if you *run* any program with a GRAPHIC command while the Proofreader is in memory.

Though the Proofreader doesn't interfere with other BASIC operations, it's a good idea to disable it before running another program. However, the Proofreader is purposely difficult to dislodge: It's not affected by tape or disk operations, or by pressing RUN/STOP-RESTORE. The simplest way to disable it is to turn the computer off and then on again. A gentler method is to SYS to the built-in reset routine (SYS 64738 for the 64; SYS 65341 for the 128). This reset routine erases any program in memory, so be sure to save the program you're typing in before entering the SYS command.

If you own a Commodore 64, you may wonder whether the Proofreader works with other programming utilities like "MetaBASIC." The answer is generally yes, *if you activate the Proofreader after installing the other utility*. For example, first load and activate MetaBASIC; then load and run the Proofreader.

When using the Proofreader with another utility, you should disable *both* programs before running a BASIC program. While the Proofreader seems unaffected by most utilities, there's no way to promise that it will work with any and every combination of utilities you might want to use. The more utilities activated, the more fragile the system becomes.

## APPENDIX B

---

### The Automatic Proofreader

```
10 VEC=PEEK(772)+256*PEEK(773):LO=43:HI=44
20 PRINT "AUTOMATIC PROOFREADER FOR ";:IF VEC=4236
4 THEN PRINT "C-64"
30 IF VEC=50556 THEN PRINT "VIC-20"
40 IF VEC=35158 THEN GRAPHIC CLR:PRINT "PLUS/4 & 1
6"
50 IF VEC=17165 THEN LO=45:HI=46:GRAPHIC CLR:PRINT
"128"
60 SA=(PEEK(LO)+256*PEEK(HI))+6:ADR=SA
70 FOR J=0 TO 166:READ BYT:POKE ADR,BYT:ADR=ADR+1:
CHK=CHK+BYT:NEXT
80 IF CHK<>20570 THEN PRINT "*ERROR* CHECK TYPING
{SPACE}IN DATA STATEMENTS":END
90 FOR J=1 TO 5:READ RF,LF,HF:RS=SA+RF:HB=INT(RS/2
56):LB=RS-(256*HB)
100 CHK=CHK+RF+LF+HF:POKE SA+LF,LB:POKE SA+HF,HB:N
EXT
110 IF CHK<>22054 THEN PRINT "*ERROR* RELOAD PROGR
AM AND CHECK FINAL LINE":END
120 POKE SA+149,PEEK(772):POKE SA+150,PEEK(773)
130 IF VEC=17165 THEN POKE SA+14,22:POKE SA+18,23:
POKESA+29,224:POKESA+139,224
140 PRINT CHR$(147);CHR$(17);"PROOFREADER ACTIVE":
SYS SA
150 POKE HI,PEEK(HI)+1:POKE (PEEK(LO)+256*PEEK(HI)
)-1,0:NEW
160 DATA 120,169,73,141,4,3,169,3,141,5,3
170 DATA 88,96,165,20,133,167,165,21,133,168,169
180 DATA 0,141,0,255,162,31,181,199,157,227,3
190 DATA 202,16,248,169,19,32,210,255,169,18,32
200 DATA 210,255,160,0,132,180,132,176,136,230,180
210 DATA 200,185,0,2,240,46,201,34,208,8,72
220 DATA 165,176,73,255,133,176,104,72,201,32,208
230 DATA 7,165,176,208,3,104,208,226,104,166,180
240 DATA 24,165,167,121,0,2,133,167,165,168,105
250 DATA 0,133,168,202,208,239,240,202,165,167,69
260 DATA 168,72,41,15,168,185,211,3,32,210,255
270 DATA 104,74,74,74,74,168,185,211,3,32,210
280 DATA 255,162,31,189,227,3,149,199,202,16,248
290 DATA 169,146,32,210,255,76,86,137,65,66,67
300 DATA 68,69,70,71,72,74,75,77,80,81,82,83,88
310 DATA 13,2,7,167,31,32,151,116,117,151,128,129,
167,136,137
```

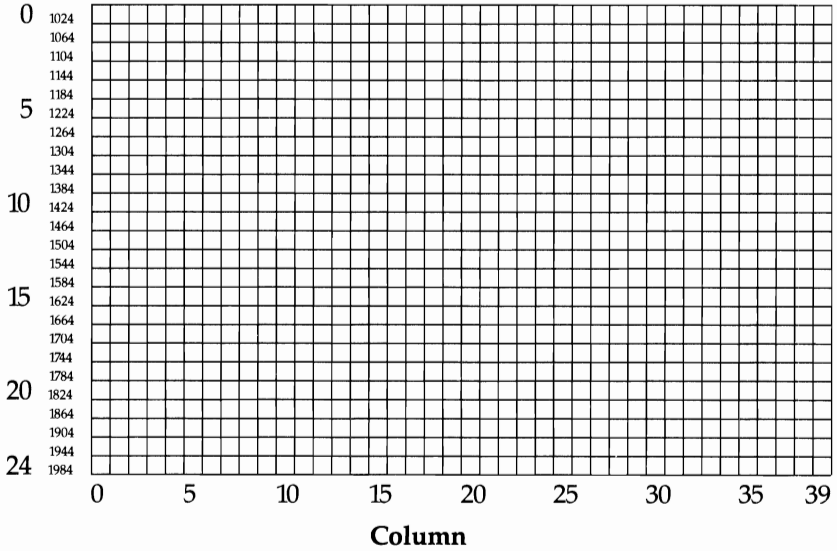
# Appendix C

## Screen Location Table

---

---

Row







# Appendix E

## 40-Column Screen Color Codes

---

---

The following are the color codes that can be POKEd directly into a color register on either the 64 or the 128. For example, to set the screen background color on the 64, you would type

**POKE 53281,x**

where *x* is the color value from the table below.

With BASIC 7.0 on the 128, you don't have to POKE color values into registers to set screen or character colors; you can use the COLOR statement instead. The color values used with the COLOR statement are slightly different from those listed below. You must add one to the value listed below to set a register to the indicated color. For example, to set the background to blue with the COLOR statement, you would enter COLOR 0,7. The 0 tells BASIC which color register to change, and the 7 tells it to set that register to blue. For more information on the COLOR statement, see *COMPUTE!'s 128 Programmer's Guide* (COMPUTE! Books, 1985).

Color	Color Value
Black	0
White	1
Red	2
Cyan	3
Purple	4
Green	5
Blue	6
Yellow	7
Orange	8
Brown	9
Light red	10
Dark gray	11
Medium gray	12
Light green	13
Light blue	14
Light gray	15

# Appendix F

## Standard ASCII Codes

---

The most widely recognized character code is ASCII (American Standard Code for Information Interchange). ASCII is a seven-bit code and thus can represent  $2^7$  (128) different characters. CP/M mode uses standard ASCII (although it does add special definitions for the additional characters 128–255).

Most microcomputer telecommunication is also conducted in ASCII. Unfortunately, neither of the two character sets in 128 or 64 mode follows the ASCII standard exactly, so unless you are communicating with another Commodore computer, you'll have to translate the characters you send into ASCII and translate those you receive back into Commodore character codes.

Dec	Hex	Character	Dec	Hex	Character
0	00	NUL	25	19	EM
1	01	SOH	26	1A	SUB
2	02	STX	27	1B	ESC
3	03	ETX	28	1C	FS
4	04	EOT	29	1D	GS
5	05	ENQ	30	1E	RS
6	06	ACK	31	1F	US
7	07	BEL	32	20	space
8	08	BS	33	21	!
9	09	HT	34	22	"
10	0A	LF	35	23	#
11	0B	VT	36	24	\$
12	0C	FF	37	25	%
13	0D	CR	38	26	&
14	0E	SO	39	27	'
15	0F	SI	40	28	(
16	10	DLE	41	29	)
17	11	DC1	42	2A	*
18	12	DC2	43	2B	+
19	13	DC3	44	2C	,
20	14	DC4	45	2D	-
21	15	NAK	46	2E	.
22	16	SYN	47	2F	/
23	17	ETB	48	30	0
24	18	CAN	49	31	1

## APPENDIX F

---

---

Dec	Hex	Character	Dec	Hex	Character
50	32	2	89	59	Y
51	33	3	90	5A	Z
52	34	4	91	5B	[
53	35	5	92	5C	\
54	36	6	93	5D	]
55	37	7	94	5E	^
56	38	8	95	5F	~
57	39	9	96	60	
58	3A	:	97	61	a
59	3B	;	98	62	b
60	3C	<	99	63	c
61	3D	=	100	64	d
62	3E	>	101	65	e
63	3F	?	102	66	f
64	40	@	103	67	g
65	41	A	104	68	h
66	42	B	105	69	i
67	43	C	106	6A	j
68	44	D	107	6B	k
69	45	E	108	6C	l
70	46	F	109	6D	m
71	47	G	110	6E	n
72	48	H	111	6F	o
73	49	I	112	70	p
74	4A	J	113	71	q
75	4B	K	114	72	r
76	4C	L	115	73	s
77	4D	M	116	74	t
78	4E	N	117	75	u
79	4F	O	118	76	v
80	50	P	119	77	w
81	51	Q	120	78	x
82	52	R	121	79	y
83	53	S	122	7A	z
84	54	T	123	7B	{
85	55	U	124	7C	
86	56	V	125	7D	}
87	57	W	126	7E	~
88	58	X	127	7F	DEL

## APPENDIX F

---

---

### Notes

1. Codes 0–31 are transmission control codes, and only a few are commonly used in microcomputer telecommunications. These include character 13, the carriage-return (RETURN) character, and character 27, the ESCape character (which is frequently used to indicate a command). Some non-Commodore computers require that RETURN be followed by character 10, the linefeed character. Many terminal programs recognize character 19 as XOFF, a signal to halt transmission temporarily. Transmission resumes when character 17, also known as XON, is received. File transfer protocols such as XMODEM use some of the other control codes.
2. For character 94, Commodore uses the up-arrow (↑) in place of the caret (^).

# Appendix G

## Screen Codes

---

---

There are 256 screen codes for each character set; codes 128–255 are the reverse images of codes 0–127. To display any character in reverse video, simply add 128 to its screen code value. Thus, POKE 1024,1:POKE 1025,1+128 displays an A and a reverse A.

























The character ROM has a total of 153 different characters. In the uppercase/graphics set, the character patterns for codes 32 and 96 are identical, as are those for 64 and 67, 66 and 93, 101 and 116, and 103 and 106.

Dec	Hex	Uppercase/Graphics Set	Lowercase/Uppercase Set
0	00	@	@
1	01	A	a
2	02	B	b
3	03	C	c
4	04	D	d
5	05	E	e
6	06	F	f
7	07	G	g
8	08	H	h
9	09	I	i
10	0A	J	j
11	0B	K	k
12	0C	L	l
13	0D	M	m
14	0E	N	n
15	0F	O	o
16	10	P	p
17	11	Q	q
18	12	R	r
19	13	S	s
20	14	T	t

## APPENDIX G

Dec	Hex	Uppercase/Graphics Set	Lowercase/Uppercase Set
21	15	U	u
22	16	V	v
23	17	W	w
24	18	X	x
25	19	Y	y
26	1A	Z	z
27	1B	[	[
28	1C	£	£
29	1D	]	]
30	1E	↑	↑
31	1F	←	←
32	20		space
33	21	!	!
34	22	“	“
35	23	#	#
36	24	\$	\$
37	25	%	%
38	26	&	&
39	27	,	,
40	28	(	(
41	29	)	)
42	2A	*	*
43	2B	+	+
44	2C	,	,
45	2D	-	-
46	2E	.	.
47	2F	/	/
48	30	0	0
49	31	1	1
50	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5

## APPENDIX G

Dec	Hex	Uppercase/Graphics Set	Lowercase/Uppercase Set
54	36	6	6
55	37	7	7
56	38	8	8
57	39	9	9
58	3A	:	:
59	3B	;	;
60	3C	<	<
61	3D	=	=
62	3E	>	>
63	3F	?	?
64	40		
65	41		A
66	42		B
67	43		C
68	44		D
69	45		E
70	46		F
71	47		G
72	48		H
73	49		I
74	4A		J
75	4B		K
76	4C		L
77	4D		M
78	4E		N
79	4F		O
80	50		P
81	51		Q
82	52		R
83	53		S
84	54		T
85	55		U
86	56		V



















## APPENDIX G

Dec	Hex	Uppercase/Graphics Set	Lowercase/Uppercase Set
87	57		W
88	58		X
89	59		Y
90	5A		Z
91	5B		
92	5C		
93	5D		
94	5E		
95	5F		
96	60	SHIFT-space	
97	61		
98	62		
99	63		
100	64		
101	65		
102	66		
103	67		
104	68		
105	69		
106	6A		
107	6B		
108	6C		
109	6D		
110	6E		
111	6F		
112	70		
113	71		
114	72		
115	73		
116	74		
117	75		
118	76		
119	77		

## APPENDIX G

---

---

Dec	Hex	Uppercase/Graphics Set	Lowercase/Uppercase Set
120	78		
121	79		
122	7A		
123	7B		
124	7C		
125	7D		
126	7E		
127	7F		

# Appendix H

## Commodore 64 and 128

### Keycodes

---

---

The 128 keyboard is arranged electrically as a matrix of 11 columns  $\times$  8 rows of keys; for the Commodore 64 it's arranged as 8 columns  $\times$  8 rows. Every 1/60 second, the computer scans the keyboard to see whether a key is pressed. If a key is pressed, the keyscan routine generates a value that corresponds to the key's position in the matrix. (The formula is  $\text{keycode} = \text{column} * 8 + \text{row}$ , where *column* has a value from 0 through 10, and *row* has a value from 0 through 7.) If no key is pressed, a value of 88 is generated. (The "no key pressed" value for the 64 is 64.) This value is stored in location 212 (location 203 for the 64). Location 213 (location 197 for the 64) will also hold the same value. The contents of this location can be used to determine which key is currently being pressed, as an alternative to using GET (or the Kernal GETIN routine in machine language). For example, these two lines have the the same effect—to pause the program until any key is pressed:

```
100 GET K$:IF K$="" THEN 100
100 IF PEEK(212)=88 THEN 100
```

The accompanying figure gives the keyscan codes for the 128 keyboard. Notice that the figure shows no values for the left and right SHIFT keys, the Commodore key, CONTROL, and ALT. They are the keys that use the missing codes 15, 52, 58, 61, and 80, respectively. These keys are detected later in the keyscan routine, and another location is used to record their status. In 128 mode the location used is 211; the corresponding location for the 64 is 653. Here are the values found there:

No shift key pressed	0
SHIFT	1
Commodore	2
CONTROL	4
ALT (128 mode only)	8
CAPS LOCK (128 mode only)	16

## APPENDIX H

---

Note that for this location the values are cumulative. If you press both SHIFT and CONTROL, the location will contain  $1 + 4 = 5$ . Holding down SHIFT, Commodore, CONTROL, and ALT together would result in a value of 15.

The other "missing" keys are not part of the keyscan matrix. RESTORE is connected to the CIA 2 chip and acts by generating an NMI interrupt. The 40/80 column key is connected to the MMU chip and is read and acted upon only at power on or reset. The CAPS LOCK key is connected to bit 6 of the 8502 chip's built-in data port and thus is read via location 1. PRINT (PEEK(1) AND 64) will show its status (0 = down, 64 = up). The SHIFT LOCK key is not scanned; it's merely a switch that has the effect of holding down the SHIFT key.

# APPENDIX H

## Commodore 64 and 128 Keycodes

ESC	TAB	ALT	CAPS LOCK	HELP	LINE FEED	40/80 Display	NO Scroll	↑	↓	←	→	F1	F3	F5	F7
72	67	-	-	64	75	-	87	83	84	85	86	4	5	6	3

←	→	£	CLR HOME	INST DEL
83	84	85	51	0

↑	↓	←	→
83	84	85	86

←	1	2	3	4	5	6	7	8	9	+					
57	56	59	8	11	16	19	24	27	32	35	40	43	48	51	0

CONTROL	Q	W	E	R	T	Y	U	I	O	P	@	*	↑	RESTORE
-	62	9	14	17	22	25	30	33	38	41	46	49	54	-
RUN STOP	SHIFT LOCK	A	S	D	F	G	H	J	K	L	:	;	=	RETURN
63	-	10	13	18	21	26	29	34	37	42	45	50	53	I
⌂	SHIFT	Z	X	C	V	B	N	M	,	.	/	SHIFT	CRSR	CRSR
-	-	12	23	20	31	28	39	36	47	44	55	-	↑	↓
													7	2
													81	82
													0	•
													7	82
													81	76

60



# Index

---

---

- @ *See* at sign
- £ *See* British pound sign
- # *See* number sign
- ASCII (American Standard Code for Information Interchange) 139
  - codes 139–40
  - values 32
- “ASCII/POKE Printer” program listing 34
- at sign 29
- “Auto Line Numbering” program listing 36
- “BASIC Line Extender” program listing 19
- blink locations (table) 46
- “Blick—64 Version” program listing 47
- “Blick—128 Version” program listing 47
- “Blink Mode—Demo 1” program listing 61–62
- “Blink Mode—Demo 2” program listing 63
- “Blink Mode” program listing 60–61
- block availability map (BAM) 118
- British pound sign 17
- cassette buffer 110, 112
- colon 17
- color swap locations (table) 78
- “Color Swap” program listing 79–81
- Commodore 64 Programmer’s Reference Guide* 127
- Commodore 64 and 128 keycodes* 149
- COMPUTE!’s 128 Programmer’s Guide* 138
- “Controlled Keyboard Input” program listing 51–53
- custom character set 72
- DATA statements 10
- disk operating system (DOS) 118
- “Disk Title Changer” program listing 121
- extension 17
- 40-column screen color codes 138
- hardware interrupt (IRQ) vector 124
- “Help Screens—64 Version” program listing 41–42
- “Help Screens—128 Version” program listing 42–44
- high-resolution graphics 89
- “Hi-Res Screen Dump” program listing 91–92
- “Input Windows Demo” program listing 53
- “Instant Keywords” program listing 8
- interrupt wedge 56
- jiffy 82
- jiffy clock 82
- joystick 104, 114
  - directions (figure) 116
- “Keyboard to Joystick Converter” program listing 106–07
- key combinations (table) 7
- keywords 6–8
- “Line Count—64 Version” program listing 11–12
- “Line Count—128 Version” program listing 12–13
- line number 35
- “List Pager” program listing 113
- lookup bug 46
- number sign 111
- “Numeric Keypad” program listing 128
- paging 109
- POKE values 32
- “Printmaker” program listing 65–66
- Programming the 64* 115
- “Quick Character Transfer” program listing 73–74
- “QuickScan” program listing 71
- quotation marks 17, 111
- RAM disk 101
- ROM, version 2 46
- screen codes 142–46
- screen color memory (table) 137
- “Screen Customizer” program listing 69
- “Screen Headliner” program listing 96–97
- screen location (table) 136
- “Searchlight” program listing 22
- semicolon 49
- “64/128 Program Mis-Matcher” program listing 4–5
- “64 RAM Disk” program listing 103
- “Slowpoke” program listing 88
- “Sound Off” program listing 15
- “Step Lister” program listing 30–31
- “Stop and Go” program listing 125

string arrays 25  
"String Search—BASIC Loader"  
  program listing 27  
"String Search—Demo Program"  
  program listing 28  
syntax error 20  
"Tape Program Rescue" program listing  
  123  
"The Automatic Proofreader" 67  
  program listing 135  
"Time Clock" program listing 83–84  
time-of-day (TOD) clock 83  
tokens 6, 29  
TRAP statements 14  
"Triple 64" program listing 24  
typing in programs 131–32  
USR function 54–55, 112–15  
"USR Joystick Reader" program listing  
  116  
vector 86  
video blink mode 54  
wedge 29–30, 46, 112  
window 48  
"Worm Demo" program listing 107–08



To order your copy of *COMPUTE!'s Commodore 64/128 Power BASIC Disk*, call our toll-free US order line: 1-800-346-6767 (in NY 212-887-8525) or send your prepaid order to:

*COMPUTE!'s Commodore 64/128 Power BASIC Disk*

**COMPUTE!** Publications

P.O. Box 5038

F.D.R. Station

New York, NY 10150

All orders must be prepaid (check, charge, or money order). NC residents add 5% sales tax. NY residents add 8.25% sales tax.

Send \_\_\_\_\_ copies of *COMPUTE!'s Commodore 64/128 Power BASIC Disk* at \$12.95 per copy. (998BDSK)

Subtotal \$ \_\_\_\_\_

Shipping and Handling: \$2.00/disk \$ \_\_\_\_\_

Sales tax (if applicable) \$ \_\_\_\_\_

Total payment enclosed \$ \_\_\_\_\_

Payment enclosed

Charge  Visa  MasterCard  American Express

Acct. No. \_\_\_\_\_ Exp. Date \_\_\_\_\_  
(Required)

Name \_\_\_\_\_

Address \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ Zip \_\_\_\_\_

Please allow 4-5 weeks for delivery.





# Power BASIC

## for the Commodore 64 and 128

Here is a tremendously useful selection of utilities for the Commodore 64 and 128 home computers. *Commodore 64/128 Power BASIC* provides you with an assortment of 32 routines that you'll use again and again to speed up programming tasks and to increase your programming efficiency. You'll find convenient solutions to problems programmers face every day, and will be able to devote more of your time to your own projects.

Here are just a few of the programs you'll find inside:

- "QuickScan" is great for keeping your place while you're scanning a long program listing. It sets up a bar that automatically moves up and down the screen, highlighting screen lines to mark your place.
- "Numeric Keypad" turns your standard Commodore 64 keyboard into a numeric keypad for fast and easy number entry.
- "Keyboard to Joystick Converter" allows you to use a joystick with programs that don't provide for joystick control.
- "Disk Title Changer" renames disks quickly and easily.
- "Stop and Go" provides a pause button that's very useful when you want to halt a program temporarily.
- "Program Mis-Matcher" compares two BASIC programs. It's convenient for examining different versions of the same program.

And these are just the beginning. With "The Automatic Proofreader" to help you type in listings correctly, you'll be able to put these routines to work in no time. In addition, appendices give supplementary information about screen color memory, screen color codes, ASCII codes, and much more. *Commodore 64/128 Power BASIC* will open up new possibilities and give you greater control over your computer than ever before.

*The programs included in this book are available on a companion disk. See the coupon in the back for details.*