152K *Lowest Price In The USA!* 152K

ATARI® Computer System Sale

• Students • Word Processing • Home • Business

LOOK AT ALL YOU GET FOR ONLY **$399** SYSTEM PRICE

LIMITED QUANTITIES

	LIST PRICE	INDIVIDUAL SALE PRICE
① Atari 130XE 152K Computer	$249.00	$134.95
② Atari 1050 127K Disk Drive	299.00	179.95
③ Atari 1027 Letter Quality 20 CPS Printer	299.00	199.95
Letter Perfect Word Processer	59.95	39.95
Atari BASIC Tutorial Manual	16.95	12.95
TOTALS	**$923.90**	**$567.75**

All connecting cables & T.V. interface included.
Monitors sold separetly.

SAVE OVER $100
All 5 ONLY
$399.00
SYSTEM SALE PRICE

Other Accessories	List	Sale
☆ 12" Hi Resolution Amber Screen Monitor	$199.00	**$59.95**
☆ 13" Hi Resolution Color Monitor	$399.00	**$159.95**

Add $9.95 for Connection Cables (Monitors Only) (Ltd. Qty)
Add $10 for UPS

15 DAY FREE TRIAL. We give you 15 days to try out this ATARI COMPUTER SYSTEM!! If it doesn't meet your expectations, just send it back to us prepaid and we will refund your purchase price!! **90 DAY IMMEDIATE REPLACEMENT WARRANTY**. If any of the ATARI COMPUTER SYSTEM equipment or programs fail due to faulty workmanship or material within 90 days of purchase we will replace it IMMEDIATELY with no service charge!!

Best Prices • Over 1000 Programs and 500 Accessories Available • Best Service • One Day Express Mail • Programming Knowledge • Technical Support

Add $25.00 for shipping and handling!!
Enclose Cashiers Check, Money Order or Personal Check. Allow 14 days for delivery. 2 to 7 days for phone orders. 1 day express mail! We accept Visa and MasterCard. We ship C.O.D. to continental U.S. addresses only. Add $10 more if C.O.D.

COMPUTER DIRECT

We Love Our Customers
22292 N. Pepper Rd., Barrington, Ill. 60010
312/382-5050 to order

any value of MEMLO. That explains all the IF-THEN statements which make the program look so strange.

To use Atari Keypad, try the version created by Program 1 first. If that doesn't work, try the other version. Use the second version with the Automatic Proofreader.

If you already have an AUTORUN.SYS file on your disk that you regularly use, you can append it to the Atari Keypad AUTORUN.SYS file so both will boot automatically. Follow these steps:

1. Boot up Atari DOS 2.0 or 2.5.

2. Rename your existing AUTORUN.SYS file. For example, call it OLDAUTO.

3. Exit to BASIC and run either Program 1 or Program 2 to create the keypad AUTORUN.SYS file on disk.

4. Enter DOS and select the COPY option. When the prompt appears, type OLDAUTO,AUTORUN.SYS/A. Don't forget the /A or you'll end up with your old AUTORUN.SYS file and have to start over again.

If your existing AUTORUN.SYS file happens to use the same memory as Atari Keypad, it would be overwritten when the keypad is booted. Another problem could crop up if your present AUTORUN.SYS file installs a routine at MEMLO and the routine isn't relocatable. If the keypad is installed at MEMLO first, the second routine would wind up at a different address than it was designed for. This would most likely cause the system to crash. Most of the time there's no trouble, however.

If you can touch-type on a keypad, you'll find Atari Keypad a great aid when entering DATA statements. But don't forget it can also be useful with other programs that call for numeric input.

For instructions on entering these listings, please refer to "COMPUTE!'s Guide to Typing In Programs" published bimonthly in COMPUTE!.

Program 1: Atari Keypad For Page 6

```
PA 10 OPEN #1,8,0,"D:AUTORUN
      .SYS"
EK 20 FOR X=1 TO 170
LE 30 READ A:PUT #1,A
PL 40 NEXT X
CP 50 CLOSE #1
DN 60 END
AM 1000 DATA 255,255,128,6,2
      38,6,32,128,6,120
HB 1010 DATA 173,8,2,141,165
      ,6,173,9,2,141
QE 1020 DATA 166,6,169,156,1
      41,8,2,169,6,141
JK 1030 DATA 9,2,88,96,169,6
      ,72,169,167,72
KN 1040 DATA 8,8,76,164,6,17
      3,252,2,201,227
AH 1050 DATA 208,12,174,238,
      6,208,3,232,208,1
GJ 1060 DATA 202,142,238,6,1
      74,238,6,240,42,201
NJ 1070 DATA 37,208,2,169,50
      ,201,1,208,2,169
MH 1080 DATA 31,201,5,208,2,
      169,30,201,0,208
AE 1090 DATA 2,169,26,201,11
      ,208,2,169,24,201
NK 1100 DATA 13,208,2,169,29
      ,201,8,208,2,169
CO 1110 DATA 27,141,252,2,10
      4,64,0,0,6,42
DB 1120 DATA 6,165,12,141,12
      9,6,165,13,141,130
DK 1130 DATA 6,169,128,133,1
      2,169,6,133,13,120
HF 1140 DATA 173,8,2,141,165
      ,6,173,9,2,141
OI 1150 DATA 166,6,169,156,1
      41,8,2,169,6,141
LH 1160 DATA 9,2,88,96,226,2
      ,227,2,0,6
```

Program 2: Atari Keypad For Low Memory

```
KG 10 START=4+PEEK(743)+PEEK
      (744)*256
PB 20 OPEN #1,8,0,"D:AUTORUN
      .SYS"
DO 30 FOR I=1 TO 190
DG 40 READ X:IF I=3 THEN X=S
      TART-INT(START/256)*25
      6
AI 50 IF I=4 THEN X=INT(STAR
      T/256)
KK 60 IF I=5 THEN X=(START+1
      20)-INT((START+120)/25
      6)*256
BL 70 IF I=6 THEN X=INT((STA
      RT+120)/256)
IN 80 IF I=15 THEN X=(START+
      47)-INT((START+47)/256
      )*256
CG 90 IF I=16 THEN X=INT((ST
      ART+47)/256)
LF 100 IF I=21 THEN X=(START
      +48)-INT((START+48)/2
      56)*256
EN 110 IF I=22 THEN X=INT((S
      TART+48)/256)
LI 120 IF I=24 THEN X=(START
      +38)-INT((START+38)/2
      56)*256
FF 130 IF I=29 THEN X=INT((S
      TART+38)/256)
BM 140 IF I=35 THEN X=(START
      +128)-INT((START+128)
      /256)*256
IA 150 IF I=40 THEN X=INT((S
      TART+128)/256)
FJ 160 IF I=46 THEN X=INT((S
      TART+49)/256)
MI 170 IF I=49 THEN X=(START
      +49)-INT((START+49)/2
      56)*256
HN 180 IF I=135 THEN X=(STAR
      T+1)-INT((START+1)/25
      6)*256
FA 190 IF I=136 THEN X=INT((
      START+1)/256)
HE 200 IF I=140 THEN X=(STAR
      T+2)-INT((START+2)/25
      6)*256
EG 210 IF I=141 THEN X=INT((
      START+2)/256)
BN 220 IF I=143 THEN X=START
      -INT(START/256)*256
KA 230 IF I=147 THEN X=INT(S
      TART/256)
PA 240 IF I=155 THEN X=(STAR
      T+47)-INT((START+47)/
      256)*256
IJ 250 IF I=156 THEN X=INT((
      START+47)/256)
PB 260 IF I=161 THEN X=(STAR
      T+48)-INT((START+48)/
      256)*256
IJ 270 IF I=162 THEN X=INT((
      START+48)/256)
PE 280 IF I=164 THEN X=(STAR
      T+38)-INT((START+38)/
      256)*256
JB 290 IF I=169 THEN X=INT((
      START+38)/256)
EP 300 IF I=175 THEN X=(STAR
      T+128)-INT((START+128
      )/256)*256
LD 310 IF I=180 THEN X=INT((
      START+128)/256)
CI 320 PUT #1,X:NEXT I
GA 330 CLOSE #1
GO 340 END
ML 1000 DATA 255,255,0,29,12
      0,29,32,0,29,120
HF 1010 DATA 173,8,2,141,47,
      29,173,9,2,141
OM 1020 DATA 48,29,169,38,14
      1,8,2,169,29,141
OJ 1030 DATA 9,2,88,169,128,
      141,231,2,169,29
CD 1040 DATA 141,232,2,96,16
      9,29,72,169,49,72
LC 1050 DATA 8,8,76,46,29,17
      3,252,2,201,227
DD 1060 DATA 208,12,174,120,
      29,208,3,232,208,1
MA 1070 DATA 202,142,120,29,
      174,120,29,240,42,20
      1
NK 1080 DATA 37,208,2,169,50
      ,201,1,208,2,169
MI 1090 DATA 31,201,5,208,2,
      169,30,201,0,208
PM 1100 DATA 2,169,26,201,11
      ,208,2,169,24,201
NL 1110 DATA 13,208,2,169,29
      ,201,8,208,2,169
DA 1120 DATA 27,141,252,2,10
      4,64,0,0,6,52
JK 1130 DATA 6,165,12,141,1,
      29,165,13,141,2
DK 1140 DATA 29,169,0,133,12
      ,169,29,133,13,120
HK 1150 DATA 173,8,2,141,47,
      29,173,9,2,141
PB 1160 DATA 48,29,169,38,14
      1,8,2,169,29,141
OO 1170 DATA 9,2,88,169,128,
      141,231,2,169,29
DN 1180 DATA 141,232,2,96,22
      6,2,227,2,0,6
```

©

Million-Color Palette For IBM PC & PCjr

John Klein and Jeff Klein

It's amazing but true—with this stunning technique you can generate more than a million apparent color variations on a PCjr. You can even display 256 colors simultaneously. The effects are less dramatic on a PC, but it's still possible to generate many more than the standard 16 colors. The programs require an Enhanced Model PCjr or a PC with color/graphics card, plus a TV set or composite color monitor. The palette is more limited on an RGB monitor, but still impressive.

No longer is your PC or PCjr restricted to a palette of 16 colors and the inability to display them all in higher resolutions. Now you can choose to display 256 colors from a palette of over 1,000,000 colors in high resolution, and display an entire palette of 256 colors in medium resolution. And each color is distinct and solid.

The secret is a combination of a technique called *tile painting* and the trick of fooling a TV or composite monitor into displaying new solid colors. To understand how it works, let's examine the way graphics are stored, changed, and displayed on the IBM video screen.

A Byte Of Pixels

Graphics images are stored differently in the computer's memory for each different graphics mode or screen. In its simplest form, the color of each *pixel*—the smallest controllable dot on the screen—is stored in a section of memory. This video memory is arranged by its location or coordinates on the screen. The image you see on the screen, therefore, is a copy of the contents of video memory. (Actually, screens are divided into several layers when stored in memory, but that's not important for this discussion; we're concerned with how the colors of pixels are represented in memory, not how each pixel is arranged.)

To figure out how many pixels can be represented in a byte of memory, remember that a byte is made up of eight bits, and a bit is the smallest unit of memory (a bit is either a zero or a one). Simply divide the amount of memory required for a certain screen mode by the number of pixels on the screen. The memory requirements for each screen mode are shown in Table 1.

Remember that RGB stands for the three primary colors of light: red, green, and blue. All colors can be made by mixing these three primary colors. That's why RGB monitors, color TVs, and composite color monitors have three electron guns inside their picture tubes, instead of the single gun found in black and white TVs and monochrome monitors. There is a red gun, a green gun, and a blue gun, all of which are controlled by the computer to produce color. If none of the guns is lighting a pixel, the pixel appears black.

Colors are represented in memory by arranging bits to denote which electron guns should be turned on or off when lighting the corresponding pixel. For instance, if a certain pixel is supposed to be blue, the group of bits representing that pixel in memory shows the blue gun is on and the others off. (A bit set to 1 means on, and 0 means off.) All the possible combinations of the three electron guns account for eight colors. To get eight more colors, the intensity, also called *luminance*, is varied by mixing a little white with the first eight colors. That's why the IBM PC and PCjr have a total of 16 color variations: two shades each of eight colors.

Table 2 shows how each of the 16 colors is represented. Remember that each bit turns an electron gun either on or off. Notice how many bits it takes to represent all the

Table 1: Screen Mode Memory Requirements

Screen Mode	Resolution	Number of Colors	Memory per Screen	Pixels/Byte	Bits/Pixel
1	320 × 200	4	16K*	4	2
2	640 × 200	2	16K	8	1
3	160 × 200	16	16K	2	4
4	320 × 200	4	16K	4	2
5	320 × 200	16	32K	2	4
6	640 × 200	4	32K	4	2

*1K = 1024 bytes

possible combinations. It takes four bits, or half of a byte (sometimes called a *nybble*) to represent all 16 colors. So, all screen modes which use four bits to represent a pixel are 16-color modes. Only four-color combinations are possible with two bits, and only two combinations are possible with one bit. That's why some screen modes can display only four or two colors at a time.

The PCjr's PALETTE command can switch which colors are being displayed, but it can't add any more colors. You're still limited to the maximum number of colors for each screen mode.

Tile Painting

Once you're familiar with how pixels are represented in video memory, the technique of tile painting is easier to understand. Tile painting uses the PAINT command found in PCjr Cartridge BASIC and IBM BASICA to fill the bytes of screen memory with certain patterns of ones and zeros. This pattern is programmable, and it represents what is displayed on the TV or monitor. Instead of painting with the actual color, you paint with the bit pattern of the color. By using bit patterns, you can actually paint with more than one color around some specified border color.

```
PAINT (x,y), CHR$(bit pattern) + CHR$
  (bit pattern) + ... ,boundary color
```

The bit pattern consists of eight bits, so its decimal equivalent can range from 0 to 255 (integers only). The bit pattern must represent the colors of the pixels per byte of the screen mode you're using. This means four colors can be painted at a time in SCREEN 4 and 6, while only two colors can be painted at a time in SCREEN 3 and 5. The color patterns are put in memory next to each other as vertical lines on the screen. The following example paints SCREEN 1 with vertical bands of blue and green lines:

```
10 SCREEN 1:CLS
20 PAINT (1,1),CHR$(102),3
```

The reason why the lines are blue and green can be seen when the number 102 is expressed in binary, revealing the bit pattern:

```
102 = 01100110
```

Table 3 shows how decimal 102 is derived from this binary number.

Table 2: Color Bits

Luminance	Bits Red	Green	Blue	Color
0	0	0	0	Black
0	0	0	1	Blue
0	0	1	0	Green
0	0	1	1	Cyan
0	1	0	0	Red
0	1	0	1	Magenta
0	1	1	0	Brown
0	1	1	1	Light Gray
1	0	0	0	Dark Gray
1	0	0	1	Light Blue
1	0	1	0	Light Green
1	0	1	1	Light Cyan
1	1	0	0	Pink
1	1	0	1	Light Magenta
1	1	1	0	Yellow
1	1	1	1	White

Table 3: Converting Binary to Decimal

Value for each digit	128	64	32	16	8	4	2	1
Binary	0	1	1	0	0	1	1	0

01 = 0001 = blue
10 = 0010 = green

```
128 * 0 =   0
 64 * 1 =  64
 32 * 1 =  32
 16 * 0 =   0
  8 * 0 =   0
  4 * 1 =   4
  2 * 1 =   2
  1 * 0 =   0
        ------
        102
```

SCREEN 1 stores four pixels per byte, so the pattern works out to these colors:

```
01     10     01     10
blue   green  blue   green
```

But here's where things get tricky. If the computer is plugged into a color TV or composite color monitor (not an RGB monitor), you won't see the blue and green vertical lines that are supposed to be there. Instead, you'll see a solid bar of color that's sort of blue. And the blue is not one of the normal 16 colors available. It is a new color—one of the 256 shades that can be created this way on SCREEN 1 of the PCjr, and one of the 16 shades that can be created on SCREEN 1 of the PC.

What's happening here is something called *artifacting*. This effect takes advantage of the limited resolution of TVs and composite color monitors. When two very small pixels are placed next to each other on these screens, there isn't enough resolution to display them properly. As a result, the pixels tend to blend together and create a false color—an artifact color. The color wouldn't be visible if the screen had more resolution, which is why you usually need a TV or composite color monitor to observe this effect. RGB monitors have enough resolution to display the pixels as they're supposed to appear.

Creating New Colors

If the binary pattern 10 01 10 01 is used in the above example instead of 01 10 01 10, the shade is slightly different—blue-green-blue-green does not appear the same as green-blue-green-blue on a color TV or or composite monitor. They mix differently to create an entirely new shade of blue-green.

The PC can mix a fewer number of colors than the PCjr for two reasons. The first is that the PC has only two graphics modes, SCREEN 1 and SCREEN 2. Tile painting produces only 16 colors in SCREEN 1

and five shades of gray in SCREEN 2. Still, these are more colors than what are normally available in these modes. The second reason is that the PC does not have a PALETTE command as the PCjr does. The PC does have a second color palette in SCREEN 1, but the mixed colors look the same as the first palette on a color TV or composite monitor.

On SCREEN 6, available only in PCjr Cartridge BASIC, there are four pixels per byte. Because the pixels are very small (640 × 400 per screen), vertical bands of four different colors can be mixed to form shades of any color. In medium resolution, 320 × 200, vertical bands of two different colors form new solid colors. Tile painting doesn't work in low resolution, 160 × 200, because the pixels are too large.

For a demonstration of how closely spaced vertical bands create new colors, enter and run Program 1 (for the PCjr only). Using the LINE command instead of PAINT, line 20 fills the first 40 columns of SCREEN 6 with purple bands on every line that is a multiple of four: 0, 4, 8, 12, and so on. Line 30 fills the next 40 columns with the same color on every vertical line that is a multiple of four plus one: 1, 5, 9, 13, and so on. Then the program fills the screen with lines of the other two colors available in SCREEN 6. The result, on a TV or composite monitor, is 12 different colors instead of the four you'd expect.

Adding up all the different combinations of four colors results in 256 shades, and all 256 can be displayed on the screen at the same time. When you take into account that the PALETTE command can change any of the four basic colors into any of the other 16 colors, there are 1,092,016 possible shades in high resolution.

Program 2 (for the PCjr only) proves it can be done. This program displays 256 shades on the screen by drawing the vertical lines using only the first four colors. After painting all the shades, it randomly changes the palettes. If the colors selected by the PALETTE command were never repeated, it would take about an hour and a half to cycle through all one million colors.

Colors In Other Modes

In SCREEN 5, there are 256 possible colors, as demonstrated by Program 3 (also for the PCjr only). In SCREEN 4 and SCREEN 1, which are the same resolution, only four basic colors are available, so tile painting lets us display up to 16 hues simultaneously. With the PALETTE command on the PCjr, you can select these 16 colors from 256 possibilities. Program 4 displays 16 shades, then uses the PALETTE command to get the rest. Vertical bands with four colors don't blend in this mode, so somehow bands of two must be painted. The secret is in line 40. Since there are four pixels per byte, the last half of the byte has to be reflected in the first half. This technique insures that only two colors are in each band of four. The first half is the same as the last half, so the first band of two will be the same as the last band of two. Program 4 will also work on the PC, but without the PALETTE command (line 80) you are limited to only 16 colors.

Tile painting doesn't work correctly in SCREEN 2, high resolution with two colors, because this screen is always in black and white. However, you can get five shades of gray, as shown by Program 5 (for PC and PCjr). Solid lines form the brightest white. Lines separated by one line of black give the next-brightest white. Lines separated by two or three lines of black yield the next two shades. The middle gray can't be displayed when using the PAINT command, because it's not possible to create a bit pattern that represents two blacks and then a white. Table 4 shows which bit patterns generate the various shades of gray.

Tile painting doesn't work at all in SCREEN 3 because the pixels are too large. To see a demo of tile painting in SCREEN 1 for the PC or PCjr, run Program 6. It fills the screen with circles, displaying up to 256 colors on the PCjr and 16 colors on the PC.

Program 7, for the PC and PCjr with an RGB monitor, demonstrates the usefulness of the many new colors in a fascinating experiment. It uses SCREEN 1 and tile painting, but in a different way than seen above. Closely spaced vertical lines don't blend together on an RGB monitor, so the previous technique won't work. So instead, Program 7 uses the second part of the PAINT command. The first CHR$(*bit pattern*) controls the horizontal line above the second CHR$(*bit pattern*). Now the PAINT command can control the horizontal as well as the vertical lines, forming a checkerboard.

Although the checkerboard blends the lines together to create new colors, the colors aren't as solid as those produced by vertical lines on a TV or composite monitor. Indeed, the effect won't look very pretty on a TV or composite monitor; it's passable on an RGB.

Program 8 (for the PCjr only) employs the same technique as Program 7, but uses SCREEN 5 on the PCjr to create all 240 possible colors on the RGB monitor at once. The PALETTE command won't create any new shades here, because all 16 colors and their possible combinations are displayed.

Program 9 (for the PCjr only) is the same as the last two, but uses SCREEN 6 on the PCjr. It does a much better job of blending, although the colors still aren't perfectly solid. Ten shades are displayed at once and the PALETTE command cycles through all 240 possible shades.

Painting Your Own Programs

To use the new colors in your own programs, simply choose one of the following example programs which uses the same screen mode. Table 5 summarizes the programs and the number of color variations possible in each.

If you're programming on a PCjr, remove the lines that deal with changing the palette. You can change palettes on the PCjr in direct mode until most of the shades you want are on the screen. We suggest not changing the palettes in the 16-color modes, because the unchanged palette creates the widest variety of colors with the least amount of extra work.

In four-color modes, the screen displays 16 shades. Pick the color

you want, then refer to Table 6 for the corresponding decimal and hexadecimal translations of the bit patterns required.

If you're using a 16-color mode with a TV or composite monitor, the screen displays 256 shades and the bit patterns can be figured as follows: First choose the color. Then, starting at zero at the upper-left corner of the screen, count in hex across the screen to the column with the color you want. Remember to count in hex (0 through 9, then A through F). Then, still working in hex, count the number of rows down to the color you want. These two numbers form the bit pattern of the chosen color. Use them as shown below:

PAINT (x,y),CHR$(&H row column), boundary

Example: If row = A and column = 2, then

PAINT (x,y),CHR$(&HA2),boundary

If you're using an RGB monitor with a 16-color mode, choose which two of the 16 colors to make into the checkerboard. Then write each of their numbers in hex (0–F). Use these numbers as the bit pattern as shown below. Switching the first and second colors will create the checkerboard.

PAINT (x,y),CHR$&H 1st color 2nd color)+CHR$(&H 2nd color 1st color),boundary

Example: If 1st color = B (light cyan) and 2nd color = 2 (green), then

PAINT (x,y),CHR$(&HB2)+CHR$(&H2B)

IBM boasts of only the checkerboard technique for shading colors. I find the other method more fascinating. Now you can enhance your screens with a new palette of bright, solid colors, which formerly were thought to be impossible on an IBM.

For instructions on entering these listings, please refer to "COMPUTE!'s Guide to Typing In Programs" published bimonthly in COMPUTE!. Also, see Table 5 for a description of the programs.

Program 1: PCjr

```
IE 10 CLEAR,,,32768!:SCREEN 6:CL
      S:KEY OFF
DJ 20 FOR X=0 TO 40 STEP 4:LINE
      (X,0)-(X,200),3:NEXT
JH 30 FOR X=41 TO 80 STEP 4:LINE
      (X,0)-(X,200),3:NEXT
IA 40 FOR X=82 TO 120 STEP 4:LIN
      E (X,0)-(X,200),3:NEXT
GJ 50 FOR X=123 TO 160 STEP 4:LI
      NE (X,0)-(X,200),3::NEXT
FA 60 FOR X=160 TO 200 STEP 4:LI
      NE (X,0)-(X,200),1:NEXT
HN 70 FOR X=201 TO 240 STEP 4:LI
      NE (X,0)-(X,200),1:NEXT
PO 80 FOR X=242 TO 280 STEP 4:LI
      NE (X,0)-(X,200),1:NEXT
NM 90 FOR X=283 TO 320 STEP 4:LI
      NE (X,0)-(X,200),1:NEXT
OO 100 FOR X=320 TO 360 STEP 4:L
       INE (X,0)-(X,200),2:NEXT
MN 110 FOR X=361 TO 400 STEP 4:L
       INE (X,0)-(X,200),2:NEXT
OL 120 FOR X=402 TO 440 STEP 4:L
       INE (X,0)-(X,200),2:NEXT
GG 130 FOR X=443 TO 480 STEP 4:L
       INE (X,0)-(X,200),2:NEXT
```

Program 2: PCjr

```
IE 10 CLEAR,,,32768!:SCREEN 6:CL
      S:KEY OFF
LO 20 RANDOMIZE TIMER:Z=-1:A=INT
      (640/16)
HB 30 FOR Y=0 TO 15
OE 40 FOR X=0 TO 15:Z=Z+1
ED 50 LINE (X*A,Y*12.5)-(X*A+A,Y
      *12.5+12.5),3,B
PC 60 IF Z<>0 THEN PAINT (X*A+1,
      Y*12.5+1),CHR$(Z),3
MH 70 LINE (X*A,Y*12.5)-(X*A+A,Y
      *12.5+12.5),0,B
JI 80 NEXT X,Y
DD 90 PALETTE RND*3,RND*15:GOTO
      90
```

Program 3: PCjr

```
HK 10 CLEAR,,,32768!:SCREEN 5:CL
      S:KEY OFF
EB 20 RANDOMIZE TIMER:Z=-1:A=INT
      (320/16)
HB 30 FOR Y=0 TO 15
OE 40 FOR X=0 TO 15:Z=Z+1
ED 50 LINE (X*A,Y*12.5)-(X*A+A,Y
      *12.5+12.5),3,B
PC 60 IF Z<>0 THEN PAINT (X*A+1,
      Y*12.5+1),CHR$(Z),3
```

Table 4: Gray Scales in SCREEN 2

	Binary	Decimal	Hex	Shade
color 1 =	1 1 1 1 1 1 1 1	= 256 =	&HFF =	White
	0 1 0 1 0 1 0 1	= 85 =	&H55 =	Dull White
	(Not accessible)		=	Middle Gray
	0 0 0 1 0 0 0 1	= 17 =	&H11 =	Dark Gray
color 0 =	0 0 0 0 0 0 0 0	= 0 =	&H00 =	Black

Table 5: Program Descriptions

Program	Screen Mode	Max Colors	Colors per Screen	PC or PCjr	Display Device
Program 1	SCREEN 6	1,092,016	256	PCjr	TV or CC*
Program 2	SCREEN 6	1,092,016	256	PCjr	TV or CC
Program 3	SCREEN 5	256	256	PCjr	TV or CC
Program 4	SCREEN 1 or 4	256	16	PCjr	TV or CC
	SCREEN 1	16	16	PC	TV or CC
Program 5	SCREEN 2	5	5	PC/PCjr	TV or CC
Program 6	SCREEN 1	256	16	PCjr	TV or CC
		16	16	PC	TV or CC
Program 7	SCREEN 1 or 4	240	10	PCjr	RGB
	SCREEN 1	20	10	PC	RGB
Program 8	SCREEN 5	240	240	PCjr	RGB
Program 9	SCREEN 6	240	10	PCjr	RGB

*CC = Composite color monitor

Table 6: Translations of Bit Patterns in Four-Color Modes

	TV or Composite		RGB	
Shade Position	Decimal	Hex	Decimal	Hex
0	0	&H00	0 + 0	&H00 + &H00
1	17	&H11	17 + 70	&H11 + &H44
2	34	&H22	34 + 136	&H22 + &H88
3	51	&H33	51 + 204	&H33 + &HCC
4	70	&H44	70 + 17	&H44 + &H11
5	85	&H55	85 + 85	&H55 + &H55
6	102	&H66	102 + 153	&H66 + &H99
7	119	&H77	119 + 221	&H77 + &HDD
8	136	&H88	136 + 34	&H88 + &H22
9	153	&H99	153 + 102	&H99 + &H66
10	176	&HAA	176 + 176	&HAA + &HAA
11	187	&HBB	187 + 238	&HBB + &HEE
12	204	&HCC	204 + 51	&HCC + &H33
13	221	&HDD	221 + 119	&HDD + &H77
14	238	&HEE	238 + 187	&HEE + &HBB
15	255	&HFF	255 + 255	&HFF + &HFF

```
MH 70 LINE (X*A,Y*12.5)-(X*A+A,Y
      *12.5+12.5),0,B
JI 80 NEXT X,Y
KC 90 GOTO 90
```

Program 4: PC/PCjr

```
CA 10 SCREEN 1:CLS:KEY OFF:COLOR
      ,0
MM 20 RANDOMIZE VAL(RIGHT$(TIME$
      ,2)):Z=-1:A=INT(320/16):Y=
      0
OD 30 FOR X=0 TO 15:Z=Z+1
II 40 LINE (X*A,0)-(X*A+A,200),3
      ,B
KI 50 IF Z<>0 THEN PAINT (X*A+1,
      1),CHR$(Z+Z*16),3
CD 60 LINE (X*A,0)-(X*A+A,200),0
      ,B
QM 70 NEXT X
GL 80 PALETTE RND*3,RND*15:GOTO
      80:' Remove this line for
      PC
```

Program 5: PC/PCjr

```
CF 10 SCREEN 2,1:CLS:KEY OFF
MJ 20 FOR X=1 TO 100:LINE (X,1)-
      (X,200),1:NEXT X
DH 30 FOR X=101 TO 200 STEP 2:LI
      NE (X,1)-(X,200),1:NEXT X
HN 40 FOR X=201 TO 300 STEP 3:LI
      NE (X,1)-(X,200),1:NEXT X
KD 50 FOR X=301 TO 400 STEP 4:LI
      NE (X,1)-(X,200),1:NEXT X
IE 60 GOTO 60
```

Program 6: PC/PCjr

```
CA 10 SCREEN 1:CLS:KEY OFF:COLOR
      ,0
LO 20 RANDOMIZE VAL(RIGHT$(TIME$
      ,2))
GD 30 X=RND*320:Y=RND*200:R=RND*
      10+10:TILE=INT(RND*(15)+1)
BM 40 CIRCLE (X,Y),R,3:PAINT (X,
      Y),CHR$(TILE+TILE*16),3:CI
      RCLE (X,Y),R,0
AE 50 IF RND*10>8 THEN PALETTE R
      ND*3+1,RND*15:' Remove thi
      s line for PC
GA 60 GOTO 20
```

Program 7: PC/PCjr

```
CA 10 SCREEN 1:CLS:KEY OFF:COLOR
      ,0
CD 20 RANDOMIZE VAL(RIGHT$(TIME$
      ,2)):Z=-1:A=INT(320/16):Y=
      0:C=0
OD 30 FOR X=0 TO 15:Z=Z+1
NN 40 LINE (X*A,0)-(X*A+A,200),3
      ,B:Y=Z+Z*16:Q=Y*4:R=INT(Q/
      256):Q=Q-R*256+R
KF 50 IF Z<>0 THEN PAINT (X*A+1,
      1),CHR$(Y)+CHR$(Q),3
CD 60 LINE (X*A,0)-(X*A+A,200),0
      ,B
QM 70 NEXT X
GL 80 PALETTE RND*3,RND*15:GOTO
      80:' Remove this line for
      PC
EM 90 C=1-C:COLOR ,C:FOR Z=1 TO
      100:NEXT:GOTO 80:' Remove
      this line for PC
```

Program 8: PCjr

```
HK 10 CLEAR,,,32768!:SCREEN 5:CL
      S:KEY OFF
BB 20 RANDOMIZE TIMER:Z=-1:A=INT
      (320/16)
HB 30 FOR Y=0 TO 15
DE 40 FOR X=0 TO 15:Z=Z+1
JO 50 LINE (X*A,Y*12.5)-(X*A+A,Y
      *12.5+12.5),3,B:Q=(Z)*16:R
      =INT(Q/256):Q=Q-R*256+R
OC 60 IF Z<>0 THEN PAINT (X*A+1,
      Y*12.5+1),CHR$(Z)+CHR$(Q),
      3
MH 70 LINE (X*A,Y*12.5)-(X*A+A,Y
      *12.5+12.5),0,B
JI 80 NEXT X,Y
KC 90 GOTO 90
```

Program 9: PCjr

```
IE 10 CLEAR,,,32768!:SCREEN 6:CL
      S:KEY OFF
BB 20 RANDOMIZE TIMER:Z=-1:A=INT
      (640/16):Y=0
OD 30 FOR X=0 TO 15:Z=Z+1
NN 40 LINE (X*A,0)-(X*A+A,200),3
      ,B:Y=Z+Z*16:Q=Y*4:R=INT(Q/
      256):Q=Q-R*256+R
KF 50 IF Z<>0 THEN PAINT (X*A+1,
      1),CHR$(Y)+CHR$(Q),3
CD 60 LINE (X*A,0)-(X*A+A,200),0
      ,B
QM 70 NEXT X
BE 80 PALETTE RND*3,RND*15:GOTO
      80
```


Computed GOTOs And GOSUBs For Commodore 64

William M. Wiese

This short, relocatable utility permits computed GOTO and GOSUB statements in Commodore 64 BASIC.

You're probably familiar with GOTO and GOSUB statements, which pass control to another line in a BASIC program. In Commodore BASIC, these keywords can only be followed by a line number, as in GOTO 100. Some other versions of BASIC let you replace the line number with a variable, such as GOTO X, or even a complex expression, such as GOSUB X+100*ABS (Y). Since the line number is computed from the expression, the term *computed* GOTO or GOSUB is used to describe this feature.

Computing the destination from an expression offers two advantages. You can make your programs easier to understand by using meaningful variable names for subroutines instead of line numbers—for instance, replacing GOSUB 1000 with GOSUB DRAW. And computed GOTO and GOSUB statements offer a more flexible and efficient means of controlling program flow. For example, say that you write a program with six subroutines: The first starts at line 1000, the second is at 2000, and so on up to line 6000. The usual way to direct the computer to the correct subroutine would be with an ON-GOSUB statement:

```
ON A GOSUB 1000,2000,3000,4000,
  5000,6000
```

With computed GOSUBs, the same thing can be accomplished with the more compact statement GOSUB A. If A=1000, the computer performs the subroutine at line 1000. If A=2000, then GOSUB 2000 is performed, and so forth.

The program below adds both of these useful statements to Commodore 64 BASIC. Type in and save a copy before you run it. Enter line 130 exactly as shown (do *not* add an extra comma after the number 57812). The program automatically saves a machine language program named "CG0.ML" on disk. If you're using tape, change the ,8 to ,1 in line 130. Once the program has been created, load it with LOAD"CG0.ML",8,1 for disk or LOAD"CG0.ML",1,1 for tape.

Expressive Programming

Once the routine is loaded into memory, you can perform a computed GOSUB with the statement SYS 49152,*expression*. Replace *expression* with any variable or expression that evaluates to a valid line number (from 0–63999). Use SYS 49179,*expression* to perform a computed GOTO. For example, if the variable DRAW equals 1000, then SYS 49152,DRAW does the same thing as GOSUB 1000, and SYS 49179,DRAW does the same thing as GOTO 1000.

It's usually advantageous to substitute variables for 49152 and 49179 in such SYS statements. For instance, your program might contain the following lines:

```
10 CG=49152
90 SYS CG,DRAW
```

In Commodore BASIC, using variables in place of numbers speeds up a program. It takes the computer less time to find the value of the variable CG than it does to calculate the value of a constant such as 49152.

In some cases, you may want to use the memory locations starting at 49152 for a different machine language routine. If you use a disk drive, you can move the computed GOTO/GOSUB routine to the cassette buffer, which begins at location 828. Simply change lines 100, 140, 150, and 210 as shown here:

```
100 FOR I=828 TO 878  :rem 234
140 POKE193,60:POKE194,3
                      :rem 89
150 POKE174,110:POKE175,3
                      :rem 132
210 DATA 76,91,3,169,255,133
                      :rem 102
```

Before running the modified program, replace the name CG0.ML in line 130 with a new name (CGML/828 or whatever) that reflects the alteration. Then load the program as described above and use SYS 828,*expression* for computed GOSUB and SYS 855,*expression* for computed GOTO.

Occasionally, computed GOTOs and GOSUBs don't seem to work correctly. For example, suppose a program contains the statement SYS 49179, 5*COS(X). If X has the value 0, then this statement should do the same thing as GOTO 5 (to confirm this, type PRINT 5*COS(0) and press RETURN). Instead, the computer performs the equivalent of GOTO 4. Such effects are the result of slight rounding errors caused when the computer converts numbers from one format to another. The 64—like virtually every other computer—stores and manipulates numbers internally in a different format from the decimal numbers we ordinarily use. In this case, the computer evaluates 5*COS(0) as 4.99999999, then throws away the fraction, ending up with the integer (whole) value of 4. To prevent such rounding errors, add a small number (.00001 is a good value) to the expression. For instance, SYS 49179, 5*COS(0)+.00001 correctly performs GOTO 5.

How It Works

Computed GOTOs and GOSUBs are surprisingly easy to add to Commodore BASIC. When the computer performs an ordinary GOSUB, it "remembers" its current place in the program by storing an address and the current BASIC line number in a special memory area called the stack. An additional byte is stored on the stack to show that a GOSUB caused the stack entry. This makes it possible for the computer to find its way back to the right spot when the subroutine ends with RETURN.

From this point onward, GOSUB and GOTO share the same code and work exactly the same. The computer looks at the ASCII line number stored in the BASIC program text (if it finds anything other than ASCII numerals, it stops with an UNDEF'D STATEMENT error). Then it converts the line number to integer form and stores it in locations 20–21. Finally, the computer searches the program text for the matching line number and (if the line exists) continues forward.

To make computed GOTOs and GOSUBs possible, this utility duplicates the way a GOSUB statement stores return information on the stack. But it adds something new to the common routine that retrieves the line number from the program text. Instead of getting the line number in the old manner, we call BASIC's main evaluation routine at memory address 44446. This routine, usually labeled FRMEVL, can evaluate any BASIC expression (unlike the normal routine, which accepts only numerals). After calling a second routine at 47095 to convert the number into a two-byte address, the utility stores the line number in locations 20–21. Since this is exactly where the GOTO routine expects to find the line number, we then jump into the computer's normal routine at address 43171.

Computed GOTOs And GOSUBs

For instructions on entering this listing, please refer to "COMPUTE!'s Guide to Typing In Programs" published bimonthly in COMPUTE!.

```
100 FOR I=49152 TO 49202
                      :rem 167
110 READ A:POKE I,A    :rem 14
120 NEXT              :rem 210
130 SYS 57812"CGO.ML",8
                      :rem 226
140 POKE 193,0 :POKE 194,192
                      :rem 140
150 POKE 174,51:POKE 175,192
                      :rem 193
160 SYS 62954         :rem 160
170 DATA 169,0,133,2,169,3
                      :rem 250
180 DATA 32,251,163,165,123,72
                      :rem 193
190 DATA 165,122,72,165,58,72
                      :rem 156
200 DATA 165,57,72,169,141,72
                      :rem 152
210 DATA 76,31,192,169,255,133
                      :rem 201
220 DATA 2,32,253,174,32,158
                      :rem 90
230 DATA 173,32,247,183,32,163
                      :rem 195
240 DATA 168,165,2,240,1,96
                       :rem 47
250 DATA 76,174,167   :rem 176
```

Refurbish Your 64

Richard Roffers And Jeffrey Hock

Enhance your Commodore 64 by modifying its built-in operating system. This unusual program eliminates several annoying bugs and adds convenient new features as well.

While the Commodore 64 is a remarkable computer, its operating system, the *Kernal*, has a few notorious shortcomings. Some 64s lock up if you type a line more than 80 characters long at the bottom of the screen, then delete a character. POKEs to screen memory are invisible on some models, and none of them handle the ASC value of a null string ("") correctly. "Refurbish Your 64" corrects these problems and makes several other improvements as well. Of course, the changes are only temporary. Restarting the computer returns it to normal.

Type in and save the accompanying program, and be sure to remove any cartridges from the expansion port. When you run the program, the computer behaves as if you just turned the power on—but with a difference. The startup message reveals that the Kernal has been modified. As you may know, the 64 has programmable RAM (Random Access Memory) "underneath" the ROM (Read Only Memory) addresses where BASIC and the Kernal are stored. This program works by copying BASIC and the Kernal from ROM into the underlying RAM, modifying them, and then turning off the ROM to make the computer use the RAM-based Kernal and BASIC.

Don't worry if that seems unclear. You can use this program without knowing how all the details work. For now, notice that the number of bytes free is shown as 51,216, far more than the usual number (38,911). Since the 64 now has RAM instead of ROM at locations 40960–49151, it thinks its BASIC program space stretches all the way from location 2048 to 53264. But that's just an illusion. We can't use the RAM from 40960–49151 without destroying the modified BASIC we just put there. Before you do anything else, reset the top-of-memory pointer to its normal value by typing the following line and pressing RETURN:

```
POKE 55,0:POKE 56,160:POKE 643,0:POKE 644,160:NEW
```

This line must always be entered immediately after you run the program (or perform a cold start with SYS 64738). Once that is done, your modified 64 is ready to go. Let's look at each modification in turn and note how you can customize this program to suit your own tastes.

Screen Colors

Everyone seems to have different preferences for default screen colors. If you don't like the usual colors, they're easy to change. Lines 1460, 1500, and 1550 define the default background, border, and character colors, respectively. Change the values in those lines to whatever color numbers you like, then rerun the program. The chosen colors will reappear whenever you press RUN/STOP–RESTORE or cause a cold start with SYS 64738.

In this and other parts of the program, you'll notice that each group of DATA statements represents one change, with the first DATA statement in each group specifying the starting address and the number of bytes to be changed in ROM. The remaining DATA statements in each group contain the actual bytes that are POKEd into RAM to make the change. The REM statements in each section explain which values you may change.

Default Device

Although most 64 owners use a disk drive, the default device for LOAD, SAVE, and VERIFY is the Datassette. Lines 1600–1700 change the default device number to 8 so that a command like SAVE "FILE" (without the ,8) saves to the disk drive rather than cassette. You can still use tape by adding device number 1 to your commands (for instance, SAVE"FILE",1). However, for nonrelocating disk loads you must still add ,8,1 to the command (as in LOAD"FILE",8,1). Replace the 8 in line 1650 with a 1 if you don't have a disk drive.

Auto Load/Run

When you press SHIFT–RUN/STOP, normally the 64 loads and runs the first program on tape. Since the disk drive is now the de-

fault device, the SHIFT-RUN/STOP routine has been modified to perform the equivalent of LOAD"*",8 followed by RUN. This was necessary because disk loads (unlike tape) always require a filename. The command LOAD"*",8 normally loads the first program file on the disk. However, in some cases the wildcard symbol * is equal to the last filename used rather than the first file on disk.

Screen POKEs

Depending on the age of your 64, POKEs to screen memory (like POKE 1024,42) may produce white characters, invisible characters (the same color as the background), or characters the same color as the cursor. This program makes all screen POKEs appear in the cursor color as on the newest 64s.

Moving CLR/HOME

This is a change you may or may not find desirable, so we've made it optional. Some people often hit the CLR/HOME key by accident when trying to press the INST/DEL key. Instead of inserting a character in a line that you're editing, the screen clears and your work is lost. To eliminate this problem, remove the REMs from lines 2060–2280. This modification exchanges the positions of the CLR/HOME key and the £ key, moving CLR/HOME to a less vulnerable position. If you make this change and use this program frequently, you may want to exchange the keycaps for those keys as well. The keycaps are easily removed by prying them straight up.

INPUT Prompt

As you probably know, INPUT permits a prompt message (for example, INPUT"YOUR CHOICE";A$ prints YOUR CHOICE?). If the prompt message is longer than one screen line, INPUT either tacks the entire prompt message onto the front of your response (when accepting string input) or causes a REDO FROM START error (when accepting numeric input). Lines 2310–2340 eliminate this bug.

LIST Freezing

The 64 normally lets you slow screen scrolling (caused by PRINTing or LISTing to the screen) by pressing the CTRL key. In many cases, it's more convenient to freeze such displays rather than merely slow them down. When the modified Kernal is installed, SHIFT (or SHIFT-LOCK) will freeze screen scrolling. Pressing CTRL while a screen is frozen causes it to scroll at the normal rate as long as both SHIFT and CTRL are pressed.

Keyboard Buffer Option

Since this modification may not be useful to everyone, we've made it optional. The computer's keyboard buffer stores keystrokes temporarily. If you type faster than the computer can digest the keystrokes, the keyboard buffer remembers them until the system is ready. The buffer is normally ten characters long; when you type more than ten characters "ahead" of the system, the extra characters are lost. There are times when a longer buffer would be useful—for example, to prevent a fast typist from overflowing the buffer or to let the computer execute long direct-mode commands as if they were typed directly on the keyboard.

If you remove the REMs from lines 2530–2720, the keyboard buffer is moved from its normal location (631–640) to an 80-byte area in the cassette buffer (starting at 828). Note that many programs expect to find the keyboard buffer in its normal place and may misbehave or crash as a result of this relocation. For this reason, be careful to test the program after incorporating this change.

Power-Up Message

This message is displayed when you first run the program, and thereafter (as long as the computer remains on) when you cause a cold start with SYS 64738. After performing a cold start, you must always reset the top-of-memory pointer as explained above. Lines 3190–3220 contain the data for the new startup message. The numbers are ASCII character codes (listed in your user's guide). To replace this message with one of your own, replace these codes with ASCII codes for the characters that you want. Do not try to add any extra characters (there's no room for them in the modified Kernal). Note that *the last ASCII code must be a 0.* If you omit the final 0, the computer may crash when it tries to print the message.

Screen Lockup

Some early models of the 64 suffer from the infamous bottom-of-screen lockup bug, caused when you type in a line more than 80 characters long at the bottom of the screen, then delete a character. This bug has been eliminated.

New Erase Key

Commodore computers provide excellent full-screen editing capabilities. However, some people prefer an "erase" key that acts like a mini black hole. When you press it, the character under the cursor disappears, and everything to the right of that character moves left one space. This is the equivalent of pressing CURSOR RIGHT followed by DELETE. We chose to make the seldom-used SHIFT-£ combination into an erase key. To erase a character, just press SHIFT-£. The new erase key repeats when you hold it down, just like the cursor keys and INST/DEL (SHIFT-9 now repeats as well, an unavoidable side effect). If you need the graphics character that SHIFT-£ normally prints, use PRINT CHR$(169).

Null String Fix

The 64's normal ASC function can't handle a null string (two quotation marks with nothing between them). A statement like PRINT ASC("") causes an ILLEGAL QUANTITY error. This is one of the easiest ROM bugs to fix, requiring only a one-byte change.

For most ordinary programming, a RAM-based Kernal and BASIC work just fine. However, since other programs may use the same RAM area (to store a high-resolution screen, make other modifications to BASIC, or whatever), you must be alert for conflicts. If another program POKEs into the RAM where the modified BASIC and Kernal are stored, the computer may crash. As mentioned earlier, turning the computer off and on restores the original, ROM-based versions of BASIC and the Kernal. The only way to make these changes permanent is to store the

modified BASIC and Kernal in EPROM (Eraseable Programmable ROM) chips and substitute them for the existing ROM chips—a job requiring specialized equipment and expertise.

Refurbish Your 64

For instructions on entering this listing, please refer to "COMPUTE!'s Guide to Typing In Programs" published bimonthly in COMPUTE!.

```
1000 REM COMMODORE 64 KERNAL M
     ODIFIER.              :rem 238
1030 REM{4 SPACES}THIS SECTION
      OF CODE IS A         :rem 130
1040 REM{4 SPACES}SMALL MACHIN
     E LANGUAGE            :rem 91
1050 REM{4 SPACES}PROGRAM WHIC
     H DOWNLOADS           :rem 224
1060 REM{4 SPACES}THE KERNAL I
     NTO THE               :rem 100
1070 REM{4 SPACES}UNDERLYING R
     AM AND THEN           :rem 143
1080 REM{4 SPACES}BANKS OUT TH
     E KERNAL ROM.         :rem 206
1150 PRINT"{CLR}{9 DOWN}
     {15 SPACES}{11 @£}"
                           :rem 208
1200 PRINT"{15 SPACES}{RVS}PLE
     ASE WAIT"             :rem 149
1210 PRINT"{2 DOWN}{4 SPACES}D
     ON'T FORGET TO RESET THE
     {SPACE}TOP OF"        :rem 105
1220 PRINT"{DOWN}{7 SPACES}MEM
     ORY POINTERS (SEE TEXT)."
                           :rem 149
1230 FOR I=49152 TO 49212:READ
      B:POKE I,B:NEXT I:rem 87
1240 DATA 169,0,133,251,169,16
     0,133,252,76,24,192,234,1
     69,0,133,251          :rem 226
1250 DATA 169,224,133,252,76,2
     4,192,234,162,32,160,0,17
     7,251,145,251         :rem 20
1260 DATA 200,208,249,230,252,
     202,208,242,96,234,120,16
     5,1,37,253            :rem 118
1270 DATA 133,1,88,96,234,120,
     165,1,5,253,133,1,88,96
                           :rem 54
1300 REM{4 SPACES}THIS DOWNLOA
     DS BASIC.             :rem 27
1320 POKE 253,1:SYS 49204:SYS
     {SPACE}49152          :rem 102
1350 REM{4 SPACES}THIS DOWNLOA
     DS THE KERNAL.        :rem 92
1370 POKE 253,2:SYS 49204:SYS
     {SPACE}49164          :rem 111
1400 REM{4 SPACES}THE DATA BEL
     OW MODIFIES           :rem 109
1410 REM{4 SPACES}THE DEFAULT
     {SPACE}BACKGROUND,
                           :rem 156
1420 REM{4 SPACES}BORDER, AND
     {SPACE}CHARACTER      :rem 245
1430 REM{4 SPACES}COLORS (IN T
     HAT ORDER).           :rem 65
1450 DATA 60633,2          :rem 68
1460 DATA 6:REM{3 SPACES}THE B
     ACKGROUND COLOR       :rem 121
1470 REM{10 SPACES}CODE OF YOU
     R CHOICE              :rem 90
1480 REM{10 SPACES}(VALUES 0 -
      15).                 :rem 195
1500 DATA 6:REM{3 SPACES}THE B
     ORDER COLOR           :rem 82
1510 REM{10 SPACES}CODE OF YOU
     R CHOICE              :rem 85
1520 REM{10 SPACES}(VALUES 0 -
      15).                 :rem 190
1540 DATA 58677,1          :rem 82
1550 DATA 14:REM{2 SPACES}THE
     {SPACE}CHARACTER COLOR
                           :rem 85
1560 REM{10 SPACES}CODE OF YOU
     R CHOICE              :rem 90
1570 REM{10 SPACES}(VALUES 0 -
      15).                 :rem 195
1600 REM{4 SPACES}THE DEFAULT
     {SPACE}DEVICE NUMBER
                           :rem 10
1610 REM{4 SPACES}FOR 'LOAD',
     {SPACE}'SAVE', AND
                           :rem 169
1620 REM{4 SPACES}'VERIFY'.
                           :rem 254
1640 DATA 57818,1          :rem 79
1650 DATA 8:REM{3 SPACES}THE D
     ISK DRIVE WILL        :rem 250
1660 REM{10 SPACES}BE THE NEW
     {SPACE}DEFAULT        :rem 8
1670 REM{10 SPACES}DEVICE.
     {2 SPACES}IF YOU DO
                           :rem 175
1680 REM{10 SPACES}NOT HAVE A
     {SPACE}DISK DRIVE,
                           :rem 218
1690 REM{10 SPACES}REPLACE THE
      8 IN LINE            :rem 136
1700 REM{10 SPACES}1650 WITH A
      1.                   :rem 84
1730 REM{4 SPACES}THE SCREEN P
     OKE FIX.              :rem 148
1740 REM{4 SPACES}THE SCREEN C
     OLOR MEMORY           :rem 169
1750 REM{4 SPACES}WILL NOW BE
     {SPACE}FILLED WITH:rem 80
1760 REM{4 SPACES}THE CURRENT
     {SPACE}CHARACTER      :rem 67
1770 REM{4 SPACES}COLOR.
                           :rem 96
1790 DATA 58586,3          :rem 90
1800 DATA 173,134,2        :rem 160
1830 REM{4 SPACES}THE LOAD/RUN
      MODIFICATION.        :rem 121
1840 REM{4 SPACES}IF YOU DO NO
     T HAVE A DISK         :rem 81
1850 REM{4 SPACES}DRIVE, DELET
     E LINES               :rem 134
1860 REM{4 SPACES}1810 THROUGH
      1890.                :rem 158
1880 DATA 60647,9          :rem 87
1890 DATA 76,207,34,58,42,13,8
     2,213,13              :rem 84
1920 REM{4 SPACES}THIS SECTION
      OF CODE WILL         :rem 229
1930 REM{4 SPACES}EXCHANGE THE
     £ KEY WITH            :rem 86
1940 REM{4 SPACES}THE CLR/HOME
      KEY.{2 SPACES}IF THE
                           :rem 83
1950 REM{4 SPACES}REMS IN LINE
     S 2060 THROUGH        :rem 229
1960 REM{4 SPACES}2280 ARE REM
     OVED THEN:            :rem 211
1970 REM                   :rem 181
1980 REM{4 SPACES}1){2 SPACES}
     THE TWO KEY CAPS MUST
                           :rem 68
1990 REM{8 SPACES}BE PHYSICALL
     Y EXCHANGED           :rem 199
2000 REM{8 SPACES}AND      :rem 121
2010 REM{4 SPACES}2){2 SPACES}
     THIS PROGRAM SHOULD
                           :rem 33
2020 REM{8 SPACES}ALWAYS BE RU
     N                     :rem 245
2030 REM{8 SPACES}IMMEDIATELY
     {SPACE}AFTER THE      :rem 48
2040 REM{8 SPACES}COMPUTER IS
     {SPACE}TURNED ON.     :rem 82
2060 REM DATA 60337,1      :rem 38
2070 REM DATA 19           :rem 49
2090 REM DATA 60340,1      :rem 35
2100 REM DATA 92           :rem 44
2120 REM DATA 60402,1      :rem 28
2130 REM DATA 147          :rem 96
2150 REM DATA 60405,1      :rem 34
2160 REM DATA 169          :rem 103
2180 REM DATA 60467,1      :rem 45
2190 REM DATA 147          :rem 102
2210 REM DATA 60470,1      :rem 33
2220 REM DATA 168          :rem 99
2240 REM DATA 60584,1      :rem 42
2250 REM DATA 255          :rem 99
2270 REM DATA 60587,1      :rem 48
2280 REM DATA 28           :rem 52
2310 REM{4 SPACES}INPUT PROMPT
      MESSAGE FIX.         :rem 54
2330 DATA 58918,2          :rem 79
2340 DATA 234,234          :rem 65
2370 REM{4 SPACES}THE FOLLOWIN
     G CODE (WHICH         :rem 248
2380 REM{4 SPACES}WILL NOT BE
     {SPACE}EXECUTED       :rem 184
2390 REM{4 SPACES}BECAUSE OF T
     HE REM                :rem 4
2400 REM{4 SPACES}STATEMENTS)
     {SPACE}WILL RELOCATE
                           :rem 98
2410 REM{4 SPACES}THE KEYBOARD
      BUFFER TO            :rem 58
2420 REM{4 SPACES}THE CASSETTE
      BUFFER AND           :rem 118
2430 REM{4 SPACES}WILL EXPAND
     {SPACE}THE KEYBOARD
                           :rem 215
2440 REM{4 SPACES}BUFFER TO 80
      CHARACTERS.          :rem 129
2450 REM{4 SPACES}IF YOU WISH
     {SPACE}TO HAVE THE:rem 30
2460 REM{4 SPACES}KEYBOARD BUF
     FER MODIFIED,         :rem 40
2470 REM{4 SPACES}REMOVE THE R
     EMS FROM LINES        :rem 70
2480 REM{4 SPACES}2530 THROUGH
      2720, AND            :rem 103
2490 REM{4 SPACES}ALSO, THE RE
     MS PRECEEDING         :rem 252
2500 REM{4 SPACES}THE DATA STA
     TEMENTS IN            :rem 69
2510 REM{4 SPACES}LINES 2930 A
     ND 2940.              :rem 197
2530 REM DATA 58669,1      :rem 55
2540 REM DATA 80:REM{2 SPACES}
     NEW BUFFER LENGTH:rem 181
2560 REM DATA 58871,2      :rem 54
2570 REM DATA 59,3         :rem 153
2590 REM DATA 58569,2      :rem 61
2600 REM DATA 60,3         :rem 139
2620 REM DATA 58575,2      :rem 52
2630 REM DATA 60,3         :rem 142
2650 REM DATA 58805,2      :rem 51
2660 REM DATA 60,3         :rem 145
2680 REM DATA 58813,2      :rem 53
2690 REM DATA 60,3         :rem 148
2710 REM DATA 58810,2      :rem 44
2720 REM DATA 61,3         :rem 143
2750 REM{4 SPACES}SCREEN LOCK-
     UP FIX.               :rem 130
2770 DATA 58769,9          :rem 98
2780 DATA 228,201,240,3,76,237
     ,230,96,234           :rem 233
2800 DATA 58748,21         :rem 131
2810 DATA 32,240,233,169,39,23
     2,180,217             :rem 130
2820 DATA 48,6,24,105,40,232,1
     6,246,133             :rem 121
2830 DATA 213,76,36,234
                           :rem 112
```

```
2860 REM{4 SPACES}THE ERASE KE
     Y.                    :rem 28
2880 DATA 60220,3          :rem 69
2890 DATA 32,194,228       :rem 228
2910 DATA 58562,16         :rem 131
2920 DATA 201,169,208,8,169,29
     ,157,119,2,232,169,20,157
     ,119,2,96             :rem 103
2930 REM DATA 58562,16:rem 105
2940 REM DATA 201,169,208,8,16
     9,29,157,60,3,232,169,20,
     157,60,3,96           :rem 229
2970 REM{4 SPACES}THE FOLLOWIN
     G PATCH CAUSES        :rem 124
2980 REM{4 SPACES}THE SHIFTED
     {SPACE}£ KEY TO :rem 135
2990 REM{4 SPACES}AUTO REPEAT.
                           :rem 224
3010 DATA 60157,6          :rem 67
3020 DATA 32,183,228,234,234,2
     34                    :rem 35
3040 DATA 58551,11         :rem 119
3050 DATA 201,41,240,6,201,20,
     240,2,201,32,96       :rem 130
3080 REM{4 SPACES}THE CHANGE O
     F THE COLD            :rem 206
3090 REM{4 SPACES}START AND WA
     RM START              :rem 214
3100 REM{4 SPACES}ROUTINE.
                           :rem 252
3120 DATA 64982,1          :rem 74
3130 DATA 229              :rem 126
3160 REM{4 SPACES}THE NEW STAR
     TUP MESSAGE.          :rem 223
3180 DATA 58483,56         :rem 137
3190 DATA 147,13,32,32,32,32,4
     2,32,82,69,86,73,83,69,68
     ,32                   :rem 56
3200 DATA 82,65,77,45,82,69,83
     ,73,68,69,78,84,32,75,69,
     82                    :rem 47
3210 DATA 78,65,76,32,42,13,13
     ,32,67,79,77,77,79,68,79,
     82                    :rem 36
3220 DATA 69,32,54,52,32,32,0,
     0                     :rem 227
3250 REM{4 SPACES}THIS CHANGE
     {SPACE}ALLOWS THE :rem 63
3260 REM{4 SPACES}SHIFT KEY TO
      INHIBIT              :rem 192
3270 REM{4 SPACES}SCROLLING.
                           :rem 139
3290 DATA 59723,11         :rem 128
3300 DATA 173,141,2,201,1,240,
     240,160               :rem 254
3310 DATA 0,132,198        :rem 161
3330 DATA 59710,4          :rem 73
3340 DATA 141,2,201,1 :rem 244
3354 REM{4 SPACES}FIX ASCII NU
     LL STRING             :rem 21
3356 DATA 46991,1          :rem 85
3357 DATA 5                :rem 33
3370 REM{4 SPACES}THE END OF D
     ATA MARKER            :rem 218
3390 DATA 99999             :rem 6
3420 REM{4 SPACES}THE FOLLOWIN
     G CODE READS          :rem 201
3430 REM{4 SPACES}THE DATA STA
     TEMENTS AND           :rem 132
3440 REM{4 SPACES}POKES THE DA
     TA INTO THE           :rem 71
3450 REM{4 SPACES}KERNAL RAM.
                           :rem 123
3470 READ A0:IF A0=99999 THEN
     {SPACE}POKE 253,253:SYS 4
     9194:SYS 64738:REM COLDST
     ART.                  :rem 50
3480 READ N:FOR I=A0 TO A0+N-1
     :READ A$:POKE I,A$:NEXT I
     :GOTO 3470            :rem 63
```


Apple ProDOS Disk Menu

K. Michael Parker

Here's a fast method of loading and running programs at the touch of a key. The program requires an Apple IIc or IIe with the ProDOS operating system.

How many times have you found yourself wishing for an easier way to load and run programs? The process of calling up a disk catalog, looking for the desired pathname, then typing (or mistyping) it can be a frustrating experience—especially if the pathname is something cryptic like FNINPT.BAO.2. Perhaps a better alternative is to select the program from a menu taken from the disk directory.

That's exactly what you can do with "ProDOS Disk Menu." To use it, create a startup disk by saving both ProDOS and BASIC.SYSTEM on a disk. Then save Disk Menu with the filename STARTUP.

When you boot this disk, a menu containing the first 16 programs in the directory appears on the 40-column screen. If more than 16 programs are on the disk, press P to view the next page. Pressing P on the last page returns you to the first page. (Disk Menu accepts both uppercase and lowercase commands.)

If you don't find the program you want, press C. A screen prompt asks you to switch disks, then Disk Menu reruns itself.

To select a program, press the up/down arrow keys to position the cursor over the desired filename, then press RETURN. A screen prompt offers three choices: (R)UN, (L)OAD, OR (U)NDO. If you made a mistake and selected the wrong program, press U to return to the menu.

Loading Multiple Programs

There are three ways to exit Disk Menu: run any program, load a BASIC program, or press Q to quit. Notice that loading a machine language program *does not* exit Disk Menu. Therefore, if you have a BASIC program that utilizes several ML subroutines, you could load the ML routines into memory one after the other, then exit DISK MENU by running the BASIC program.

The programming techniques used in Disk Menu are quite simple. The program retrieves the volume name from the disk and opens the volume directory. It reads the directory into an array, skipping all nonprogram files (except the type mentioned below). Then, depending on the current page, the program reads the filenames into the page array for display and selection.

A few parts of the program may need some explanation. For example, line 325 skips past the first few records on the volume directory, which do not contain information essential to the menu.

Disk Menu does not list any file types other than BASIC and binary files (.BAS and .BIN). Although data is sometimes stored in binary files, it is usually considered

good practice to store data and programs on separate disks.

When writing Disk Menu, I was tempted to make it include files contained in subdirectories, but refrained because in my experience such files are usually chained to other programs, and would therefore only clutter up the menu. However, if you use subdirectories differently, the necessary alterations should be fairly simple.

Apple ProDOS Disk Menu

For instructions on entering this listing, please refer to "COMPUTE!'s Guide to Typing In Programs" published bimonthly in COMPUTE!.

```
51 10 D$ = CHR$ (4)
53 12 L = 1
E2 20 A$(20) = "":A$(21) = "PRES
      S <RETURN> TO ACCEPT CHOIC
      E":A$(22) = "'Q'-QUIT":A$(
      23) = "'P'-PAGINATE  'C'-G
      ET NEXT CATALOG"
44 100 HOME
54 110 INVERSE : PRINT "  D I S
      K   M E N U ": NORMAL
34 200 REM GET VOLUME LABEL
81 210 PRINT D$;"PREFIX/"
3A 220 PRINT D$;"PREFIX"
15 230 INPUT VL$
F7 240 PRINT D$
24 250 VTAB 2: PRINT VL$: REM DI
      SPLAY VOLUME NAME
ED 300 REM GET DIRECTORY
C9 310 PRINT D$;"OPEN";VL$;",TDI
      R"
D7 320 PRINT D$;"READ";VL$
D1 325 INPUT Z$: INPUT Z$: INPUT
       Z$
CC 330 INPUT L$(L)
DF 335 CH$ = MID$ (L$(L),18,3)
4E 340 IF L$(L) = "" THEN 360
FA 345 IF CH$ < > "BAS" AND CH$
      < > "BIN" THEN 330
6A 350 L = L + 1: GOTO 330
BF 360 PRINT D$;"CLOSE"
72 370 MAX = L - 1
75 400 PAGE = 0
F4 410 GOSUB 2000: REM LOAD PAGE
       INTO ARRAY
52 420 GOSUB 3000: REM PRINT ARR
      AY
FF 430 GOSUB 4000: REM ACCEPT IN
      PUT
F1 440 GOTO 5000: REM RUN/LOAD
B6 2000 REM  INITIALIZE ARRAY
68 2010 FOR I = 4 TO 19:A$(I) =
      "": HTAB 5: VTAB I: PRIN
      T SPC( 16): NEXT
27 2020 REM  LOAD PAGE INTO ARR
      AY
64 2030 N = PAGE * 16:PAGE = PAG
      E + 1
20 2040 IF (MAX - N) > = 16 THEN
       LIM = 16: IF (MAX - N)
      = 16 THEN PAGE = 0
4D 2045 IF (MAX - N) < 16 THEN L
      IM = MAX - N:PAGE = 0
A5 2047 A = 4
A1 2050 FOR I = (N + 1) TO (N +
      LIM)
DB 2060 A$(A) = L$(I)
4B 2070 A = A + 1: NEXT
F2 2080 RETURN
4D 3000 REM PRINT ARRAY
B0 3020 FOR I = 4 TO 19
76 3030 HTAB 5: VTAB I
09 3040 PRINT MID$ (A$(I),2,16)
B7 3050 NEXT
C5 3060 FOR I = 21 TO 23: VTAB I
      : PRINT A$(I): NEXT
43 3062 CR = 4: INVERSE : VTAB C
      R: HTAB 5: PRINT MID$ (A
      $(CR),2,16): NORMAL
A2 3065 VTAB CR: HTAB 4
EF 3070 RETURN
A3 4000 REM ACCEPT INPUT
53 4010 GET C$
3C 4020 IF C$ < > CHR$ (10) AND
      C$ < > CHR$ (11) AND C$
      < > CHR$ (13) AND C$ < >
       "Q" AND C$ < > "C" AND
      C$ < > "q" AND C$ < > "c
      " AND C$ < > "P" AND C$
      < > "p" THEN 4010
88 4030 IF C$ < > CHR$ (10) AND
      C$ < > CHR$ (11) THEN 45
      00
02 4040 REM MOVE CHOICE
5D 4050 VTAB CR: HTAB 5: NORMAL
      : PRINT MID$ (A$(CR),2,1
      6)
EF 4060 ON ASC (C$) - 9 GOSUB 41
      00,4200
BF 4070 INVERSE : VTAB CR: HTAB
      5: PRINT MID$ (A$(CR),2,
      16): NORMAL
A7 4075 VTAB CR: HTAB 4
74 4080 GOTO 4010
D6 4100 REM DOWN
DA 4110 IF CR = 19 THEN CR = 3
D3 4120 IF CR < > 19 THEN CR = C
      R + 1
E2 4130 RETURN
77 4200 REM UP
6D 4210 IF CR = 4 THEN CR = 20
CD 4220 IF CR < > 4 THEN CR = CR
       - 1
E4 4230 RETURN
82 4500 IF C$ = "Q" OR C$ = "q"
      THEN HOME : END
2E 4510 IF C$ = "P" OR C$ = "p"
      THEN POP : GOTO 410
8D 4520 IF C$ = CHR$ (13) THEN R
      ETURN : REM  LOAD/RUN SU
      BROTINE
02 4530 REM C$ MUST BE C OR c SO
       CONTINUE
CC 4540 HOME : INPUT "INSERT NEW
       DISK THEN PRESS <RETURN
      >";ANS$
7F 4550 POP : GOTO 10
21 5000 REM RUN/LOAD
A7 5002 FILE$ = MID$ (A$(CR),2,1
      6):TYPE$ = MID$ (A$(CR),
      18,3)
EA 5010 HTAB 5: VTAB 3
B2 5020 PRINT "(R)UN,(L)OAD, OR
      (U)NDO": HTAB 28: VTAB 3
5D 5030 GET E$
55 5040 IF E$ < > "R" AND E$ < >
       "L" AND E$ < > "1" AND
      E$ < > "r" AND E$ < > "U
      " AND E$ < > "u" THEN 50
      30
DA 5045 HTAB 5: VTAB 3: PRINT "
                              "
50 5047 IF E$ = "U" OR E$ = "u"
      THEN PAGE = 0: GOTO 410
7F 5050 IF E$ = "L" OR E$ = "1"
      THEN 5100
9E 5060 HOME : PRINT D$;"-";FILE
      $
17 5070 NEW
E0 5100 IF TYPE$ = "BAS" THEN HO
      ME : PRINT D$;"LOAD";FIL
      E$
44 5110 IF TYPE$ = "BIN" THEN PR
      INT D$;"BLOAD";FILE$
D9 5120 GOTO 420                  ©
```

Atari Fine Scrolling

Karl E. Wiegers

Unlock the secrets of fine-scrolling screen displays with this step-by-step tutorial, complete with example programs. Recommended for intermediate BASIC and machine language programmers. The techniques work on all Atari 400/800, XL, and XE computers.

An especially powerful graphics feature of Atari computers is their ability to scroll all or part of a screen display. Both text and graphics screens can be scrolled horizontally, vertically, or diagonally by various increments. Scrolling is seen in such diverse applications as racing games, in which the moving roadway lends apparent motion to the stationary cars, and in strategic war games, in which players can manipulate the screen as a "window" over a much larger map.

There are two general types of scrolling: *coarse scrolling* and *fine scrolling*. Coarse scrolling moves the screen in increments of eight pixels (the size of one character); fine scrolling moves the screen in increments of one pixel and is much more realistic. Earlier articles have addressed the rudiments of coarse scrolling (see "Fun with Scrolling" by David Plotkin in *COMPUTE!'s Second Book of Atari*). As we'll see in a moment, fine scrolling is actually a combination of coarse and fine scrolling. Since these techniques require machine language to work properly, we'll present a vertical blank interrupt routine you can add to your BASIC programs to obtain the smooth, continuous scrolling effect seen in many Atari games.

Coarse Scrolling

Let's review some details about how Atari computers display information on the video screen. A special microprocessor chip called ANTIC governs the display process. ANTIC gets its instructions from a short program in memory called the *display list*. The display list tells ANTIC what kind of graphics mode to use for each display line, how many lines to show, where in RAM to find the data to be displayed, and other information. The starting RAM location of the display list can be found using this formula:

```
DL=PEEK(560)+256*PEEK(561)
```

Ordinarily, the block of RAM containing screen display data is defined when a GRAPHICS statement is executed in BASIC. The first byte of screen memory—which is displayed in the upper-left corner of the screen—is identified by the fifth and sixth bytes in the display list in the usual low-byte, high-byte format:

```
MEMST=PEEK(DL+4)+256*PEEK
(DL+5)
```

An elegant feature of the Atari operating system is that the section of RAM to be displayed on the screen can be altered simply by changing the values in DL+4 and DL+5, the pointers to screen RAM. For example, consider a graphics mode 2 display, with 20 characters or bytes of RAM per line. If we add 20 to the screen RAM pointers, the twenty-first byte of the original block of screen RAM would appear in the upper-left corner of the screen. This causes every part of the display to jump up by one mode line: a vertical coarse scroll. Conversely, subtracting 20 from the screen RAM pointers scrolls the display downward by one mode line in graphics mode 2.

Program 1 is a simple vertical coarse-scrolling routine written in BASIC for graphics mode 0. (Type this listing with the line numbers shown; we'll be adding to it later.) In line 150, the starting byte of screen RAM is incremented by 40 to generate each step of the scroll. Then the starting location is factored into its corresponding high- and low-byte values (lines 160—170), which are inserted into the display list (lines 180—190). Coarse scrolling can only change the position of display information in relatively large jumps, equal to the height of a character in whatever graphics mode is being used. It yields a jerky, rough appearance when scrolling a screenful of data.

These same principles apply to the concept of horizontal scrolling. However, horizontal scrolling is a bit more complex because it involves fooling the computer into thinking that each mode line is wider than the usual screen display. To make things easier, we'll stick to vertical scrolling.

Mixed Scrolling

The secret to fine scrolling, as mentioned above, is to mix coarse and fine scrolling. Atari computers were designed to allow vertical fine scrolling in increments of one video scan line (there are 192 in a normal full-screen display). In graphics mode 0, which has eight scan lines per mode line, the fine scrolling capability thus permits seven increments of vertical movement between mode lines. To scroll a display by more than just one mode line, your program must execute seven fine scrolls, then one coarse scroll. The final coarse scroll, in effect, appears onscreen as the eighth fine scroll.

All of this requires two basic steps. First, the program must in-

form ANTIC which mode lines in the display are enabled for fine scrolling. Second, the program must store into an appropriate hardware register an integer representing the number of scan lines to scroll.

The first step, enabling the desired mode lines for scrolling, takes us back to the display list. We've already seen how to find the display list in RAM and how to alter the bytes pointing to the start of screen memory. Most of the other instructions in the display list identify the kind of graphics mode line to display. (For a more detailed discussion of display lists, see Craig Chamberlain's article "How to Design Custom Graphics Modes" in *COMPUTE!'s First Book of Atari Graphics*.) To enable a mode line for vertical fine scrolling, you must set bit 5 of its display list instruction. This is equivalent to adding 32 to the contents of the byte, and it must be done for each mode line you want to scroll. If you like, you can define several blocks of scrollable lines. Mode lines which don't have bit 5 set can be coarse-scrolled, but not fine-scrolled.

The second step, telling ANTIC how many scan lines to scroll, requires a simple POKE into a register called VSCROL at location 54277 (hex $D405). VSCROL affects all lines which have been enabled for vertical fine scrolling. For instance, the statement POKE 54277,4 shifts the display in each enabled mode line upward by four scan lines. Notice that you can POKE only positive integers into VSCROL (or into any other byte, for that matter). In effect, this means you can scroll the display upward but not downward. To simulate downward scrolling, you must start with the display scrolled fully upward (store a 7 in VSCROL for graphics modes 0 or 1, or 15 for mode 2, and so on), then POKE a smaller number into VSCROL to move the contents of each mode line downward by one or more scan lines.

Here, then, is the procedure for a complete mixed-scrolling routine:

1. Fine scroll a number of scan lines which is one less than the pixel height of the graphics mode.

2. Reset VSCROL to the starting value (0 if scrolling up, the maximum value if scrolling down).

3. Coarse scroll by one mode line.

4. Repeat the procedure.

A Fine Example

To add vertical fine scrolling to our previous example of coarse scrolling, merge the lines in Program 2 with those in Program 1. After running the program, you must press SYSTEM RESET to restore the original display list.

Notice that the bottom of the screen moves up slightly after running this program. Because of the way that ANTIC works, a block of mode lines enabled for fine scrolling results in a loss of one mode line of display area. This shortens the screen display.

To make this program scroll downward rather than upward, change the following lines:

```
80 POSITION 2,5
110 FOR S=7 TO 0 STEP -1
150 MEMST=MEMST-40
200 POKE 54277,7
```

As the display scrolls downward, you'll see the display list itself come into view, since it's normally found immediately before the start of screen memory. The display list appears mostly as a string of uppercase Bs. That's because the internal character code for an uppercase B is 34, the same as the display list instruction for a graphics 0 line enabled for vertical fine scrolling. You'll also see the fifth (and occasionally sixth) character in the display list change with each coarse scroll. These are the pointers to screen memory we discussed earlier.

To see a scrolling demo in graphics mode 1 instead of graphics 0, press SYSTEM RESET, type NEW, reload Program 1, and once again add the lines in Program 2. Then substitute these lines:

```
10 GRAPHICS 1+16
60 POKE DL+6+X,38
90 PRINT #6;"MODE ONE DEMO"
150 MEMST=MEMST+20
220 GOTO 220
```

Now to convert it for graphics mode 2, press RESET and make these changes:

```
10 GRAPHICS 2+16
50 FOR X=0 TO 9
60 POKE DL+6+X,39
80 POSITION 2,11
90 PRINT #6;"MODE TWO DEMO"
100 FOR D=1 TO 9
110 FOR S=0 TO 15
```

Scrolling Behind The Scenes

As you run these demos, you'll notice that they still suffer from some unsightly flickers and jumps, even though they're clearly a big improvement over simple coarse scrolling. The problem is that BASIC can't POKE the display list and scroll registers fast enough to synchronize with the TV or monitor's electron beam which is displaying the video image. To achieve smooth, flicker-free scrolling, your program must change all the registers during the split-second when the beam is displaying nothing on the screen. This *vertical blank interval* happens 60 times a second when the beam returns from the bottom to the top of the screen to sweep another video "frame." Since BASIC isn't nearly fast enough for this job, a machine language routine is required.

Program 3 is a BASIC loader which incorporates such a routine. (Program 4 is the source code for machine language programmers; don't type it in unless you have an assembler.) Be sure to save a copy of Program 3 before running it for the first time. When you type RUN, it stores the machine language routine in memory page 6 (starting at location 1536, hex $600), then sets up a *vertical blank interrupt (VBI)*, a mechanism which calls the routine during each vertical blank interval. The program also modifies the display list as described above and initializes a few memory locations (203–206) for the VBI routine.

After the screen clears, you'll see it fill with a mass of apparently random letters, numbers, and graphics symbols. That's because the program has scrolled the display past the end of usable RAM and into the BASIC cartridge itself. The scrolling continues until you press SYSTEM RESET.

An apparent limitation of a VBI scrolling routine is that it can't scroll the display faster than 60 times a second, because it's called only 60 times a second. If you want to scroll faster, you can scroll more than one scan line at a time—although it won't appear as smooth.

There's also a way to scroll more slowly. This routine uses a counter at location 203 to control the scroll rate. It checks to see how many vertical blank intervals have passed since the last fine scroll, then compares the result against a preset limit to see if it's time for another fine scroll. To make the routine wait for more than one vertical blank interval between fine scrolls, change the 1 in line 60 of Program 3 to a higher number.

The comments in Program 4 tell machine language programmers how to modify this VBI routine to work in other graphics modes.

Program 5 is a BASIC loader for a downward-scrolling VBI routine. It's not a stand-alone program—it must be combined with certain lines in Program 3 as described in the REM statements. (Program 6 is the source code for Program 5 so machine language programmers can study the technique. Again, don't type in Program 6 unless you have an assembler.)

So far we've seen simple demos of the Atari's scrolling capabilities. Now let's use them for something fun.

Empire State Building

Scrolling is most often used in programs that have a larger display than can be shown on a single screen. By scrolling across parts of the display data, you can use the screen as a window onto other sections of RAM. Consider, for example, that a graphics mode 2 screen has 12 lines of 20 bytes each, or only 240 bytes of information. That leaves enough memory in the computer to create a display containing thousands of bytes of data—maybe a dozen or more screens. This is the technique seen in such classic Atari games as *Caverns of Mars* and *Eastern Front 1941*.

Let's try a simple example. Program 7 shows the Empire State Building as it might appear to a parachutist leaping out of a helicopter over Manhattan. The building is composed of redefined graphics mode 2 characters. It took 1200 bytes of RAM to store the building and background, which are conveniently located in a character string called ESB$. The beauty of this approach to allocating memory is that your program can easily find the first byte of ESB$ with BASIC's string ADR function. Then it can use this address as the upper-left corner of the screen by modifying the screen display pointers in the mode 2 display list. The 1200 bytes of ESB$ amount to five screens of graphics mode 2 data.

The VBI routine used in the Empire State Building example is slightly different from that in Program 4. First, it had to be modified for graphics mode 2. Second, it has a counter which is incremented after each coarse scroll. When the counter reaches a preset value (corresponding to street level in this case), the scrolling stops. You can change the 48 in line 150 of Program 7 to stop the scrolling at some other point. Press SYSTEM RESET each time before running this program to keep the redefined characters from getting messed up.

Just The Beginning

These examples illustrate the power of the graphics scrolling ability of Atari computers, but they're just a start. We don't have room in this article to cover extensions of these techniques, such as horizontal fine scrolling; diagonal scrolling; joystick-controlled scrolling; and altered perspective scrolling, in which cleverly designed character sets are combined with scrolling routines to create effective three-dimensional effects. With the ideas presented here, you can probe some of these techniques on your own.

For instructions on entering these listings, please refer to "COMPUTE!'s Guide to Typing In Programs" published bimonthly in COMPUTE!.

Program 1: Coarse Scrolling Demo

```
DC 10  GRAPHICS 0
PH 20  DL=PEEK(560)+256*PEEK(
       561):REM Start of disp
       lay list
PN 30  MEMST=PEEK(DL+4)+256*P
       EEK(DL+5):REM Start of
        screen memory
KB 80  POSITION 2,15
HD 90  PRINT "THIS IS A DEMO
       OF COARSE SCROLLING"
NP 100 FOR D=1 TO 15:REM Loo
       p to scroll through 1
       5 lines
LB 130 FOR DLAY=1 TO 100:NEX
       T DLAY:REM Delay loop
GO 150 MEMST=MEMST+40
HI 160 HI=INT(MEMST/256):REM
        New high byte for sc
       reen memory in displa
       y list
MJ 170 LO=MEMST-HI*256:REM N
       ew low byte
HO 180 POKE DL+4,LO
HG 190 POKE DL+5,HI
BG 210 NEXT D
```

Program 2: Fine Scrolling Demo

(Merge these lines with Program 1.)

```
II 40  REM Enable vertical fi
       ne scroll on all mode
       lines except first and
        last
BH 50  FOR X=0 TO 21
JM 60  POKE DL+6+X,34
PO 70  NEXT X
NI 90  PRINT "THIS IS A DEMO
       OF FINE SCROLLING"
JL 110 FOR S=0 TO 7:REM Fine
        scroll 7 times
EK 120 POKE 54277,S
LL 130 FOR DLAY=1 TO 20:NEXT
        DLAY
CH 140 NEXT S
DH 200 POKE 54277,0:REM Rese
       t vertical fine scrol
       l register
```

Program 3: VBI Routine BASIC Loader

```
OC 10  GRAPHICS 0
IK 20  REM Read data for VBI
       routine
OM 30  REM Increase the 1 in
       line 60 to slow the sc
       roll rate
KM 40  FOR X=1 TO 63:READ A:P
       OKE 1535+X,A:NEXT X
LC 50  DATA 104,160,10,162,6,
       169,7,76,92,228
GI 60  DATA 230,203,169,1,197
       ,203,208,42,230,204
DL 70  DATA 165,204,141,5,212
       ,169,0,133,203,169
HE 80  DATA 7,197,204,176,25,
       169,0,141,5,212,133
PM 90  DATA 204,169,40,160,4,
       24,113,205,145,205,144
ND 100 DATA 7,200,169,0,113,
       205,145,205,76,98,228
NB 110 REM Modify display li
       st for vertical fine
       scrolling on all line
       s
HF 120 DL=PEEK(560)+256*PEEK
       (561)
EO 130 POKE DL+3,98
II 140 FOR X=0 TO 21:POKE DL
       +6+X,34:NEXT X
ML 150 POSITION 2,20
OP 160 PRINT "FINE SCROLL WI
       TH VERTICAL"
JO 170 PRINT "BLANK INTERRUP
       T"
HG 180 REM Initialize variab
       les for VBI routine
BF 190 POKE 203,0:POKE 204,0
JF 200 POKE 205,PEEK(560)
JI 210 POKE 206,PEEK(561)
NJ 220 REM Turn on VBI routi
       ne; press SYSTEM RESE
       T to make it stop
CN 230 A=USR(1536)
```

Program 4: VBI Routine Source Code

(An assembler is required to enter this listing.)

```
10 ;VBI routine
20 ;for combined fine and
    coarse scrolling
30 ;in graphics mode 0.
40 ;
50 ;Change the 1 in line
    180 to scroll slower.
60 ;Change the 7 in line
    260 to the number of s
    can lines
70 ;per mode line minus 1
    for other graphics mo
    des.
80 ;Change the 40 in line
    320 to the number of
90 ;bytes per mode line f
    or other graphics mode
    s
0100 ;
0110   *=$0600          ;Load
    into page 6.
0120   PLA              ;Remov
    e argument count.
0130   LDY #10          ;These
     4 statements set up
0140   LDX #6           ;a def
    erred vertical blank
0150   LDA #7           ;inter
    rupt - use LDA #6 for
0160   JMP $E45C        ;an im
    mediate VBI.
0170   INC $CB          ;$CB i
    s counter for number
    of
0180   LDA #1           ;VB cy
    cles before next scro
    ll.
0190   CMP $CB          ;If no
    t up to desired inter
    val
0200   BNE EXIT         ;then
    exit VBI
0210   INC $CC          ;$CC i
    s counter for number
0220   LDA $CC          ;of fi
    ne scrolls.
0230   STA $D405        ;Store
     in vertical fine scr
    oll register.
0240   LDA #0           ;reset
     VB counter
0250   STA $CB
0260   LDA #7           ;Have
    we done 7 fine
0270   CMP $CC          ;scrol
    ls yet?
0280   BCS EXIT         ;No, e
    xit VBI.
0290   LDA #0           ;Yes,
0300   STA $D405        ;reset
     vertical scroll regi
    ster,
0310   STA $CC          ;reset
     scroll counter,
0320   LDA #40          ;and c
    oarse scroll by
0330   LDY #4           ;addin
    g 40 to low byte
0340   CLC             ;of sc
    reen memory pointer.
0350   ADC ($CD),Y
0360   STA ($CD),Y
0370   BCC EXIT         ;If ca
    rry not set then exit
     VBI,
0380   INY             ;else
    increment high byte
0390   LDA #0           ;of sc
    reen memory pointer.
0400   ADC ($CD),Y
0410   STA ($CD),Y
0420 EXIT JMP $E462
```

Program 5: Downward VBI BASIC Loader

(Combine with Program 3.)

```
DI 10 REM This loads a VBI r
      outine
GC 20 REM for downward fine
      scrolling
AI 30 REM into page 6 (1536,
      $600).
CH 35 REM Call it with A=USR
      (1536).
NF 40 REM Initialize locatio
      ns 203-205 as in Progr
      am 3 before running,
MF 50 REM but execute POKE 2
      04,7 instead.
PB 60 REM Don't forget to mo
      dify display list for
DP 70 REM fine scrolling bef
      ore using this routine
LC 80 FOR X=1 TO 65:READ A:P
      OKE 1535+X,A:NEXT X
LF 90 DATA 104,160,10,162,6,
      169,7,76,92,228,230,20
      3,169,1,197,203,208
EE 100 DATA 44,165,204,201,2
      55,240,8,141,5,212,19
      8,204,76,58,6,169,7,1
      33
AC 110 DATA 204,141,5,212,16
      0,4,56,177,205,233,40
      ,145,205,176,8,200,17
      7
CP 120 DATA 205,56,233,1,145
      ,205,169,0,133,203,76
      ,98,228
```

Program 6: Downward VBI Source Code

(An assembler is required to enter this listing.)

```
10 ;VBI routine
20 ;for downward fine scr
    olling.
30 ;
40   *=$0600           ;Loa
    d into page six
50   PLA               ;Rem
    ove argument count
60   LDY #10           ;Set
    up deferred VBI
70   LDX #6
80   LDA #7
90   JMP $E45C
0100   INC $CB         ;Inc
    rement VB counter
0110   LDA #1          ;Cha
    nge the 1 to scroll s
    lower
0120   CMP $CB         ;Tim
    e to fine scroll yet?
0130   BNE EXIT        ;No,
     exit VBI
0140   LDA $CC         ;Yes
    , see if time for coa
    rse scroll
0150   CMP #255
0160   BEQ COARSE      ;Yes
    , go to coarse scroll
     routine
0170   STA $D405       ;No,
     then fine scroll
0180   DEC $CC         ;Dec
    rement fine scroll co
    unter
0190   JMP INCREMENT   ;Exi
    t via VB counter
0200 COARSE LDA #7     ;Coa
    rse scroll routine
0210   STA $CC         ;Res
    et fine scroll counte
    r
0220   STA $D405       ;Res
    et fine scroll regist
    er
0230   LDY #4          ;Get
     low byte of screen m
    emory pointer
0240   SEC
0250   LDA ($CD),Y
0260   SBC #40         ;Sub
    tract 40 from it
0270   STA ($CD),Y
0280   BCS INCREMENT   ;Exi
    t via VB counter unle
    ss
0290   INY             ;you
     need to get the high
     byte
0300   LDA ($CD),Y
0310   SEC
0320   SBC #1          ;and
     subtract 1 from it
0330   STA ($CD),Y
0340 INCREMENT LDA #0 ;Re
    set VB counter
0350   STA $CB
0360 EXIT JMP $E462   ;Exi
    t from VBI
```

Program 7: Empire State Building Demo

```
JJ 30 REM Reserve memory for
      redefined characters
      in GR. 2
LD 40 POKE 106,PEEK(106)-2:C
      HBASE=256*PEEK(106)
JG 50 GRAPHICS 18:POKE 559,0
      :REM Turn off video di
      splay
MA 60 SETCOLOR 0,12,8:REM Gr
      een for top of buildin
      g
AN 70 SETCOLOR 1,8,6:REM Blu
      e for sky
LD 80 SETCOLOR 3,1,8:REM Yel
      low for windows
FF 90 REM ESB$ is screen dis
      play RAM, A$ is charac
      ter for wall with wind
      ows
MG 100 DIM A$(16),ESB$(1200)
PJ 110 A$="■":A$(16)=A$:A$(2
      )=A$
NA 120 ESB$="A":ESB$(1200)=E
      SB$:ESB$(2)=ESB$
FP 130 REM VBI routine
NM 140 FOR X=1 TO 71:READ A:
      POKE 1535+X,A:NEXT X
MI 150 DATA 104,160,10,162,6
      ,169,7,76,92,228,165,
      207,201,48,240,52,230
      ,203,169,2
NG 160 DATA 197,203,208,44,2
      30,204,165,204,141,5,
      212,169,0,133,203,169
CM 170 DATA 15,197,204,176,2
      7,169,0,141,5,212,133
      ,204,230,207,169,20,1
      60,4,24
KN 180 DATA 113,205,145,205,
      144,7,200,169,0,113,2
      05,145,205,76,98,228
```

```
EH 190 REM Turn on new chara
       cter set
AB 200 POKE 756,CHBASE/256
LJ 210 REM Read redefined ch
       aracters
OC 220 FOR X=1 TO 5:READ OFF
       SET:REM A,B,C,D
HN 230 FOR J=0 TO 7:READ A:P
       OKE CHBASE+8*OFFSET+J
       ,A:NEXT J:NEXT X
CB 240 DATA 1,255,255,255,25
       5,255,255,255,255
AL 250 DATA 2,252,252,252,25
       2,252,252,252,252
IN 260 DATA 3,63,63,63,63,63
       ,63,63,63
LP 270 DATA 4,0,68,68,0,0,68
       ,68,0
FM 280 DATA 35,255,255,255,2
       55,255,255,255,255
OG 290 REM Create Empire Sta
       te Building with rede
       fined characters
HO 300 FOR X=230 TO 270 STEP
        20
PC 310 ESB$(X,X+1)="BC":NEXT
        X
ID 320 FOR X=290 TO 330 STEP
        20
LF 330 ESB$(X,X+1)="##":NEXT
        X
JD 340 FOR X=349 TO 429 STEP
        20
EJ 350 ESB$(X,X+3)=A$(13):NE
       XT X
JL 360 FOR X=448 TO 588 STEP
        20
EL 370 ESB$(X,X+5)=A$(11):NE
       XT X
JL 380 FOR X=606 TO 986 STEP
        20
CG 390 ESB$(X,X+9)=A$(7):NEX
       T X
OC 400 FOR X=1003 TO 1183 ST
       EP 20
ME 410 ESB$(X,X+15)=A$:NEXT
        X
OP 420 REM These are suppose
       d to be doors
AK 430 ESB$(1166,1166)="E":E
       SB$(1170,1170)="E":ES
       B$(1174,1174)="E"
NK 440 REM Set up GR. 2+16 d
       isplay list for verti
       cal fine scrolling
HL 450 DL=PEEK(560)+256*PEEK
       (561)
HH 460 POKE DL+3,103
PG 470 FOR X=DL+6 TO DL+15:P
       OKE X,39:NEXT X
LN 480 REM Start display RAM
        at beginning of ESB$
JJ 490 HI=INT(ADR(ESB$)/256)
BI 500 LO=ADR(ESB$)-256*HI
JB 510 POKE DL+4,LO:POKE DL+
       5,HI
GO 520 REM Initialize variab
       les for VBI
HB 530 POKE 203,0:POKE 204,0
       :POKE 207,0
JH 540 POKE 205,PEEK(560)
JP 550 POKE 206,PEEK(561)
KD 560 REM Turn video displa
       y back on
AB 570 POKE 559,34
JD 580 REM Start VBI going;
       it will stop automati
       cally
DG 590 A=USR(1536)
GI 600 REM This is needed to
        keep GR. 2 display o
       n the screen
GH 610 GOTO 610                ©
```

Commodore Program Chaining

Orlando Lee Stevenson

Take advantage of Commodore's automatic chaining feature to link two or more BASIC programs together. The method illustrated here applies to all Commodore computers.

Program chaining is a method of linking separate programs together, making them run, in effect, as one large program. Why would you need to chain? Some BASIC programs simply grow too large to fit into memory: Chaining lets you break them into two or more program modules that work together as one. This method also lets you interconnect an entire group of programs, moving from one to another whenever you like.

LOAD In Program Mode

Let's say you have two programs you want to chain together. The solution can be as simple as placing LOAD "PROGRAM NAME",8 (disk) or LOAD "PROGRAM NAME" (tape) in place of an END statement. In Commodore BASIC, a LOAD command executed as part of a program automatically loads and runs the specified program. If the programs are completely unrelated, nothing more needs to be done.

However, if the programs are related, you'll probably want to pass variable values from one to the other as well—a procedure that requires some care. On all Commodore computers except the 128, variables and arrays are stored in memory immediately following the end of BASIC program text. Since different programs are of different lengths, the actual location of variables depends on the length of the program. The computer uses two-byte address pointers to keep track of where everything is stored, and updates the pointers as needed while the program runs. When you perform LOAD in program mode, the computer does not reset the pointers for variables, arrays, and strings. Thus, after it loads a second program, the computer still knows how to find and use all of the first program's variables.

The success of this procedure depends on the relative length of the chained programs. If the first program is *longer* than the second, all is well: When the second program loads in, its shorter program text doesn't extend as far as the area where variables are stored. (Remember, the first program's variables are still located in the same

place). However, if the first program is *shorter* than the second, you have trouble. When the second program loads, its longer text overwrites the variables. Though the pointers still point to the right area, the variable data which used to be there has been replaced with program lines. Once that happens, the variables are lost.

This is not a problem with BASIC 7.0 on the Commodore 128, because it keeps variables in a separate 64K bank of memory. Thus, 128 programs can be chained freely without worrying about overwriting variables, and all the following discussion about preserving variables does not apply. However, you should still read the section entitled "Chain with Care." And don't forget that variables *will* be overwritten if you're running the 128 in 64 mode, just as they would be on a 64.

Changing The Signposts

The easiest solution is to make sure the first program in a chain is longer than all the rest. However, in many cases the first program in a chain is quite short. It may be a menu program—one that simply lets you choose among several programs to load and run.

Fortunately, there's an answer. By resetting the first program's pointers, you can make it store variables in an area that won't be disrupted by following programs in the chain. Here are the steps to follow (you can use any BASIC programs to practice this technique):

1. First, find the length of every program in the chain. Load the program and type the appropriate line below in direct mode (without a line number), then press RETURN.
   For the VIC, 64, 128 in 64 mode, Plus/4, and 16:
   PRINT PEEK(45)+PEEK(46)*256
   For PET/CBM (Upgrade and 4.0 BASIC):
   PRINT PEEK(42)+PEEK(43)*256
   This number is the location where the program's text ends and its variable storage begins. Write down the end-of-text number and note which program it belongs to, then repeat for every program in the chain.
2. Scan the list of numbers to find the longest program: It's the one with the highest end-of-text number. Now reload that program and find the contents of the two addresses you PEEKed above. For the VIC, 64, Plus/4, or 16, type **PRINT PEEK(45),PEEK(46)** in direct mode; substitute the proper addresses if you are using a PET/CBM. Two numbers are printed. These are the actual pointer values for the longest program. Write them down, labeling the first number LO and the second HI. You now know the lowest safe storage address for variables in this chain.
3. Reload the first program in the chain. Do not run it or enter any direct mode statements that would create variables. Enter the following lines, *replacing LO and HI with the numbers you recorded in step 2.* For instance, if LO is 20 and HI is 9, you would type the first line as POKE 45,20:POKE 46, 9+1. Don't forget to press RETURN after each line.
   For the VIC, 64, Plus/4, and 16:
   POKE 45,LO:POKE 46,HI+1
   POKE 47,LO:POKE 48,HI+1
   POKE 49,LO:POKE 50,HI+1
   For PET/CBM (Upgrade and 4.0 BASIC):
   POKE 42,LO:POKE 43,HI+1
   POKE 44,LO:POKE 45,HI+1
   POKE 46,LO:POKE 47,HI+1
4. Finally, resave this program. *Do not delete the original version* (see explanation below). Step 3 sets the first program's end-of-text and variable pointers to an address 256 bytes above the end of the longest program (the extra bytes provide a margin for error). Though it artificially increases the length of the first program, this method lets you run the entire package without losing variables.

Chain With Care

This method of program chaining has limitations. User-defined functions—created with DEF FN()—cannot be passed at all, since their definitions are stored in program text, not as variables. Such functions must be redefined in every program that uses them.

Strings may cause problems as well. In the VIC, 64, and PET/CBM versions of BASIC *dynamic* strings (which result from a string operation such as A$="HELLO"+B$) are stored outside the program text, they can be passed like other variables. The same is not true of *static* strings. Like a function definition, a static string exists only in a program line (10 A$="HELLO"). If you need to pass a static string, simply add a null string to it (for instance, replace 10 A$="HELLO" with 10 A$= "HELLO"+""). The string operation (+) turns it into a dynamic string, storing it outside the program. This is not a problem in the 128, Plus/4, and 16 versions of BASIC, where all strings are effectively dynamic.

Be careful when editing chained programs. If you lengthen a program, it may become the longest one in the chain and overwrite variables when it loads. Do *not* edit and resave a program after breaking out with RUN/STOP (since the pointers are set at artificially high locations, the program's length is abnormal). Instead, reload the program to set the pointers correctly, then make the changes and save it again. Whenever you edit any of the programs in the chain, you should also repeat steps 1–4, using the *original version* of the first program. It's critical that you know the true length of this program, not the inflated length it was given in steps 3 and 4.

There are other ways to pass variables while chaining, but they're inevitably cumbersome. One approach is to store variable data in a separate memory area while one program loads another. For instance, say that A=10. Just before the first program loads the second, it POKEs the value of A into a safe memory location (say, 49152 for the Commodore 64). The first thing the second program does is retrieve A's value with a statement like A=PEEK(49152). Since a single memory location can hold only a number from 0–255, it requires multiple POKEs and PEEKs to pass larger numeric values. Passing arrays, strings, or more complex numbers (negative values, for instance) takes even more work and ingenuity. ©

Commodore Dynamic Keyboard

Part 3

Jim Butterfield, Associate Editor

Parts 1 and 2 of this series showed how the dynamic keyboard technique—which allows the computer to seemingly type on its own keyboard—lets you do things that would otherwise be difficult or impossible from within a program. Now we'll look at the trickiest application of this technique—writing a program that changes itself as it runs.

Let's quickly review how the dynamic keyboard technique works. First, the program prints the desired command at a specific screen location. Then a RETURN character is placed in the keyboard buffer. Finally, the program stops with the cursor flashing over the screen command. The RETURN in the keyboard buffer causes the operating system to read the command on the screen and carry it out, just as if you pressed the RETURN key. Using the same principle, we can put several commands on the screen and make the program execute them all.

The following table shows the location of the keyboard buffer counter and the start of the keyboard buffer on most Commodore computers:

	Counter	Buffer
VIC-20, Commodore 64	198	631
Commodore 16, Plus/4	239	1319
PET/CBM (Upgrade and 4.0 BASIC)	158	623
Original ROM PETs	525	527
B128 (Model 700)	209	929

For a single-line command, POKE a value of 1 into the counter and a value of 13 (RETURN) into the buffer. To execute more than one screen line of commands, use a higher count and more RETURN characters. On the B128 computer, it's wise to execute a BANK 15 command before the POKEs.

Self-Editing Programs

The usual way to change a program is to type in a new line and press RETURN. The line is either added to the program or it replaces an existing line with the same line number. A program can do this, too, using the dynamic keyboard technique. But there's a hitch. Whenever you enter a program line, the computer performs a CLR command, which closes all open files and clears the contents of all variables and arrays. This can be annoying, since it's hard for a program to continue running after its variables are gone. But with some careful programming, you can still make things work.

The solution is to identify your key variables, make the program change itself, then reinstate the variables with the dynamic keyboard technique. In effect, the variables are temporarily stored on the screen and put back in the program by the equivalent of a direct command. Tricky? Crude? Whatever your opinion of this method, the point is that it works. There are other ways to do the job, but you usually want to get it done in the most direct way possible.

You might be wondering why you'd ever need to design a program that modifies itself, anyway. Here's an example. Suppose you have something in a special part of memory—a machine language program, a screen picture, or a data table. Whatever it is, you want to take the information and build it into a series of DATA statements so it can be reconstituted by a BASIC program when needed. Perhaps you'd like to publish a small machine language program in a newsletter or magazine, and want readers to be able to type it in as DATA statements rather than the more complex hexadecimal code. How to do it?

First, let's write some data into memory so that you'll have something to convert to DATA statements. Here's a quick program to put a series of prime numbers into memory locations 828 to 881:

```
100 POKE 828,2            :rem 192
110 POKE 829,3            :rem 195
120 N=3                    :rem 81
130 FOR A=830 TO 881      :rem 216
140 N=N+2                 :rem 203
150 FOR M=3 TO SQR(N)+.1 STEP2
                          :rem 106
160 T=N/M                 :rem 242
170 IF T=INT(T) GOTO 140
                           :rem 22
180 NEXT M                 :rem 37
190 PRINT N;              :rem 176
200 POKE A,N              :rem 124
210 NEXT A                 :rem 19
```

That's not the most efficient prime number generator, but it does put the numbers into memory. The last number should be 251. Now, suppose you want these values in DATA statements so that a different program will be able to POKE them back at the start of the run.

Frenzied Activity

Type NEW to make space for the new program. The following program is written for the Commodore 64. If you're using another computer, refer to the table above to find the right POKE values for lines 75 and 80.

```
10 L=100:A=830          :rem 206
15 PRINT CHR$(147)      :rem 225
20 PRINT                :rem 239
25 PRINT                :rem 244
30 PRINT L;"DATA";      :rem 16
35 D$=STR$(PEEK(A))     :rem 50
40 IF N>0 THEN D$=","+MID$(D$,
   2)                   :rem 51
45 PRINT D$;            :rem 153
50 A=A+1:N=N+1:IF N<10 AND A<8
   82 GOTO 35           :rem 54
55 PRINT                :rem 247
60 PRINT "L=";L+10;":A=";A;
                        :rem 193
65 PRINT ":GOTO 15"     :rem 21
70 PRINT CHR$(19)       :rem 176
75 POKE 198,1:IF A<882 THEN PO
   KE 198,2             :rem 224
80 POKE 631,13:POKE 632,13
                        :rem 85
85 END                  :rem 68
```

Be sure to type the semicolon at the ends of lines 30, 45, and 60, and note that some of the strings printed in lines 60 and 65 start with a colon character. When you RUN the program, you'll see a frenzy of activity on the screen for a few moments. Then the action stops with the cursor over a line which says L=160:A=882:GOTO 15. Don't execute this line. Instead, move the cursor down, type LIST, and press RETURN. You'll find that the program contains six new lines of DATA statements.

Start the new DATA lines at line number 100 (variable L). Since the data maker program ends at a lower line number, there's no danger of replacing existing lines with new ones. Never increase the line number directly. Instead, print a higher value onto the screen. When the dynamic keyboard reinstates the variable, it's ten higher than before. There's no need to set variable N back to zero. The CLR caused by changing the program effectively clears all variables to zero.

After the DATA lines have been created—you've generated only a few—you might want to get rid of the program that made them. You could do this manually by clearing the screen and giving the direct command:

```
FOR J=10 TO 85 STEP 5:PRINT J:
  NEXT J
```

This prints the line numbers on a blank screen. You could then move the cursor back and strike RETURN 16 times, eliminating the lines. It would take a little ingenuity, but you could even cause the program to wipe itself out using the dynamic keyboard. (Hint: Crunch the program into less than ten lines—then stuff the keyboard buffer with the same number of RETURN characters.)

Convert ASCII To BASIC

Occasionally, you might have a sequential file on disk or tape that contains a program. You'd like to run this program, but LOAD can't handle a sequential file. Dynamic keyboard lets you bring the file into memory and convert it to a regular program file. How could this need arise? There are several possibilities. First, the program might have been transmitted over a communications link. It's possible to download a program in a form that's ready to run, but it's common to transmit a program in ASCII form—as ASCII characters only, rather than the usual mixture of ASCII and BASIC tokens. Now, however, you must change the listing back into a working program.

Here's another way it might happen. You want to transfer a program between two slightly incompatible computers. Perhaps you have a PET program that you'd like to use on a Plus/4, or vice versa. You may be surprised to find that the SCRATCH command used in a PET program doesn't load correctly on a Plus/4. Knowing the technical reason for this (the computers use different tokens for some commands) doesn't help solve the problem. Since ASCII listings contain no tokens, you'll find that they transport more easily than ordinary programs.

Another possibility is that you want to merge two programs into one. The dynamic keyboard offers one way to do this. Note that we're breaking down the distinction between data and programs—as personified by sequential and program files, respectively. This opens the door to such things as program-analyzing programs and program-writing programs.

Change A Program To A File

Let's start by writing a simple program. Anything will do, but let's use the following:

```
100 FOR J=1 TO 10
110 PRINT J,SQR(J)
120 NEXT J
```

Remember to type NEW before entering this program. Now store it as a sequential file:

```
OPEN 1,8,6,"0:PROGFILE,S,W":CMD
  1:LIST
```

After you press RETURN, the disk drive operates briefly, then the cursor returns. Now type:

```
PRINT#1:CLOSE 1
```

It is *very important* to close the file in exactly this manner.

The program is now stored as an ASCII listing in a sequential file named PROGFILE. If displayed on the screen, it would look almost like the original program. But it consists of nothing but ASCII characters. Thus, the first line contains the characters 1-0-0 (the line number), then a space, then the letters F-O-R, and so on. This is quite different from the tokenized form in which programs are usually stored.

This file has a few oddities caused by the way in which it was stored. Unlike most data files, it begins with two RETURN characters. And it ends with the word READY (after all, when you LIST a program to the screen, it always ends with the word READY). None of this is critical, but if you plan to do advanced work with such a file, keep these things in mind.

Keep The File Open

However, there's another problem to consider. Every time you enter a new line with the dynamic keyboard, CLR closes the file you're using. You've learned how to recreate variables, but how do you reinstate the file? You can use the following fact: The file isn't really closed, it's just "disconnected."

CLR signals that no files are open simply by putting the value zero in the computer's number-of-open-files counter. If this is the only file you're handling, you can reconnect it by POKEing a 1 into the counter. The counter is found at one of the following locations, depending on your machine:

	File Counter
VIC-20, Commodore 64	152
Commodore 16, Plus/4	151
PET/CBM (Upgrade and 4.0 BASIC)	174
Original ROM PETs	610
B128 (Model 700)	864

Let's write a program to bring in this file. You know in advance that the line numbers start at 100, so you can safely put the loader program at lower numbers. If that isn't true for other situations, you might try renumbering the program with very high line numbers (above 60000, for instance).

Again, the following example runs on the VIC-20 and Commodore 64. Change the POKEs in lines 50, 65, and 70 to suit your machine. Be sure to include the semicolon at the end of line 40.

```
10 OPEN1,8,8,"PROGFILE":rem 84
15 PRINT CHR$(147)     :rem 225
20 PRINT               :rem 239
25 PRINT               :rem 244
30 GET#1,X$:X=ASC(X$) :rem 178
35 IFX>47 AND X<58 THEN F=1
                       :rem 175
40 IF F=1 THEN PRINT X$;
                       :rem 26
45 IF X<>13 GOTO 30    :rem 202
50 IF ST=0 THEN PRINT "POKE 15
   2,1:GOTO 15"         :rem 5
55 IF ST<>0 THEN PRINT "CLOSE
   {SPACE}1"           :rem 241
60 PRINT CHR$(19)      :rem 175
65 POKE 198,2          :rem 154
70 POKE 631,13:POKE 632,13
                       :rem 84
75 END                 :rem 67
```

When you run this program, it loads and merges the sequential ASCII listing. Typing 14 lines in order to add three more may seem inefficient. But the principle works on programs of any size.

It's been a long voyage. If you've stayed with it, you can probably see how the dynamic keyboard technique expands what you can do with the computer. Though it requires care, it also creates new possibilities. "Dynamic keyboard" is not just a buzzword, although you may add it proudly to your vocabulary. It's a new resource. ©

Advanced Commodore 128 Video

Jim Butterfield, Associate Editor

Here's how to relocate screen memory and set up a custom character set on the Commodore 128—two valuable techniques worth mastering on any computer. When you run the example program, be ready for a surprise. For intermediate and advanced BASIC programmers.

You can do a lot of graphics on the Commodore 128 with an elementary knowledge of the new BASIC: circles, squares, lines, and points appear by means of simple BASIC commands. But advanced programmers may still need to get into the mechanics of video. Here's a simple exercise for 128-mode 40-column screens that will give a little insight into the "works."

The question often arises: How can I implement a new character set? Some people want to design their own personalized codes or graphics symbols for the screen; others are interested in foreign languages. In 40 columns, the 8564 video chip is practically identical to the 6567 of the Commodore 64. With a few new rules, we can put the chip's features to work in the same way.

Because the Commodore 128 makes it easy, I'll be including some hexadecimal addresses in the following listing. If you'd rather use decimal numbers, the computer will do quick conversions for you, and you can make the substitutions in the program.

Changing Addresses

Let's build the program step by step and note points of interest.

```
100 POKE 58,DEC("C0")
110 CLR
```

I'm planning to put the screen and its new character set into memory bank 1, at addresses $C000 to $CBFF—character set at $C000, screen at $C800. (By the way, if you'd rather use the decimal value 192 instead of DEC("C0"), be my

guest. I prefer C0 because it's easier to visualize it as part of the full address $C000. Be sure to type a zero and not the letter O or you'll get an error.) Bank 1 is where BASIC puts its variables; we wouldn't want these to get mixed up with our screen. So we cut down the top-of-variable-memory pointer to $C000. There's really no danger of a memory conflict with this small program, but we might as well do it right.

The CLR command makes sure the other variable pointers don't get mixed up by this change.

```
120 TRAP 500
```

This command may be unfamiliar to many Commodore programmers. It sets up an *error trap* so that if anything goes wrong in the following code, the computer hops to line 500, which will restore the screen. This saves us from the horrible prospect of watching the program stop with a syntax error while the screen is still scrambled and unreadable. The TRAP command gives us another bonus: If the computer freezes—or is just too slow—we can press STOP, and the program zips to line 500 and wraps things up.

```
130 BANK 15
```

We're about to fiddle with the insides of computer chips (registers), so this command calls for memory bank 15 to make the chips accessible. This assures that the next few POKEs will be directed to the right place.

```
140 POKE DEC("DD00"),148
```

Except for the decimal number conversion ($DD00=56576), this POKE is identical to the way it's done on the Commodore 64. Briefly, it means: Display video out of the memory slice in the range $C000 to $FFFF. We haven't specified the bank yet, but we'll get around to it in a moment.

```
150 POKE DEC("0A2C"),32
```

We're still in bank 15, but this address isn't a chip. The address $0A2C (decimal 2604) is below $4000 (16384). When we're using bank 15, all such low addresses go to RAM, bank 0. This POKE sets the position of the character set and the screen within the video slice we've selected. The calculation goes like this: We want the screen to be at $C800, which is 2K above the start of the video slice at $C000, so multiply the 2 by 16 and add a similar value for the character set. In this case, the character set is right at the start of the slice; so we add 0 to get a value of 32.

On the Commodore 64, we'd do exactly the same calculation, but we'd put the result in address $D018 (53272). In fact, that's the same address at which our value will end up in the Commodore 128, but we must let the computer's interrupt routine deliver it there for us. So instead of POKEing the value directly into $D018, we store it at $0A2C (2604). As part of the computer's interrupt procedure, it will copy the contents of this location into $D018.

```
160 POKE DEC("D506"),68
```

This tells the computer to take video from bank 1. If we wanted video from bank 0, we'd POKE a value of 4—or just leave this line out, since that's the value that will be there in any case.

```
170 POKE 217,4
```

This POKE tells the computer to take its video from RAM, not ROM. We don't need to give this one for the addresses we have chosen, since there is no conflict. This very low address has a special banking rule: All addresses below hex $400 (1024) go to RAM bank 0, regardless of the bank which has been specified.

Relocating The Screen

Now our video is set up and ready to go. We'd better put something on the screen so we can see it working. It seems sensible to copy our old screen to the new place; then we'll copy the character set. We'll make a slight change so you can see how to create a new set of characters.

First, our screen must move from bank 0, address $400, to bank 1, address $C800. We must move the whole thousand characters.

```
200 FOR J=0 TO 999
210 BANK 0:X=PEEK(1024+J)
220 BANK 1:POKE DEC("C800")+J,X
230 NEXT J
```

This moves screen memory, but since the character set is not in place, the result would look rather muddy. We can read the character set by selecting bank 14; it is found in this bank at addresses $D000 to $D7FF. There are 256 characters times 8 bytes per character, which means 2,048 bytes to move. Just as we moved the screen in the lines above, we must move the character bytes one at a time, flipping between banks 14 and 1.

We'll also change the characters slightly as we move them. This allows us to see that indeed we've taken control of the character set.

```
300 FOR J=DEC("C000") TO
  DEC("C7FF") STEP 8
310 FOR K=0 TO 7
320 BANK 14
330 X=PEEK(J+4096+7-K)
340 BANK 1
350 POKE J+K,X
360 NEXT K
370 NEXT J
```

This puts the character set in place. When you run the program (after typing in the additional lines below), you should see your original computer screen—slightly changed. We could insert a delay loop to prolong the effect, but the screen takes long enough to change that you'll have plenty of time to see what happens.

Cleaning Up

We're finished—almost. We must be neat and put everything back the way it was. This also gives you a chance to see the original values that were in the various registers and addresses.

```
500 BANK 15
510 POKE DEC("DD00"),151
520 POKE DEC("0A2C"),20
530 POKE DEC("D506"),4
540 POKE 217,0
```

These lines restore the original screen. A little study should enable you to guess at what each POKE does—or undoes.

Finally, we need two last lines to complete the job. But there's an important note: *Do not* enter these lines until you've tested the program and found it good. If your program has a problem, you'll want to be able to look at the variables (by using commands such as PRINT J) to find out what went wrong. These final lines make it impossible for you to do so.

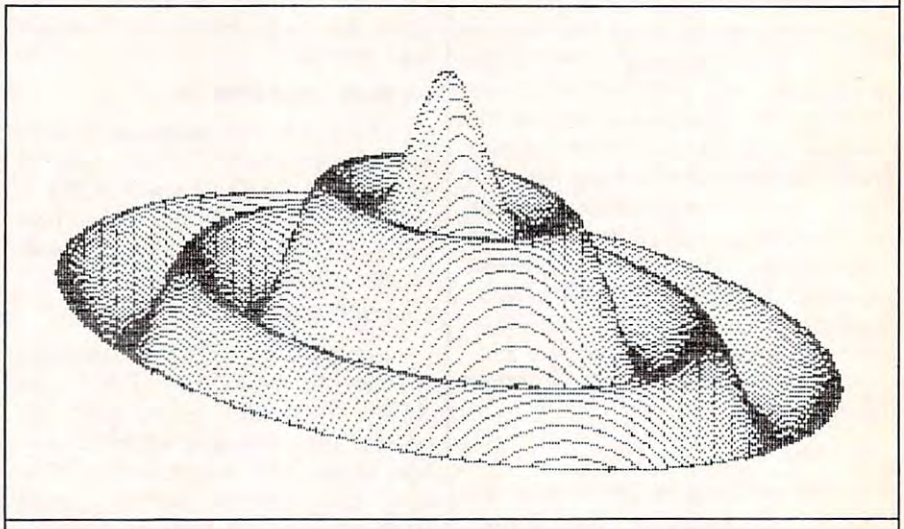
```
550 POKE 58,DEC("FF")
560 CLR
```

We've given back to the computer its variable storage memory. And the job is complete. ©

Apple Hi-Res Screen Dump

Mark Russinovich

You can easily dump high-resolution graphics pictures onto a dot matrix printer with this efficient machine language utility. It's also an ideal way to add a screen dump option to your own BASIC programs. It requires an Apple IIe or II+ computer with at least 48K RAM and an Epson or Epson-compatible printer, as well as an Epson or Epson-compatible parallel interface card that connects to slot 1. For both DOS 3.3 and ProDOS.



Have you ever wished you could print out an image that appears on the hi-res screen? This can be useful for inserting graphs or charts directly into text, or just saving interesting pictures and mathematical plots. With the program below, "DUMP," you can do all these things with minimal effort.

Using DUMP

To get started, type in Program 1 using the "Apple MLX" machine language entry program found elsewhere in this issue. Be sure you understand the instructions for using MLX before you begin entering the data from Program 1. The required starting and ending addresses for DUMP are:

Starting address: 9000
Ending address: 91DF

After you finish typing in the data, use MLX to save it to disk with the name DUMP. To install DUMP in memory for later use, just type BRUN DUMP. It loads itself into memory, protects itself from Applesoft by resetting HIMEM, and changes the ampersand vector to allow access from Applesoft.

Program 2 makes it easy to catalog disks, load hi-res pictures, view the pictures, and dump them on the printer. Just select the function you want from the menu. When you choose to print the hi-res screen, the program asks you to specify the size of the printout. There are nine sizes, ranging from a small block to a full page. (Owners of Epson MX-series printers should note that sizes 2, 3, 6, and 7 will *not* work with their printers. These sizes use codes available only on Epson's newer FX and RX models.) Next you'll be asked for a tab value. This lets you position the picture exactly where you want it. Specify the tab value in pica characters, making sure the value does not exceed the width of the page minus the width of the picture. Otherwise,

Table of DUMP Sizes

Size	Width Pica	Width Elite	Width Inch	Height Char	Height Inch
1	24	28	2.31	14	2.31
2	35	42	3.50	14	2.31
3	35	42	3.50	32	5.31
4	47	56	4.62	14	2.31
5	47	56	4.62	32	5.31
6	58	69	5.75	32	5.31
7	58	69	5.75	48	8.00
8	70	84	7.00	32	5.31
9	70	84	7.00	48	8.00

the picture might be cut off at the edge of the page, or wrap around to the middle. If you enter a tab value of zero, DUMP automatically centers the picture on the page.

To embed a picture within the text of a document, you should leave room for the pictures in your document by changing the margins. For your convenience, the accompanying table shows the widths and heights of all nine print sizes. After printing out your document, rewind the paper to position the print head about one line above the space you left for the picture. Then run Program 2 and request the size and tab value you planned for. This procedure might take a little practice before you can place a picture exactly where you want it.

Note that DUMP sets the printer to all of its default values after running. If you were using a special mode or typeface, you'll have to restore that mode after running DUMP.

DUMP With Other Programs

DUMP is especially handy when used with graphing and drawing programs, and for this reason you may want to add it to programs of your own. To do this, add a line at the beginning of the program similar to this:

```
10 PRINT CHR$(4);"BRUN DUMP"
```

Later in the program, add a screen dump option to your menu. Prompt the user for size and tab values, then enter this command:

```
&P,S,T
```

where P specifies hi-res page (1 or 2), S is the size (1–9, but remember the MX-series limitation mentioned above), and T is the tab value. Program 2 is an example of how this is done. Numbers, variables, or expressions can be used in the command. For instance, to print out hires page 1, with a size of 3 and a tab of 15, this form could be used:

```
10 A=15
20 & 1,A/5,A
```

After DUMP has finished printing the picture, it returns control to Applesoft and the program continues running.

The ampersand command can also be entered in immediate mode.

Program 1: MLX Data For DUMP

```
9000: A9 4C 8D F5 03 A9 18 8D 96
9008: F6 03 A9 90 8D F7 03 A9 9F
9010: FF 85 73 A9 8F 85 74 60 77
9018: 20 5B 91 20 F8 E6 E0 01 7A
9020: F0 07 E0 02 F0 07 20 C9 65
9028: DE A9 20 D0 02 A9 40 85 F0
9030: E6 20 BE DE 20 F8 E6 E0 26
9038: 0A B0 1A CA 8E D5 91 20 89
9040: BE DE 20 F8 E6 E0 00 D0 97
9048: 06 AC D5 91 BE 6B 91 8E C0
9050: DB 91 4C 58 90 20 C9 DE 4A
9058: A0 00 84 F9 84 FA 84 EF 03
9060: 84 FB 84 FD 84 FE 84 FF 5C
9068: AC D5 91 A9 1B 20 62 91 D1
9070: A9 33 20 62 91 B9 74 91 4B
9078: 20 62 91 B9 7D 91 8D D6 34
9080: 91 B9 86 91 8D D7 91 B9 6B
9088: 8F 91 8D D8 91 AA CA 8E 70
9090: D9 91 B9 98 91 8D DA 91 CD
9098: 98 0A A8 B9 A8 91 85 06 D5
90A0: B9 A9 91 85 07 A0 00 B1 FF
90A8: 06 99 DC 91 C8 C0 03 D0 08
90B0: F6 A9 0A 20 62 91 AC DB 89
90B8: 91 F0 08 A9 20 20 62 91 52
90C0: 88 D0 F8 A0 00 A9 1B 20 80
90C8: 62 91 B9 DC 91 20 62 91 E7
90D0: C8 C0 03 D0 F5 A5 FB 85 B7
90D8: FC A5 FC C9 C0 B0 28 A6 DD
90E0: F9 A4 FA 20 11 F4 A4 EF 1F
90E8: B1 26 A4 FF 39 A1 91 F0 65
90F0: 16 AD DA 91 A4 FD F0 0B 07
90F8: AE D6 91 4A CA D0 FC 88 1A
9100: 4C F6 90 05 FE 85 FE E6 5C
9108: FC E6 FD A5 FD CD D8 91 E7
9110: D0 C7 A5 FE AC D7 91 20 3A
9118: 62 91 88 D0 FA A9 00 85 F2
9120: FE 85 FD E6 FF A5 FF C9 B2
9128: 07 D0 06 E6 EF A9 00 85 DD
9130: FF E6 F9 D0 02 E6 FA A5 A0
9138: F9 C9 18 D0 98 A5 FA F0 1D
9140: 94 A9 00 85 EF 85 F9 85 7F
9148: FA A5 FB 6D D9 91 85 FB C4
9150: C9 C0 B0 03 4C B1 90 20 39
9158: 5B 91 60 A9 1B 20 62 91 E3
9160: A9 40 AE C1 C1 30 FB 8D AE
9168: 90 C0 60 1D 16 16 11 11 1E
9170: 0A 0A 05 05 15 15 12 15 42
9178: 12 12 12 12 12 01 01 02 25
9180: 01 02 02 03 02 03 01 03 36
9188: 03 01 01 05 05 03 03 07 1F
9190: 07 03 07 03 03 02 03 02 31
9198: 40 40 60 40 60 60 70 60 C1
91A0: 70 01 02 04 08 10 20 40 BD
91A8: BA 91 BD 91 C0 91 C3 91 C3
91B0: C6 91 C9 91 CC 91 CF 91 CB
91B8: D2 91 4C 18 01 5A 48 03 B9
91C0: 5A 48 03 4B 18 01 4B 18 AB
91C8: 01 5A 78 05 5A 78 05 4C 6D
91D0: 48 03 4C 48 03 A2 D6 C9 01
91D8: A5 8A A0 C5 C8 FF A0 31 9A
```

Program 2: DUMP Example

```
81 10 ONERR GOTO 40
52 20 D$ = CHR$ (4)
D5 30 PRINT D$"BRUN DUMP"
55 40 TEXT : HOME
13 50 HTAB 9: PRINT "**********
      **********": HTAB 9: PRINT
       "*                    *":
      HTAB 9: PRINT "* APPLE HI-
      RES DUMP *": HTAB 9: PRINT
       "*                    *":
      HTAB 9: PRINT "**********
      **********"
1F 60 PRINT : PRINT : PRINT : HT
      AB 12: PRINT "ENTER CHOICE
      :": HTAB 12: PRINT "------
      --------": PRINT : HTAB 12:
       PRINT "1) CATALOG": PRINT
      : HTAB 12: PRINT "2) LOAD
      SCREEN"
BA 70 PRINT : HTAB 12: PRINT "3)
      VIEW SCREEN": PRINT : HTA
      B 12: PRINT "4) PRINT SCRE
      EN": PRINT : HTAB 12: PRIN
      T "5) QUIT"
D7 80 VTAB 22: HTAB 12: GET A$:
      IF A$ = "1" THEN 140
2A 90 IF A$ = "2" THEN 160
F6 100 IF A$ = "3" THEN 170
74 110 IF A$ = "4" THEN 200
48 120 IF A$ = "5" THEN END
D0 130 PRINT CHR$ (7): GOTO 80
60 140 PRINT : HOME : PRINT D$"C
      ATALOG"
E3 150 PRINT : PRINT "PRESS ANY
      KEY:";: GET A$: GOTO 40
DD 160 PRINT : VTAB 22: INPUT "E
      NTER FILE NAME: ";FL$: PR
      INT D$"BLOAD"FL$",A$2000"
      : HOME : GOTO 40
81 170 POKE - 16302,0: POKE - 16
      297,0: POKE - 16304,0: PO
      KE - 16368,0
87 180 X = PEEK ( - 16384): IF X
       < 128 THEN 180
2B 190 POKE - 16368,0: TEXT : HO
      ME : GOTO 40
11 200 PRINT : VTAB 22: INPUT "E
      NTER SIZE OF DUMP (1-9):
      ";S: IF S < 1 OR S > 9 TH
      EN 200
DD 210 VTAB 24: PRINT "(0=AUTO C
      ENTER)";: VTAB 23: HTAB 1
      : INPUT "ENTER TAB SETTIN
      G: ";T: IF T < 0 OR T > 5
      0 THEN 210
7A 220 POKE - 16302,0: POKE - 16
      297,0: POKE - 16304,0: &
      1,S,T: TEXT : HOME : GOTO
       40
```

Disassembler For Atari

William Casner

This versatile utility disassembles any machine language program in memory or on disk. It can also display a memory dump and check disks for bad sectors. The program works on any 400/800, XL, or XE with at least 16K RAM for tape or 24K for disk.

Here is a BASIC utility for disassembling machine language (ML) programs and examining the contents of your Atari's memory. Type in "Disassembler" and save it to disk or tape before running it for the first time. Since this program is largely self-prompting, you should be able to use it with little or no instruction. To choose one of its three main options, press the OPTION, SELECT, or START keys as prompted. In each case, you may choose to send output to a printer rather than to the screen.

Using The Disassembler

The first option, disassembly, translates ML object code into its 6502 mnemonics. After you choose this option, the computer asks whether you wish to disassemble a particular memory area, a particular sector on the disk, or a binary file stored on disk. This allows you the freedom to disassemble virtually any ML program, even autoboot programs that normally take control of the computer as soon as you load them into memory.

The size of the disk file you can disassemble depends on the memory capacity of your computer: With 48K or 64K, you can disassemble files as large as 21K (more than 21,000 bytes). When disassembling memory, you must provide hexadecimal starting and ending addresses of the area you wish to disassemble.

The second option is a listing, or memory dump. Again, you can look at a particular memory area, a particular disk sector, or a binary file. In this case, however, the disassembler displays each byte in ASCII form rather than as a 6502 mnemonic. This function is useful for examining parts of a program that contain data rather than ML instructions.

Finally, you can scan a disk for bad sectors. After you select this option, the program checks every sector on the disk, listing the type and sector location of any errors that are found.

As you may know, CTRL-1 can be used to pause any scrolling screen display. Press Q at any input point (except the menus themselves) to return to the main menu. If you wish to abort a disassembly or memory dump, press the START key: The computer asks you to press any key to continue, then returns you to the main menu.

Take special care while typing the DATA statements in lines 1230–1310. Don't omit any commas or spaces, but don't add any extra ones, either. Mistakes could lead to incorrectly decoded mnemonics. If the program stops with an ERROR 3, 6, or 8 message in line 1220, it probably means you have a typing error somewhere in the DATA lines.

Disassembler For Atari

For instructions on entering this listing, please refer to "COMPUTE!'s Guide to Typing In Programs" published bimonthly in COMPUTE!.

```
DN 10 DIM R$(1032),SC$(128),
      ML$(36),ML2$(34),G$(31
      ),A$(20),T$(15),F$(4),
      U$(1)
ED 20 DIM P(4):P(1)=4096:P(2
      )=256:P(3)=16:P(4)=1
PO 30 DIM TY$(11):TY$="VUBC@
      ZJYXNR":SPACE=FRE(0)-6
      00:DIM S$(SPACE)
MN 40 FOR J=1 TO 36:READ B:M
      L$(J)=CHR$(B):NEXT J
BP 50 DATA 104,104,141,11,3,
      104,141,10,3,104,141,5
      ,3,104,141,4,3,169,1,1
      41,1,3
MJ 60 DATA 169,82,141,2,3,32
      ,83,228,173,3,3,133,20
      3,96
AA 70 FOR J=1 TO 34:READ B:M
      L2$(J)=CHR$(B):NEXT J
MM 80 DATA 104,104,141,105,3
      ,104,141,104,3,104,141
      ,101,3,104,141,100,3,1
      69,7
DM 90 DATA 141,98,3,162,32,3
      2,86,228,173,99,3,141,
      3,2,96
OA 100 GOSUB 1180:GOTO 200
HG 110 REM
BB 120 S=0:FOR X=1 TO 4
DI 130 A=ASC(A$(X,X))-48:IF
       A>9 THEN A=A-7
```

```
PC 140 S=S+P(X)*A:NEXT X:RET
       URN
HK 150 REM
CN 160 F=0:F$="":FOR X=1 TO
       4
KI 170 F=INT(A/P(X)):A=A-(F*
       P(X)):IF F<10 THEN F=
       F-7
AB 180 F$(X)=CHR$(F+55):NEXT
        X:RETURN
HO 190 REM
FB 200 ? "{CLEAR}":POSITION
       2,8:? "Press [OPTION] f
       or Disassembler":? "
       {6 SPACES}[SELECT] for
       Code Lister"
KP 210 ? "{6 SPACES}[START] f
       or Sector Scan":? :?
       ">";
DE 220 ON PEEK(53279)-2 GOTO
        230,220,240,980:GOTO
        220
FG 230 DIS=1:T$="Disassemble
       r":G$="{CLEAR}[ADDR ME
       MORY]{3 SPACES}[OPCODE
        OPERAND]":GOSUB 1120:
       GOTO 250
MA 240 DIS=0:T$="Lister":G$=
       "{CLEAR}{9 SPACES}[BYT
       E]{4 SPACES}[BYTE REP
       ORT]"
IA 250 ? "{CLEAR}":? :? "
       {13 SPACES}";T$
JH 260 POSITION 2,8:? "Press
        [OPTION] Memory":? "
       {6 SPACES}[SELECT] Sect
       or":? "{6 SPACES}[STAR
       T] File"
MO 270 FOR X=1 TO 50:NEXT X:
       ? :? ">";
ED 280 ON PEEK(53279)-2 GOTO
        290,280,360,500:GOTO
        280
AB 290 ADD=1:SA=SS:? :? "Sta
       rting address(4 digit
       hex)";
CM 300 INPUT A$:GOSUB 1090:I
       F LEN(A$)<>4 THEN ? "
       {3 UP}":GOTO 290
MK 310 GOSUB 120:Y=S
OC 320 ? :? "Ending address(
       4 digit hex)";
CJ 330 INPUT A$:GOSUB 1090:I
       F LEN(A$)<>4 THEN ? "
       {3 UP}":GOTO 320
BC 340 GOSUB 120:SE=S:S=Y:IF
        DIS=0 THEN 860
GK 350 GOTO 630
NI 360 ADD=0:MAXS=INT(SPACE/
       128):S=ADR(S$):GOSUB
       1120
EJ 370 ? :? "Starting sector
        (1-719)";:INPUT A$:I
       F LEN(A$)=0 THEN SS=1
       :GOTO 390
LA 380 GOSUB 1090:SS=VAL(A$)
       :IF SS<1 OR SS>719 TH
       EN ? "{3 UP}":GOTO 37
       0
JF 390 ? :? "Ending Sector (
       1-719)";:INPUT A$:IF
       LEN(A$)=0 THEN ES=SS:
       GOTO 420
PG 400 GOSUB 1090:ES=VAL(A$)
       :IF ES<SS OR ES>719 T
       HEN ? "{3 UP}":GOTO 3
       90
MG 410 IF ES-SS>MAXS THEN ?
       "Only room for ";MAXS
       ;" sectors":? "{6 UP}
       ":GOTO 370
CO 420 SECTOR=SS-1:? :? "Pre
       ss [RETURN] to begin";:
       INPUT S$
MN 430 SECTOR=SECTOR+1:A=USR
       (ADR(ML$),SECTOR,ADR(
       SC$)):IF PEEK(203)=1
       THEN 460
CA 440 ? "Sector ";SECTOR;"
       bad: ";PEEK(203):IF S
       ECTOR=ES THEN 1060
GJ 450 GOTO 430
AC 460 S$(LEN(S$)+1)=SC$:IF
       SECTOR=ES THEN SE=S+L
       EN(S$):GOTO 480
GL 470 GOTO 430
EF 480 IF DIS=0 THEN 860
GP 490 GOTO 630
ND 500 ADD=1:S$(1)=" ":S$(SP
       ACE)="I":S$(2)=S$
MB 510 SS=0:SE=0:? :? "Enter
        D#:filename.ext";:IN
       PUT A$:IF LEN(A$)=0 T
       HEN ? "{3 UP}":GOTO 5
       10
LE 520 GOSUB 1090:IF A$(2,2)
       <>":" AND A$(3,3)<>":
       " THEN ? "[BAD FILENA
       ME FORMAT]":? "{4 UP}
       ":GOTO 510
LK 530 CLOSE #2:TRAP 550:OPE
       N #2,4,0,A$:GET #2,A:
       IF A=255 THEN GET #2,
       A:IF A=255 THEN 570
PA 540 CLOSE #2:TRAP 40000:?
        :? "Not a [DOS 2.5] F
       D file":GOTO 1060
DC 550 CLOSE #2:TRAP 40000:I
       F PEEK(195)=170 THEN
       ? "[FILE NOT FOUND]":?
       "{4 UP}":GOTO 510
BK 560 ? "[UNEXPECTED ERROR]
       ERROR":GOTO 1060
KN 570 GET #2,A:GET #2,B:SS=
       B*256+A
KA 580 GET #2,A:GET #2,B:SE=
       B*256+A
DJ 590 NOBYTES=SE-SS+1:IF NO
       BYTES>SPACE THEN ? :?
        "Not Enough RAM ":GO
       TO 1060
JG 600 A=USR(ADR(ML2$),NOBYT
       ES,ADR(S$)):IF PEEK(2
       03)=255 THEN ? :? "ER
       ROR #";PEEK(203):GOTO
        1060
NK 610 CLOSE #2:S=ADR(S$):SA
       =S:SE=S+NOBYTES
EB 620 IF DIS=0 THEN 860
EH 630 IF S>SE THEN 1060
NL 640 ? G$;:IF PTR=1 THEN ?
       #3;G$
BD 650 IF PEEK(53279)=6 THEN
        1060
II 660 G$="{31 SPACES}"
HF 670 A=S-SA+SS:GOSUB 160:G
       $(1,4)=F$
HL 680 Z=PEEK(S):A=Z:GOSUB 1
       60:G$(6,7)=F$(3,4)
HG 690 IF R$(Z*4+1,Z*4+1)="
       " THEN G$(17,19)="???
       ":S=S+1:GOTO 630
DG 700 G$(17,19)=R$(Z*4+1,Z*
       4+3):U$=R$(Z*4+4):IF
       U$=" " OR U$="A" THEN
        G$(24,24)=U$:S=S+1:G
       OTO 630
DC 710 G$(24,24)="$":A=0:FOR
        J=1 TO 11:IF U$=TY$(
       J,J) THEN A=J:J=11
FM 720 NEXT J:ON A GOTO 730,
       740,750,760,770,780,7
       90,800,810,820,830:ST
       OP
GN 730 G$(27,28)=",Y":GOTO 7
       80
GN 740 G$(27,28)=",X":GOTO 7
       80
CN 750 G$(23,23)="(":G$(27,2
       9)=",X)":GOTO 780
CP 760 G$(23,23)="(":G$(27,2
       9)="),Y":GOTO 780
PE 770 G$(23,23)="#"
NM 780 A=PEEK(S+1):GOSUB 160
       :G$(9,10)=F$(3,4):G$(
       25,26)=F$(3,4):S=S+2:
       GOTO 630
KK 790 G$(23,23)="(":G$(29,2
       9)=")":GOTO 820
GB 800 G$(29,30)=",Y":GOTO 8
       20
FE 810 G$(29,30)=",X"
HO 820 A=PEEK(S+2)*256+PEEK(
       S+1):GOSUB 160:G$(9,1
       0)=F$(3,4):G$(12,13)=
       F$(1,2):G$(25,28)=F$:
       S=S+3:GOTO 630
CI 830 Z=PEEK(S+1):A=Z:GOSUB
        160:G$(9,10)=F$(3,4)
       :IF Z<128 THEN G$(22,
       22)=">":A=S-SA+SS+Z+2
       :GOTO 850
PD 840 G$(22,22)="<":A=S-SA+
       SS+Z-254
NE 850 GOSUB 160:G$(25,28)=F
       $:S=S+2:GOTO 630
MB 860 ? G$:?
EN 870 IF S>SE THEN 1060
BI 880 IF PEEK(53279)=6 THEN
        1060
FI 890 IF ADD=0 THEN ? ,S-AD
       R(S$),:GOTO 910
HG 900 A=S-SA+SS:GOSUB 160:?
        "{8 SPACES}";F$;" ";
       S;"  ";
NO 910 Z=PEEK(S):A=Z:GOSUB 1
       60:? F$(3,4);"
       {5 SPACES}";
LG 920 IF Z=125 THEN ? "
       {ESC}{CLEAR}":GOTO 97
       0
NM 930 IF Z=157 THEN ? "
       {ESC}{INS LINE}":GOTO
        970
NN 940 IF Z=158 THEN ? "
       {ESC}{DEL LINE}":GOTO
        970
KL 950 IF Z=29 THEN ? "{ESC}
       {DEL LINE}":GOTO 970
IK 960 ? CHR$(Z)
PB 970 S=S+1:GOTO 870
FG 980 ? "Insert diskette to
        scan":? :? "Press [RE
       TURN] when ready";:INP
       UT A$:GOSUB 1090
PC 990 N=0:SECTOR=0:? "
       {CLEAR}":POSITION 14,
       2:? "[SECTOR SCAN]":? :
       ? "{12 SPACES}Bad Sect
       ors"
FK 1000 ? ,"Sector","Error"
DN 1010 SECTOR=SECTOR+1:A=US
       R(ADR(ML$),SECTOR,AD
       R(SC$))
BI 1020 IF PEEK(203)<>1 THEN
        ? ,SECTOR,PEEK(203)
       :N=N+1
JN 1030 IF SECTOR=720 THEN ?
        :? ,N;" [BAD SECTORS
       ]":GOTO 1060
AI 1040 IF PEEK(53279)=6 THE
       N 200
MB 1050 GOTO 1010
NP 1060 ? :? "Press any key
       to continue":? ">";
DO 1070 IF PEEK(764)=255 THE
       N 1070
GG 1080 POKE 764,255:GOTO 20
       0
```

```
EN 1090 IF LEN(A$)=0 THEN 11
        10
IK 1100 IF A$(1,1)="Q" THEN
        POP :GOTO 200
KD 1110 RETURN
KB 1120 PTR=0:? "Do you wish
         to print(Y/[]";
NM 1130 IF PEEK(764)=43 THEN
         PTR=1:GOTO 1150
DJ 1140 IF PEEK(764)=255 THE
        N 1130
KC 1150 POKE 764,255:IF PTR=
        1 THEN CLOSE #3:TRAP
         1170:OPEN #3,8,0,"P
        :":TRAP 40000
JK 1160 ? :? :RETURN
PB 1170 CLOSE #3:TRAP 40000:
        ? :? "[illegible]
        [illegible]":GOTO 10
        60
AI 1180 ? "{CLEAR}[illegible]
        [illegible]
        [illegible]":? :?
PF 1190 ? "{7 SPACES}PLEASE
        WAIT...."
OG 1200 R$(1)=" ":R$(1032)="
        ":R$(2)=R$
MM 1210 SC$(1)=" ":SC$(128)=
        "":SC$(2)=SC$
PI 1220 FOR X=0 TO 255:READ
        F$:R$((X*4)+1,(X*4)+
        4)=F$:NEXT X:RETURN
OG 1230 DATA BRK ,ORAB,,,,OR
        AZ,ASLZ,,PHP ,ORA@,A
        SLA,,,ORAN,ASLN,,BPL
        R,ORAC,,,,ORAU,ASLU,
        ,CLC ,ORAY,,,,ORAX
KH 1240 DATA ASLX,,JSRN,ANDB
        ,,,BITZ,ANDZ,ROLZ,,P
        LP ,AND@,ROLA,,BITN,
        ANDN,ROLN,,BMIR,ANDC
        ,,,,ANDU,ROLU,,SEC ,
        ANDY,,,
AP 1250 DATA ANDX,ROLX,,RTI
        ,EORB,,,,EORZ,LSRZ,,
        PHA ,EOR@,LSRA,,JMPN
        ,EORN,LSRN,,BVCR,EOR
        C,,,,EORU,LSRU,,CLI
        ,EORY,,,
IA 1260 DATA EORX,LSRX,,RTS
        ,ADCB,,,,ADCZ,RORZ,,
        PLA ,ADC@,RORA,,JMPJ
        ,ADCN,RORN,,BVSR,ADC
        C,,,,ADCU,RORU,,SEI
        ,ADCY,,,
HB 1270 DATA ADCX,,,,STAB,,,
        STYZ,STAZ,STXZ,,DEY
        ,,TXA ,,STYN,STAN,ST
        XN,,BCCR,STAC,,,STYU
        ,STAU,STXV,,TYA ,STA
        Y,TXS ,,
AJ 1280 DATA STAX,,,LDY@,LDA
        B,LDX@,,LDYZ,LDAZ,LD
        XZ,,TAY ,LDA@,TAX ,,
        LDYN,LDAN,LDXN,,BCSR
        ,LDAC,,,LDYU,LDAU,LD
        XV,
EN 1290 DATA CLV ,LDAY,TSX ,
        ,LDYX,LDAX,LDXY,,CPY
        @,CMPB,,,CPYZ,CMPZ,D
        ECZ,,INY ,CMP@,DEX ,
        ,CPYN,CMPN,DECN,,BNE
        R
FC 1300 DATA CMPC,,,,CMPU,DE
        CU,,CLD ,CMPY,,,,CMP
        X,DECX,,CPX@,SBCB,,,
        CPXZ,SBCZ,INCZ,,INX
        ,SBC@,NOP ,,CPXN,SBC
        N,INCN,
ID 1310 DATA BEQR,SBCC,,,,SB
        CU,INCU,,SED ,SBCY,,
        ,,SBCX,INCX,
```

©

CAPUTE!

Modifications or Corrections
To Previous Articles

Atari Witching Hour

Goblins apparently invaded our lister program while this Halloween game from the October issue (p. 54) was printing. The mysterious {=} character in lines 1310 and 1320 should instead be the vertical line character, SHIFT-=.

Skyscape

The Commodore 64, Atari, and TI versions of this astronomy plotting program from the November issue (p. 62) do not work properly for latitudes between the equator and 24 degrees south. Trying to plot a skyscape for a location in this area—Peru or northern Australia, for example—results in an ILLEGAL QUANTITY ERROR message or a misplaced sun. In the Commodore 64 version (Program 1), the culprit is the second ABS in line 2510. The line should read as follows:

```
2510 IF ABS(LL)<24 THEN LB=40*
  INT(LL/7+.5)
```

The correction is the same for the Atari version (Program 2), except that the line number is 2540. For the TI version (Program 5), make the change to line 2440.

All About IBM Batch Files

The {CTRL-P} character which appears in Programs 2, 3, and 4 of this overview of batch files is not correct. Wherever this character appears, you should instead type whatever key or key combination produces an ESCape character, CHR$(27). If you use the EDLIN text editor from the IBM PC-DOS system disk, the proper replacement is {CTRL-V}[. That is, hold down the CTRL key and type V, then release CTRL and V and type [. Note that the left bracket ([) is in addition to any brackets that are already in the listing. For example, with EDLIN the first line of Program 3 would be typed as follows:

{CTRL-V}[[2J {CTRL-V}[[32m

Other text editors or word processors may require another combination. Check the manual for the editor you are using to see what you need to type to produce ASCII character 27.

There is also a correction for the last paragraph in the article (p. 88). The statement shown as:

```
IF .- - %1. GOTO .NOPARAM
```

should read:

```
IF .== %1. GOTO :NOPARAM
```

64 Color Plotter

There are no errors in this graphics utility program from the "64 Multicolor Graphics Made Easy" article in the October issue (p. 90). However, there was one point that the article failed to make completely clear: Programs with "Color Plotter" commands work *only if they are typed in while Color Plotter is active.* If you type in a program containing Color Plotter commands—for example, Program 2 from the article—in regular BASIC, then activate Color Plotter, the program appears correct when you list it, but will not run. Instead, all Color Plotter commands will cause syntax errors. You can convert the faulty program statements to true Color Plotter statements by activating Color Plotter, listing the problem line on the screen, moving the cursor to that line, and pressing RETURN. Always be sure that Color Plotter is active before typing in any programs using its special commands. And remember that you have to reactivate Color Plotter each time you press RUN/STOP–RESTORE.

©

Classified

SOFTWARE

TWO NEW DISKS FOR 64-128
1-Electronic Encyclopedia (first of series) menu select from OHMS thru FILTERS- all needed formulas + explanations.
2-A new personal sci-fi game - great sounds - very interactive - uses your name.
$29/disk. Both for $55 + $3 p/h per order.
TECH ED LTD., RFD #2, Exeter, NH 03833.

COMMODORE: TRY BEFORE YOU BUY. Top 25 best-selling games + classics, new releases. Visa, MasterCard. Free brochure. Rent-A-Disk, 908 9th Ave., Huntington, WV 25701. (304) 522-1665

ATARI ST USERS - Life organizer & entertainment jackpot. Big software package. Write: MC, 94 Macalester Bay, Winnipeg, Manitoba R3T 2X5 Canada

TIM, The Investment Manager. COMEX, gold and silver management. FUN-ANL stock analysis program. C64. All three $19.95 or write for free details to: Author's Club Software, 6027 S. High, Suite 410, Oklahoma City, OK 73149

TI-99/4A QUALITY SOFTWARE for Business, Home and Entertainment ** BONUS Software Offer! ** Send for FREE Catalog to MICRO-BIZ HAWAII, Box 1108 Pearl City, HI 96782

Free Educational Software Catalog
158 pp, color pictures, 1000+ programs for Com., Apple, IBM, Atari. Send name & address to: Interstate Software, P.O. Box 8952, Boise, ID 83707, (208) 342-3347

ORGANIZE YOUR GENEALOGY with BRANCHES on your C64 & 1541 drive. Send $14.95 to Inventure Soft., 3353 S. Main, Suite 101, Salt Lake City, Utah 84115

TI-99/4A Software/Hardware bargains. Hard-to-find items. Huge selection. Fast service. Free catalog.
D.E.C., Box 690, Hicksville, NY 11801

GOMUKO — Compiled basic program 15 by 15 board, Unmove, load/save game user changable logic: $14.95. Star traders for 2 to 5 players. Show off your 64 when guests come over: $14.95. Send check or m.o. to: Author's Club Software, 6027 S. High, Suite 410, Oklahoma City, OK 73149

LOTTO PICKER. Go for Million Dollar Jackpots! Picks all USA Lotto games +! PRO FOOTBALL ANALYST. Beat the points consistently and easily! They pay for themselves! IBM/C64/TI99 $35. Order 1-800-341-1950 Ext. 77. Mail Orders: RIDGE, 170 Broadway, Suite 210-C NYC, NY 10038.

FREE PROGRAMS! FREE PROGRAMS! C64/C16/+4/V20/TI99-4A/TIMEX 1000/2068/IBM pc/TRS80III/4/PC3/CoCo/MC10. Send stamps! EZRAEZRA, Box 5222 TDE, San Diego, CA 92105

FREE SOFTWARE CATALOG!
Call Toll-Free 1-800-554-1162, Tevex, Inc. Save ⅓ off retail prices. We carry SSI, Elect. Arts, Infocom, and many more!

NEW BARMAID IS THE LIFE OF THE PARTY!
Barmaid software shows you how to mix over 100 drinks. Add more, too. Search by full or part name of drink. Shows the ingredients & prep. Pass sobriety test before exiting. Great gift for the IBM-PC family who have everything. $34.95 ck or m.o., Visa/MC from: DPR Software, Inc., 588 Rte. 70 West, Bricktown, NJ 08723, (201) 920-8890

BIBLE QUIZ GAMES and other Bible software for the C64. Fun and learn for all ages. BIBLE-MATCH-WITS I (easy), II (hard). STAIRWAY TO HEAVEN 1, 2, 3. Colorful, animation, graphics, and sound. Each $29.95. Order or send SASE for brochure to COMPEDS, P.O. Box 147, Narrows, VA 24124

80 Column Word Processor for 48K Atari.
No additional hardware required. Any printer. Easy to use!! disk, complete manual, warranty, $69.95. FLAPS LANDING, 393 Carmen Rd., Amherst, NY 14226

COMPUTER SOFTWARE! 35%–40% OFF! Send phone number, specify software. We will call with quote. Next day shipping! C & D Assoc., Box 851, Mt. Prospect, IL 60056

ATARI-ENHANCEMENTS TO BASIC-800/XL/XE A disk based program that fixes the 800 and 800XL lockup bug and also adds many valuable commands to BASIC. Reviewed in May '85 *Analog* and June '85 *COMPUTE!* $24.95 VISA/MC/CK/MO/COD 412-627-3596 FIRST BYTE, Box 32, Rices Landing, PA 15357

WARGAME to prevent war! - STATE OF WAR situation simulator predicts conflict/peace events for all nations, alliances, guerrilla groups. Free info: Kilborn, Box 4692-Stn. E, Ottawa, Canada K1S 5H8

Free Spirit Software for the C64:
POSTMASTER - Simple, efficient, mailing list program. Disk: $19.95
BASICally SIMPLE - A quick, easy method to learn Basic programming. Disk: $20
TECHNIQUE - Learn to program graphics, animation, sound, music easily. Disk: $29.95
ITALY - Travel and educational game. Disk: $15
Order from: Free Spirit Software, Inc. 5836 S. Mozart, Chicago, IL 60629

HARDWARE

Trade in your used Commodore or Atari on a brand new C-128 or Atari ST. This offer may not be available through retail outlets. Brochure $2.00 and SASE. NEW WEST TECHNOLOGY, 4B Monroe Pkw., Box 200, Ste. 134, Lake Oswego, OR 97034

HARDWARE & SOFTWARE 30% BELOW RETAIL. Apple, Atari, C64, IBM-PC, TI-99. Over 1000 titles. Hard to find items. Send $1.00 for catalog. Specify computer. Multi-Video, P.O. Box 246, East Amherst, NY 14051

DISK SERVICE MANUAL II:
Comprehensive maintenance + repair manual on standard-bus 5.25", 8" drives, microfloppies and Apple/Commodore drives. No special software or equipment required! Over 100 labeled photos or illustrations. Manual plus free $1 catalog: $22. CONSUMERTRONICS, P.O. Drawer 537-X, Alamogordo, NM, 88310

PROWRITER, C.ITOH 8510 P, APPLE DMP, and IMAGEWRITER OWNERS: Add an extra 2K of buffer capacity. 100% compatible. Only $19.95! Send to: PRINTER-BUFFER, P.O. Box 1097, St. Louis, MO 63026

MISCELLANEOUS

64 AUTHOR'S CLUB - We get you published. Send for free details or send $25.00 (a 50% savings) to: Author's Club, 6027 S. High, Suite 410, Oklahoma City, OK 73149

HELP IS ON THE WAY!
Just call 1-800-334-0868 to get your free copy of the latest COMPUTE! Books Catalog! If you need help in getting information on all of the latest COMPUTE! book titles available plus all COMPUTE! backlist titles, call us today!

RIBBONS for ANY PRINTER at LOW PRICES!!
DELTA MICRONICS
BOX 10933, ERIE, PA 16514
(814) 455-5667

*** MR. SOFTWARE CO. ALL POPULAR TITLES ***
* Printers, Monitors, Drives, VISA, MC *
heavy discounts - Send $1.00 for catalog
11-9 Exton Complex, Somers Point, NJ 08244

FREE! USE YOUR MODEM! Call our innovative electronic shopping center, FANTASY PLAZA. VISA and MASTERCARD accepted. 300 BAUD. You've never seen anything like it! Use your Modem NOW! (818) 840-8066

EARN MONEY, PART OR FULL TIME, AT HOME with your computer. 50 page manual with forms. Money back guarantee, $9.95, JV Tech, P.O. Box 563, Ludington, MI 49431

STOCKING STUFFER
1986 Calendar. Complete history of computers, especially micros. Spiral bound, $6 ch or m.o. Same day mailing. Heat Stroke Software, Box 62171, Tucson, AZ 85734-6171

COMPUTE! Classified is a low-cost way to tell over 350,000 microcomputer owners about your product or service.

Rates: $25 per line, minimum of four lines. Any or all of the first line set in capital letters at no charge. Add $15 per line for boldface words, or $50 for the entire ad set in boldface (any number of lines.)

Terms: Prepayment is required. Check, money order, American Express, Visa, or MasterCard is accepted. Make checks payable to COMPUTE! Publications.

Form: Ads are subject to publisher's approval and must be either typed or legibly printed. One line equals 40 letters and spaces between words. Please underline words to be set in boldface.

General Information: Advertisers using post office box numbers in their ads must supply permanent address and telephone numbers. Orders will not be acknowledged. Ad will appear in next available issue after receipt.

Closing: 10th of the third month preceding cover date (e.g., June issue closes March 10th). Send order and remittance to: Harry Blair, Classified Manager, COMPUTE!, P.O. Box 5406, Greensboro, NC 27403. To place an ad by phone, call Harry Blair at (919) 275-9809.

Notice: COMPUTE! Publications cannot be responsible for offers or claims of advertisers, but will attempt to screen out misleading or questionable copy.

The Beginners Page

Tom R. Halfhill, Editor

No Strings Attached

For the past few months, we've been discussing various kinds of *numeric variables*—those that store numbers. But BASIC has a second general type of variable that's worth knowing about, too—*string variables*.

Instead of storing numbers, string variables store characters. Characters can be letters of the alphabet, the numerals 0–9, punctuation marks, the foreign letters or graphics symbols found on some keyboards, spaces, and even special codes which have meaning only to computers.

In program listings, string variables resemble regular variables, but are denoted with a trailing dollar sign, as in A$ (pronounced "A-string"). Usually, all the rules that apply to numeric variable names on your computer's BASIC also apply to string variable names. For instance, the Commodore 64 allows variable names of any length, but the computer recognizes only the first two characters for purposes of telling them apart; ditto on the Apple; the IBM also allows names of any length, but recognizes the first 40 characters; the TI allows names up to 15 characters long, recognizing all 15; and the Atari allows names of any length and recognizes all characters.

String variables are easy to set up and use. You're probably already familiar with *literal strings*, such as the word HELLO found in the following program line:

```
10 PRINT "HELLO"
```

A literal string is analogous to a numeric constant—it doesn't change. PRINT "HELLO" always prints the word HELLO. But storing a string of characters in a string variable has the same advantages as storing a number in a numeric variable—your program can manipulate the variable (and therefore the characters it stores) at will. Here's a quick example:

```
10 A$="HELLO"
20 PRINT A$
```

(Atari users should add this line: 5 DIM A$(10). We'll explain why later.) When you run this program, it prints HELLO just like the previous program. But now add these lines:

```
30 A$="HI MOM!"
40 PRINT A$
```

Even though the PRINT statement in line 40 is identical to the one in line 20, it prints a different message: HI MOM! instead of HELLO. The reason, as you may have surmised, is that we assigned a new string of characters to the string variable A$ in line 30. In effect, we changed the "value" of A$ from HELLO to HI MOM!.

This is just a taste of how string variables can be modified by a running program. We'll cover many more possibilities over the next few columns. The important thing at this point is to grasp the advantage of string variables: They allow your programs to manipulate characters, words, and sentences instead of just numbers.

A DIM Memory

Take another look at the statements in lines 10 and 30 above. These are the string variable versions of assignment statements, just as the statement A=10 assigns the value 10 to the numeric variable A. (In case you're wondering, the rarely seen keyword LET—as in LET A=10—can be used in string variable assignments, too, but is optional in almost all BASICs these days. It's customary to omit it.)

When you assign a string of characters to a string variable, BASIC stores the string in computer memory and uses the variable as a reference marker—sort of like the thumb tabs on the pages of a large dictionary. A program statement such as PRINT A$ tells BASIC to look up the string of characters in memory, retrieve it, and print it on the screen.

In TI-99/4A BASIC and most Microsoft-style BASICs (including those supplied with Commodore, Apple, and IBM computers), there's a limit on the length of the string that can be assigned to a string variable—255 characters. If you try to assign a longer string, you'll either get an error message or the string will be *truncated* (cut off) at the 255-character limit.

In Atari BASIC, a string can be of any length up to the limit of available program memory. On a 48K or 64K Atari with the Disk Operating System (DOS) and BASIC in memory, there's room for a string of more than 30,000 characters—although that wouldn't leave much memory for a very long program. Because the length of Atari strings is so flexible, Atari BASIC requires you to declare the maximum length of a string variable before using it in the program. Otherwise, the computer wouldn't know how much space to reserve for the string (Microsoft BASIC always knows that strings won't be longer than 255 characters).

The Atari BASIC statement for declaring a string's length is DIM(x), where x equals the maximum number of characters (DIM stands for DIMension). An example of the DIM statement is in line 5 above. It reserves memory for a string up to ten characters long, room enough for HI MOM! with a few characters to spare. The DIM statement *must* precede the first use of the string variable in the program, or you'll get an error. If you try to assign a string longer than the DIMed length, the string is truncated at the limit *without* an error message.

Next month we'll start delving a little deeper into how to use string variables in various ways in your programs. ©

Computers and Society

David D. Thornburg, Associate Editor

Another Kind Of Home Computing

At first glance, the Emergency Housing Consortium of Santa Clara County may seem to be an unlikely place to find personal computers. This agency, founded four years ago by Barry Del Buono, helps meet the emergency and long-term housing needs for residents of Santa Clara and San Mateo counties—two of the most populous counties in California's Silicon Valley.

To an outsider, the apparent affluence of this area masks its pockets of poverty—poverty that strikes quite hard, given the high cost of local housing. With rental units costing as much as $2,000 per month, many families who are down on their luck end up living in their cars or on the streets.

This is where Del Buono's agency steps in. In four years, the Emergency Housing Consortium has grown from one person to a staff of 35 people who oversee four shelters housing 600 people per night. In addition, the Consortium helps people find permanent housing and jobs.

As the agency began to grow, Del Buono contacted the Community Affairs program at Apple Computer, Inc. to apply for a corporate grant of computer equipment. He was convinced that computer technology could help his clients gain an edge on locating permanent housing. He envisioned an interagency network that would include a constantly updated list of low-cost area housing. Such a network was needed because by the time most of his people found out about a low-cost rental opportunity, it was already taken.

Apple granted four complete computer systems to the Consortium to share with three other housing agencies. The equipment included an Apple IIe computer with the extended 80-column card (expanding the memory to 128K RAM), a monitor, two disk drives, a ten-megabyte hard disk, a 1200 bps modem, and an Imagewriter printer. Apple also provided numerous pieces of its own software, as well as some products from other manufacturers (such as HabaMerge).

From Fast Food To Figures

The Apple Community Affairs grants are awarded primarily to nonprofit groups interested in using microcomputer networks to communicate and share information with other groups that have similar social objectives. Apple emphasizes the importance of cooperation between groups and the ways in which computers can help people cooperate across organizational boundaries.

When Apple provided the Consortium with the equipment and support it needed, the database envisioned by Del Buono became a reality. The legwork was done by volunteers and by the homeless clients themselves. "Pretty soon we were coming up with incredible stuff," says Del Buono. "We had the information available on a daily basis, and it was being updated all the time. Walk-ins could now come to our center and, in a short time, could walk out again with a list of appropriately priced rentals."

The computers became useful in other ways, too. Because the machines also store information about the Consortium's clients, it's easy to compile detailed statistics on them. This type of information is important to an agency that obtains funding from public sources.

Perhaps more importantly, the computers have provided opportunities for the clients themselves to learn how to use today's technology. One woman who had last worked for a fast-food restaurant is now the agency's statistician. She is so good at her job that she recently led a workshop at Apple. Other formerly homeless people working for the Consortium are also acquiring job skills that are transferable to the private sector. They are seeing how access to technology has a direct impact on improving their lives. This helps them recognize the importance of developing appropriate job skills in the information age.

Fringe Benefits

Meanwhile, thanks to the combined efforts of the clients and volunteers, the Consortium's constantly updated housing list is so valuable that it's now being sold to other agencies on a subscription basis. Even corporations are calling the Consortium to get rental information for their new employees!

Del Buono is convinced the Consortium couldn't be what it is today without the help of computers. His agency is decentralized, operating four shelters in two counties, and is linked to other agencies as well.

Above all, Del Buono has shown that computer technology can benefit the very poor—to create a concrete product that improves their quality of life. "When you don't have a lot of money, you need a competitive edge," he says. "That's what we get with the computer."

For more information on the Apple Community Affairs program, contact Fred Silverman at Apple Computer, 20525 Mariani Avenue, Cupertino, CA 95014. Tax deductible donations can be made to the Emergency Housing Consortium of Santa Clara County, P.O. Box 2346, San Jose, CA 95109. ©

Pieces Of Our Past—The Computer Puzzle

Last month I told the story of my "Phantom Programmer," Hunter Baker, a high school student I recruited to organize my attic and computer room. The story ended with me nervously charging into the computer room waving a machete in the middle of the night after mistaking Hunter and his friend, Amy Powell, for burglars. Actually, they were working on a school project for National History Day: a history trivia game for the IBM computer.

Hunter and Amy entered their program in the regional History Day competition and won first place in the Senior Media Presentation category. And no wonder! They had spent dozens of hours collecting hundreds of history questions and typing them into the computer, where they were stored as six random data files representing six question categories: Presidents, Places, Historical Figures, U.S. Constitution, Wars and Battles, and Trivial Trivia. And while Hunter was writing a program that managed all the questions, Amy was using Mouse Systems' *PC Paint* to create seven beautiful picture screens—a title screen and a screen for each category.

Confident after their victory in the regional competition, Hunter and Amy took their history trivia game—called "Pieces of Our Past"—to the state competition at Lynchburg College, in Lynchburg, Virginia.

But the two young people received a rude shock. The state judges said their program was not a media project at all, and gave them low grades in almost every category. One judge wrote that the project "shows no work." Another judge gave Hunter and Amy 0 out of a possible 15 points for "Quality of the Medium." Several judges gave the program low grades for historical accuracy, yet every one of Hunter and Amy's questions and answers came from reliable sources such as textbooks and encyclopedias.

Worst of all, the judges refused to interact with the program. During the judging they sat in their chairs, far away from the computer screen and keyboard, and declined to come any closer—even when Hunter and Amy invited them. Later, one judge wrote on the judging sheet: "Not effective media presentation. I couldn't see the screen."

Hunter and Amy returned from the History Day competition disappointed and bewildered. They had put an enormous amount of work into their project. They had come up with an innovative approach to learning history facts, and they had demonstrated a mastery of their medium. Hunter's program made use of random data files, elaborate graphics (created, pixel by pixel, by Amy), and music. By storing the pictures in the IBM's video memory and the music in another memory buffer, Hunter's program was able to display a picture, play music, and build the question arrays all at the same time.

But their program lost. Why? Do history teachers fear the computer? Don't they recognize the computer as a valid educational medium, like slides, filmstrips, videotapes, or 8mm movies?

A New Media Is The Message

I think history teachers are no more afraid of computers than anyone else, but like almost everyone else, few of them see the computer as "media." And since the computer is a new form of media, with its own special needs and limitations, no one was quite prepared for Hunter and Amy's project, which was so different that it bewildered the judges, confounded the rules, and didn't fit into any of the project categories.

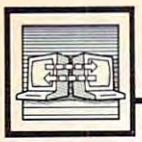
I imagine the judges had no idea how much work and original thinking went into "Pieces of Our Past." All this work was stored, invisibly and electronically, inside the computer as hundreds of lines of code, computer records, and screen maps.

And the judges were not prepared to *interact* with a media project. In the past, they had sat back, passively, and been informed, educated, or entertained. Now they were being asked to sit down in front of an unfamiliar keyboard, read the display screen, and *answer questions without any preparation.* What a fright! They might have pressed the wrong key and looked foolish. Or worse, they might have answered one of the history questions incorrectly in front of their colleagues (all fellow history teachers, instructors, and professors).

Everyone—Hunter, Amy, the judges—was burned by this experience. In the future, I doubt if Hunter and Amy will be quite as innovative or work quite as hard or independently on a project like this. And I know the judges feel bad, too. They saw merit in Hunter and Amy's project, but they didn't understand it, and they didn't know how to compare it with the other projects, or rate it according to the rules of the competition.

This was just a small incident, but I fear similar ones are occurring all over the U.S. when young people try to incorporate computers into projects that baffle and confuse their elders. Bright, self-motivated young people can come up with all sorts of ingenious uses for computers that many of us older folks have never dreamed of. But I'm worried that we may not be ready for them when they do.

What do you think? Have you had any similar experiences? Please write me care of COMPUTE!. ©



Telecomputing Today

Arlan R. Levitan

In Pursuit Of Lower Phone Bills

For months I'd been bugging a friend about his reluctance to add a modem to his home computer. Then, while visiting a computer store one day, I was busy inspecting a surge protector designed to protect the surge protector I already own when something caught my eye. It was a modem that would work with John's Commodore 128, complete with software for only $39.95. I walked over to him and waved the modem package slowly back and forth before his eyes for maximum hypnotic effect.

"That sure is a good price for a modem," John admitted. "But wouldn't I end up paying at least that much every month in phone bills and information service charges?"

He had me there. I recalled my own introduction to telecomputing and the trauma induced by various bills totaling over a hundred dollars for an uncontrolled spree of telecomputing.

Node-To-Node Networking

Sound like a familiar complaint? Now there's a solution. How would you like unlimited access to hundreds of computer bulletin boards all over the country for a flat fee of $25 a month?

If you live in the metropolitan areas of Atlanta, Boston, Chicago, Dallas, Denver, Detroit, Houston, Los Angeles, New York, Philadelphia, San Francisco, or Washington D.C., such a service is available. It's called PC Pursuit, and it's marketed by GTE Telenet, one of the giants of the packet-switching business.

What's packet switching? It's a system used daily by hundreds of businesses that have centralized computer systems linked to branch offices in different cities. Rather than leasing expensive data lines to link each branch office to the central computer, they call a local node or connection point that hooks into a special long-distance network. The call is routed through the packet-switching network to another node that is local to the firm's central computer.

The vast bulk of data traffic on packet-switching networks occurs during the business day. Although the networks are also used by people accessing commercial information services during off-hours, there's still a lot of extra capacity. PC Pursuit is an attempt by GTE Telenet to make productive use of those idle resources. Here's how it works:

Registered users call a special access number via their modem and computer. When the connection is established, the PC Pursuit system asks for their phone number, the city they wish to call, and the phone number they're trying to reach. Next, the system disconnects, temporarily freeing the phone line. Within 20 seconds, the system calls back. The user re-establishes the modem link, and then the system rings the number of the BBS. If a computer answers, PC Pursuit reports that the connection is complete. It's as if the user had directly called the remote computer himself. While the process may sound somewhat complicated, it actually requires only three pieces of information from the caller, and takes only about a minute.

A Few Limitations

For the most part, PC Pursuit works well. I spent my first evening calling BBSs in Los Angeles, Houston, and Dallas that I had been limiting my use of to keep my long distance bill from resembling the national debt. The quality of the connections is quite good, and the few noisy lines I've encountered can probably be blamed on a poor local connection at either end of the telecomputing link.

The cost savings can be significant, especially if you're a heavy user. I figured that the cost of making all of my PC Pursuit calls for the first month alone would have been well over $200. Prospective users must consider whether the one-time $25 registration and monthly $25 usage fees will actually save money.

PC Pursuit does have its limitations. You can use the service only from a single, registered phone number, typically the home number your computer is connected to. The service is offered only during Telenet's off-hours, from 6 p.m. to 7 a.m. Monday through Friday, and on weekends from 6 p.m. Friday to 7 a.m. Monday. Each PC Pursuit connection can last only 60 minutes at a time (you can, however, make multiple calls to the same number).

Since it takes longer to make a call and is a somewhat more complicated process, it is more difficult to use the redialing routines within terminal programs to bust the busy signals of the most popular bulletin boards. And although PC Pursuit tells you if the requested number is busy, the actual call cannot be monitored via a speaker on direct-connect modems. That makes it impossible to hear recorded messages informing you that a line has been disconnected or its number changed.

Also, packet switching reduces the speed of the telecomputing link. I clocked my average PC Pursuit connection at just a little less than 1000 bps (bits per second), even though I was using a 1200 bps modem. Things slow down even more when transferring files with protocols such as XMODEM—I clocked the speed at 720–950 bps.

If you're interested in more information, call GTE's bulletin board at 1-800-835-3001. Or, if you prefer to talk to a human, call 1-800-368-4215. ©

Programming the TI

C. Regena

Christmas Graphics

Try this special Christmas program. (It can only be typed in on a TI-99/4A console.)

We Three Kings

```
100 REM  WE THREE KINGS
110 CALL CLEAR
120 T=375
130 CALL SOUND(2*T,494,2,392,6,165,8)
140 CALL CHAR(152,"0001010303FF7F1F")
150 CALL CHAR(153,"07070F1F1C30304")
160 CALL SCREEN(2)
170 CALL CHAR(154,"8080C0C0E0FFFEF8")
180 CALL CHAR(155,"E0E0F078180C0C02")
190 CALL CHAR(33,"00000005155DF7F78")
200 CALL SOUND(T,440,2,370,6,165,9)
210 CALL CHAR(34,"020666606067773F1")
220 CALL CHAR(35,"0000000040C0E0F")
230 CALL SOUND(2*T,392,2,330,6,165,8)
240 CALL CHAR(36,"E0C0D29293333373")
250 CALL CHAR(37,"0107010001010101")
260 CALL CHAR(38,"50F0C0103080B0B")
270 CALL CHAR(39,"0100010307070404")
280 CALL CHAR(40,"F000808898C86")
290 PRINT TAB(10);"!"
300 CALL SOUND(T,330,2,196,6,165,9)
310 CALL CHAR(41,"7B7B7BFBFBFBFBFB")
320 CALL CHAR(42,"808080E0FCF8F8F")
330 CALL SOUND(T,370,2,311,6,123,9)
340 CALL CHAR(43,"0303070604010307")
350 CALL CHAR(44,"90000020F0F0F0F")
360 CALL SOUND(T,392,2,311,7,123,9)
370 CALL CHAR(45,"0307070707070F0F0F")
380 CALL CHAR(46,"80C0C0C0E0E0E0F")
390 CALL SOUND(T,370,2,311,6,123,8)
400 CALL CHAR(47,"0000080818183839")
410 CALL CHAR(48,"F9F9F8F8F9FDFDFC")
420 CALL SOUND(2*T,330,2,196,6,165,8)
430 PRINT TAB(10);CHR$(34)
440 PRINT TAB(8);"# $  %&"
450 CALL CHAR(49,"F6E6E040089CFEFE")
460 CALL CHAR(50,"0707070000E0F01")
470 CALL CHAR(51,"F0F0F8FC7E7C3911")
480 CALL CHAR(52,"0000000001C3F8F")
490 PRINT TAB(7);"'( )* +,"
500 CALL CHAR(53,"0000000030F8FC")
510 CALL CHAR(54,"0000010307070718")
520 CALL CHAR(55,"1F9FCFC7E7F3F9FC")
530 CALL SOUND(2*T,494,2,392,6,165,8)
540 CALL CHAR(56,"F0F0E6F0E0C3CF1F")
550 CALL CHAR(57,"39190949E1F1F9F9")
560 CALL CHAR(58,"FDFDFDFDFDFDFDFD")
570 CALL CHAR(59,"7C0CE0FCFFFFFFFF")
580 CALL CHAR(60,"0001030303837C7C7")
590 CALL SOUND(T,440,2,370,6,165,9)
600 CALL CHAR(61,"E7F3F8FCFEFFFFFF")
610 CALL CHAR(62,"C0E6FE7E3F1F8FC7")
620 CALL SOUND(2*T,392,2,330,6,165,8)
630 CALL CHAR(63,"070F00383F000F3F")
640 CALL CHAR(64,"FFFFFF3F8000FEFE")
650 CALL CHAR(65,"80C0C0D0102C4C1C")
660 PRINT TAB(7);"-./01 2345"
670 CALL CHAR(66,"0303070391C1F3F")
680 CALL CHAR(67,"FCFEFFFFFF1FC0CE")
690 CALL SOUND(T,330,2,196,6,165,9)
700 CALL CHAR(68,"FCFEFFFFFF1FC0CE")
710 CALL CHAR(69,"0000010101091939")
720 CALL SOUND(T,370,2,311,6,123,8)
730 CALL CHAR(70,"7BF9F9F8F8FCFEFF")
740 CALL CHAR(71,"F8F0E0C3070F1F3F")
750 CALL SOUND(T,392,2,311,5,123,8)
760 CALL CHAR(72,"9F9F9F9F9F9F9F9F")
770 CALL CHAR(73,"F9F9F9F1F1F1F3F3")
780 CALL SOUND(T,370,2,311,6,123,9)
790 CALL CHAR(74,"FFFFFFFFFFFFF")
800 CALL CHAR(75,"E7E7E7EFEF870723")
810 CALL SOUND(2*T,330,2,196,6,165,8)
820 CALL CHAR(76,"FFFFFFFFFFFFFFFF")
830 CALL CHAR(77,"E1F0F8FCFCFEFEFE")
840 CALL CHAR(78,"3F3F3F3F3F3F3F3F")
850 PRINT TAB(6);"6789:;<=>?@ABCD"
860 CALL CHAR(79,"FCF8F0E0C0C1C1C3")
870 CALL CHAR(80,"1800808080")
880 CALL CHAR(81,"3F7F000007FFFFFF")
890 CALL CHAR(82,"FF9300400C0818101")
900 PRINT TAB(5);"EFGHI:JKLMNOPQRS"
910 CALL SOUND(2*T,392,4,330,8,165,10)
920 CALL CHAR(83,"3838781080808")
930 CALL CHAR(84,"79F9F9F9F9710101")
940 CALL CHAR(85,"FFFFFFFEFEFCF8F")
950 PRINT TAB(5);"TUVHW:LXLYVZ[\]"
960 CALL CHAR(86,"3F7F7F7FFFFFFFF")
970 CALL CHAR(87,"F3F3F3F3F3F3F3F3")
980 CALL SOUND(T,392,5,330,9,165,11)
990 CALL CHAR(88,"636171717078787C")
1000 CALL CHAR(89,"F7F7F7FCF8F8F1E1")
1010 CALL SOUND(2*T,440,4,370,8,147,10)
1020 CALL CHAR(90,"C2828688808080802")
1030 CALL CHAR(91,"0101010303070F1F")
1040 CALL CHAR(92,"FEFCFCF8F8F8F8F")
1050 PRINT TAB(4);"^_`LHWabcdVeHf"
1060 CALL CHAR(93,"0303030604040C18E")
1070 CALL CHAR(94,"0707070707070703")
1080 CALL SOUND(T,440,5,370,9,147,11)
1090 CALL CHAR(95,"F9F9F9F9F9F9F9F9")
1100 CALL CHAR(96,"F8F0F0F0E0E4E4C4")
1110 CALL SOUND(2*T,494,3,392,7,196,9)
1120 CALL CHAR(97,"FCFCFCFCFCFCFCFC")
1130 CALL CHAR(98,"6000000030F3F3F3F")
1140 CALL CHAR(99,"3F3F1F0080E1FBFA")
1150 PRINT TAB(4);"g_hLHijLNklmno"
1160 CALL CHAR(100,"C3870F3F7F7F7F75")
```

```
1170 CALL CHAR(101,"337373F
     3F3F3F3F3")
1180 CALL SOUND(T,494,4,392
     ,8,196,10)
1190 CALL CHAR(102,"F0F0E0E
     0C0808")
1200 CALL CHAR(103,"2323636
     3C3C38383")
1210 CALL SOUND(T,587,2,392
     ,6,247,8)
1220 CALL CHAR(104,"C4C68E0
     E0E0E0E9E")
1230 CALL CHAR(105,"E7E7E7E
     7E0E0E0E5")
1240 CALL SOUND(T,523,2,370
     ,6,220,8)
1250 CALL CHAR(106,"FCFCFCF
     C000000F4")
1260 CALL CHAR(107,"F2F2F2F
     2F2E60404")
1270 CALL SOUND(T,494,2,392
     ,6,196,8)
1280 CALL CHAR(108,"7F7FFCF
     CF0C1031F")
1290 CALL CHAR(109,"8F0F1F1
     F1F9DBC38")
1300 CALL SOUND(T,440,3,370
     ,7,220,9)
1310 CALL CHAR(110,"F3F3F3F
     3F3F0F0F")
1320 CALL CHAR(111,"9E9C988
     08")
1330 CALL SOUND(T,494,3,392
     ,7,220,9)
1340 CALL CHAR(112,"070F1F3
     F7F7F7E7C")
1350 CALL CHAR(113,"F8F1E3C
     78E18")
1360 CALL SOUND(T,440,3,370
     ,7,220,9)
1370 CALL CHAR(114,"9C99830
     70F1F3F7F")
1380 CALL CHAR(115,"1F1F1F0
     00000707")
1390 CALL SOUND(2*T,392,3,3
     30,7,247,9)
1400 CALL CHAR(116,"E5E5E50
     000387C78")
1410 CALL CHAR(117,"F4F4F40
     0000C1C7C")
1420 CALL CHAR(118,"FFFFFF0
     00000E0E")
1430 PRINT TAB(4);"pqrUstuv
     wxyz{"
1440 CALL CHAR(119,"3C3C3C3
     C3E3E3E3E")
1450 CALL CHAR(120,"C07F3F0
     3")
1460 CALL SOUND(T,370,3,311
     ,7,123,9)
1470 CALL CHAR(121,"FFFEFEF
     000010101")
1480 CALL CHAR(122,"3830707
     0F0E0E0E")
1490 CALL SOUND(3*T,330,3,1
     96,7,165,9)
1500 CALL CHAR(156,"FF")
1510 CALL CHAR(157,"0102040
     81020408")
1520 CALL CHAR(158,"8080808
     08080808")
1530 PRINT TAB(4);"| } ~"
1540 PRINT TAB(4);"|"
1550 CALL HCHAR(2,27,152)
1560 CALL HCHAR(3,27,153)
1570 CALL HCHAR(2,28,154)
1580 CALL HCHAR(3,28,155)
1590 CALL CHAR(123,"F87C7C3
     C3E1F1F0E")
1600 CALL CHAR(124,"3030303
     03030303")
1610 CALL CHAR(125,"FFFCFCF
     8F0F0E0E")
1620 CALL SOUND(2*T,370,2,2
     94,6,220,8)
1630 CALL CHAR(126,"707070F
     0F0F0F0F8")
1640 CALL CHAR(127,"F8F1F1E
     1E1E0E0E")
1650 CALL CHAR(128,"FCF8F0F
     0F0F0F0F")
1660 CALL HCHAR(22,11,127)
1670 CALL HCHAR(22,12,128)
1680 CALL CHAR(129,"C0C0C0C
     0C0C0C0C")
1690 CALL CHAR(130,"1F1F0F0
     F07070703")
1700 CALL SOUND(T,440,2,262
     ,8)
1710 CALL CHAR(131,"0103030
     303030303")
1720 CALL CHAR(132,"C0C0C08
     0808")
1730 CALL SOUND(2*T,392,0,2
     94,6,196,8)
1740 CALL COLOR(16,16,1)
1750 CALL CHAR(133,"0E0C1C1
     C383060E")
1760 CALL CHAR(134,"0000010
     103030307")
1770 CALL CHAR(135,"E0C0C08
     08")
1780 CALL HCHAR(22,13,129)
1790 CALL CHAR(136,"78783C1
     C0C030606")
1800 CALL CHAR(137,"6060404
     0404")
1810 CALL SOUND(T,392,1,294
     ,7,196,9)
1820 CALL HCHAR(2,23,156,3)
1830 CALL VCHAR(4,28,158,5)
1840 CALL HCHAR(4,26,157)
1850 CALL HCHAR(5,25,157)
1860 CALL SOUND(2*T,392,0,2
     94,6,196,8)
1870 CALL CHAR(138,"0303030
     10101")
1880 CALL CHAR(139,"8080808
     08080C0C")
1890 CALL CHAR(140,"0606030
     C0C1C1C3C")
1900 CALL HCHAR(22,14,130)
1910 CALL HCHAR(22,16,131)
1920 CALL CHAR(141,"0103070
     F0F0E0E0C")
1930 CALL CHAR(142,"C0C08")
1940 CALL SOUND(T,294,1,247
     ,6,196,8)
1950 CALL CHAR(143,"30383C3
     C0E")
1960 CALL CHAR(144,"0603030
     F0F070707")
1970 CALL SOUND(2*T,392,1,2
     47,7,165,9)
1980 CALL CHAR(145,"0000000
     00080C0E")
1990 CALL CHAR(146,"000080C
     0E0E")
2000 CALL CHAR(147,"E0F0703
     8")
2010 CALL HCHAR(22,17,132)
2020 CALL HCHAR(22,18,133)
2030 CALL CHAR(148,"C0C0C0C
     0F0783C1C")
2040 CALL CHAR(149,"3C1C0E0
     707")
2050 CALL SOUND(T,330,1,262
     ,6,131,8)
2060 CALL CHAR(150,"1C1C180
     8")
2070 CALL HCHAR(23,7,134)
2080 CALL HCHAR(23,8,135)
2090 CALL HCHAR(23,10,136)
2100 CALL HCHAR(23,11,137)
2110 CALL SOUND(2*T,392,1,2
     94,6,165,8)
2120 CALL HCHAR(23,12,124)
2130 CALL HCHAR(23,13,129)
2140 CALL HCHAR(23,14,138)
2150 CALL HCHAR(23,15,139)
2160 CALL HCHAR(23,16,140)
2170 CALL HCHAR(23,17,141)
2180 CALL HCHAR(23,18,142)
2190 CALL HCHAR(24,6,143)
2200 CALL HCHAR(24,7,144)
2210 CALL HCHAR(24,8,145)
2220 CALL HCHAR(24,10,138)
2230 CALL HCHAR(24,11,146)
2240 CALL HCHAR(24,12,143)
2250 CALL HCHAR(24,13,147)
2260 CALL HCHAR(24,15,148)
2270 CALL HCHAR(24,16,149)
2280 CALL HCHAR(24,17,150)
2290 CALL SOUND(T/2,9999,30
     )
2300 CALL SOUND(2*T,392,2,2
     94,7,165,9)
2310 CALL SOUND(T,392,4,294
     ,9,165,11)
2320 CALL SOUND(2*T,392,2,2
     94,7,165,9)
2330 CALL SOUND(T,294,2,247
     ,6,196,8)
2340 CALL SOUND(2*T,392,2,2
     47,6,165,8)
2350 CALL SOUND(T,330,2,262
     ,6,131,8)
2360 CALL SOUND(2*T,392,2,2
     94.6,196,8)
2370 CALL SOUND(T,9999,30)
2380 CALL SOUND(2*T,392,3,2
     47,7,165,9)
2390 FOR C=1 TO 15
2400 CALL COLOR(C,16,1)
2410 NEXT C
2420 CALL SOUND(T,392,4,247
     ,8,165,10)
2430 CALL SOUND(2*T,440,2,3
     70,6,147,8)
2440 CALL SOUND(T,494,2,370
     ,7,147,9)
2450 CALL SOUND(2*T,523,1,3
     92,5,131,8)
2460 CALL SOUND(T,494,1,392
     ,5,196,7)
2470 CALL SOUND(2*T,440,1,3
     92,6,147,8)
2480 CALL SOUND(T,494,2,370
     ,6,147,9)
2490 CALL SOUND(2*T,392,2,2
     47,6,196,8)
2500 CALL SOUND(T,392,3,294
     ,7,196,9)
2510 CALL SOUND(2*T,392,2,2
     47,6,196,8)
2520 CALL SOUND(T,294,2,196
     ,7,123,8)
2530 CALL SOUND(2*T,392,2,3
     30,6,131,8)
2540 CALL SOUND(T,330,2,262
     ,6,131,9)
2550 CALL SOUND(3*T,392,2,2
     94,6,247,9)
2560 CALL COLOR(16,12,1)
2570 CALL COLOR(16,16,1)
2580 CALL KEY(0,K,S)
2590 IF S<1 THEN 2560
2600 CALL CLEAR
2610 PRINT "HAVE A HAPPY HO
     LIDAY SEASON!":::::
2620 END                    ©
```

INSIGHT: Atari

Bill Wilkinson

The Hidden Power Of Atari BASIC

This month we're going to look at good old Atari BASIC. For once, though, I'm not going to talk about its problems. Instead, I'm going to tell you about a few of its many virtues. If you've been reading my column since it first appeared in the September 1981 issue of COMPUTE!, then some of this may seem repetitive; but it's time to introduce newcomers to some of this material.

Unfortunately, I am beginning to see more and more poorly written Atari BASIC programs. Generally, what happens is that someone not too well-versed in Atari BASIC attempts to translate a program from another computer's BASIC and botches the job. The last straw, for me, was a recently released book which is full of CAI (Computer Assisted Instruction) programs. All the programs do much the same thing, and all the programs are...well, just a lot of work for so little value.

Now, I'm all for using a computer for drill and practice, even though most of the educational programs which do this are dull and unimaginative (and often overpriced). But even the plainest of CAI programs can at least free up a teacher or parent for 20 or 30 minutes while a student is checking his or her knowledge. And if all you want your CAI program to do is ask questions and wait for a response, then all such programs can be essentially the same. So that's what I'm going to give you this month: a "formula" program for drill and practice.

I also mentioned that we would look at some of the virtues of Atari BASIC, so let's do that first. Among microcomputer BASICs, Atari BASIC is nearly unique in its flexibility in the use of GOTO, GOSUB, and RESTORE. Specifically, each of these statements accept any numeric expression as the line number they reference. Combined with Atari BASIC's variable-name flexibility, this means you can code such oddities as:

```
GOSUB CALCULATEGROSSPAY
```

and

```
RESTORE 20000+10*CURRENTROOM
```

Most Atari BASIC books refer to these capabilities briefly, if at all. But there is some real hidden power here, as we are about to find out. Rather than belabor the point, let's take a look at the accompanying listing and analyze it a step at a time.

Using Variables As Labels

Line 1010 is fairly obvious, so let's start with lines 1060 to 1080. The variables being set here are actually going to be used as labels, the targets of GOTO and GOSUB statements. The only thing you have to be careful of with this method is renumbering—some renumbering utilities warn you when they encounter a variable being used as a label, and some don't.

Now, after setting the DATA pointer in line 1090, we get a line of DATA, assigning the first byte to the variable TYPE$. The action we take next depends on what type of line we got. We use an exclamation point to indicate a screen clear is needed, a colon for an extra blank line, and a period to flag an ordinary text line. In any of these cases, we print the rest of the line and get another one. If the type is an asterisk, the program halts. If the type is a question mark, then it's time for the student to answer.

At this time, let's look at the DATA in lines 10000–10003. The first line begins with an exclamation point, so the screen is cleared and it is printed. Then the colon asks for a blank line before the next line is displayed. Finally, the question mark tells the program to ask for a response. But what's the rest of that funny stuff: 1=,Y,0,10010?

Back at lines 1200–1260, you can see that the digit (a 1 in line 10002) tells the number of possible answers to the question, and the next character indicates the type of answer which is acceptable (the equal sign here asks for an exact match). The program then prompts the user for an answer (the #16 suppresses the INPUT prompt) and prepares to test its validity. The loop in 1310–1360 checks each valid answer against the user's response.

If an exact answer is needed, even the length of the answer counts. (Example: In line 10002, we have allowed only a single exact answer, the letter Y.) Another flag indicates whether the valid answer can be found somewhere in the user's response line. Line 10012, for example, passes any answer containing the word GRANT (such as MIGRANT WORKERS), so some care is needed in using this type. Finally, if none of the valid answers matches the user's response, the program falls through to lines 1400–1420.

So far, all this has been very straightforward, and it would work on almost any BASIC. Now comes the tricky stuff. Look at line 1320, where we READ numbers into the variables GOSUBLINE and DATALINE. What we're doing is establishing an action to take and a new set of DATA to access if the user's response matches a valid answer. Similarly, in line 1420 we read values to be used if no valid answer is given. Finally, the "magic" of this program is revealed in lines 1510 and 1520.

If we READ a number other than zero for GOSUBLINE, the program actually GOSUBs to that number. And, in any case, we change the DATA pointer to the

new DATALINE. If you can't predict what happens if you answer DUCK to the second question (because of the DATA in lines 10012–10014), please type this program and try it out.

Now, the real beauty of this program is that it works with almost any kind of question and answer session. It allows for multiple choice questions (use a format like ?3=,A,0,100,B,0,100,C,0,200), true/false, and so on. It provides for special help if needed (via the GOSUBLINEs). And, last but by no means least, it is expandable. You could add many different statement types, question types, or whatever quite easily. And it's all made possible thanks to Atari BASIC.

Multiple Choice Quiz

```
DL 1000 REM === INITIALIZATI
        ON ===
MF 1010 DIM LINE$(120), ANS$
        (20), TYPE$(1)
IB 1060 INEXACT=2000:EXACT=2
        100
NI 1070 MAINLOOP=1100:QUESTI
        ON=1200
MC 1080 MATCHED=1500
CC 1090 RESTORE 10000:REM wh
        ere we start
PL 1100 REM === THE MASTER L
        OOP ===
FE 1110 READ LINE$:TYPE$=LIN
        E$
FJ 1120 IF TYPE$="?" THEN GO
        TO QUESTION
DN 1130 IF TYPE$="!" THEN PR
        INT CHR$(125);
DC 1140 IF TYPE$=":" THEN PR
        INT
GN 1150 IF TYPE$="*" THEN EN
        D
CE 1160 PRINT LINE$(2)
GB 1170 GOTO MAINLOOP
ON 1200 REM === PROCESS A QU
        ESTION ===
JI 1210 QCNT=VAL(LINE$(2,2))
DI 1220 TYPE$=LINE$(3)
PL 1230 POSITION 2,20
KG 1240 PRINT CHR$(156);CHR$
        (156);
DC 1250 PRINT "Your Answer ?
        ";
FL 1260 INPUT #16,LINE$
OG 1300 REM === PROCESS THE
        ANSWER ===
NF 1310 FOR ANS=1 TO QCNT
CK 1320 READ ANS$,GOSUBLINE,
        DATALINE
BL 1330 IF TYPE$="#" THEN GO
        SUB INEXACT
JP 1340 IF TYPE$="=" THEN GO
        SUB EXACT
CD 1350 IF MATCH THEN GOTO M
        ATCHED
OL 1360 NEXT ANS
EI 1400 REM === ANSWER DOESN
        'T MATCH ===
JA 1410 REM (read error cond
        itions and fall thru
        )
PJ 1420 READ GOSUBLINE,DATAL
        INE
NO 1500 REM === ANSWER MATCH
        ED ===
FF 1510 IF GOSUBLINE THEN GO
        SUB GOSUBLINE
CO 1520 RESTORE DATALINE
GB 1530 GOTO MAINLOOP
LD 2000 REM === INEXACT MATC
        H ROUTINE ===
GK 2010 MATCH=0:ALEN=LEN(ANS
        $)
GB 2020 SIZE=LEN(LINE$)-ALEN
        +1
AL 2030 IF SIZE<1 THEN RETUR
        N
BH 2040 FOR CHAR=1 TO SIZE
GL 2050 IF LINE$(CHAR,CHAR+A
        LEN-1)=ANS$ THEN MAT
        CH=1:RETURN
CF 2060 NEXT CHAR
KJ 2070 RETURN
BN 2100 REM === EXACT MATCH
        ROUTINE ===
EO 2110 MATCH=(ANS$=LINE$)
KF 2120 RETURN
KK 10000 DATA !Ready to try
        out this program?
KP 10001 DATA :  (answer Y o
        r N)
LJ 10002 DATA ?1=,Y,0,10010
FL 10003 DATA 0,10000
PJ 10010 DATA !A tribute to
        Groucho Marx:
CK 10011 DATA :Who is buried
         in Grant's tomb?
MH 10012 DATA ?2#,GRANT,0,10
        040
HF 10013 DATA DUCK,10020,100
        30
CJ 10014 DATA 10050,10060
OH 10020 REM special sound r
        outine
KI 10021 FOR FREQ=120 TO 20
        STEP -10
CG 10022 FOR VOLUME=15 TO 0
        STEP -0.5
JK 10023 SOUND 0,FREQ,10,VOL
        UME
LF 10024 NEXT VOLUME:NEXT FR
        EQ
NI 10025 RETURN
LB 10030 DATA !You said the
        secret word!
LH 10031 DATA :You win $100.
DK 10032 DATA *
KL 10040 DATA !Great!  You g
        et the consolation
PE 10041 DATA .prize of $50.
DL 10042 DATA *
LE 10050 REM raspberry
FK 10051 FOR VOLUME=15 TO 0
        STEP -0.25
PK 10052 SOUND 0,4,80,VOLUME
        :NEXT VOLUME
NJ 10053 RETURN
KM 10060 DATA !Sorry.  You l
        ost.
DM 10061 DATA *
```

Donald B. Trivette

Diary Of A Home Application

Few of us appreciate how much time and effort goes into successful software. This month we're going to take an inside look at one of today's best-selling home programs for IBM computers—*Andrew Tobias' Managing Your Money.*

September 1982. Micro Education Corporation of America (MECA) looks for ways to enter the lucrative personal computer market. A market survey and analysis shows that home users are interested in financial software. MECA decides to develop a tutorial-type financial program for the Atari and to employ a "big name" to promote the product. Louis Rukeyser of *Wall Street Week* is contacted; he graciously declines. Another proposal goes to syndicated columnist Sylvia Porter; no reply is received. MECA looks for another name.

December 1982. Andrew Tobias, author of the best-selling book *The Only Investment Guide You'll Ever Need,* agrees to provide his name and guidance—for a percentage. (He too almost missed out when his agent, unaware of the potential profits, neglected to return MECA's calls. Eventually MECA contacted Tobias directly.)

Tobias and Jerry Rubin, the master programmer/designer and president of MECA, meet to discuss possibilities. They decide to develop the product for the IBM PC, which seems to be gaining momentum in the market. The program begins to evolve from the initial concept of a tutorial with cartoon characters and balloons of text to a more serious program that can be used to plan and record financial transactions.

Steve Wagar, a recent Yale computer science graduate, begins writing a special computer language called SEESAW (System Elegantly Enmeshing Screens And Worksheets) in which *MYM* will be programmed. Spencer Martin, taking a year off between high school and college to swim in the Olympic trials, joins the programming team. So does Jim Russell, a student at the Massachusetts Institute of Technology.

Summer 1983. Rubin gleefully demonstrates a screen to Tobias, who, knowing nothing about computers and programming, fails to appreciate its significance. "My God," he thinks, "six months and all we've got is one or two screens where I can type my name and a few numbers."

Fall 1983. The project is far enough along so that Tobias, using a *WordStar* interface to SEESAW, can write the help screens and compose the program's text—a job that eventually takes six months.

January 1, 1984. The goal for the release of *MYM* slips by while initial product testing begins in Westport, Connecticut. A group of 20 people—some experienced computer users, some not—are given copies to take home. Later they are invited to headquarters for a debriefing while Rubin, Tobias, and others watch tensely through a one-way mirror, suppressing the urge to pound on the wall and yell, "No, you idiot, not that key, the other key!"

MYM takes shape as an integrated financial program with nine sections or chapters. Chapter 1 is for new users who know nothing about computers; Chapter 2 is a reminder pad; Chapter 3 is a budget and checkbook; Chapter 4 is an income tax estimator; Chapter 5 is for insurance planning; Chapter 6 is a calculator; Chapter 7 is a portfolio manager; Chapter 8 is a net worth summary combining data from the other chapters; and Chapter 9 is a comprehensive index.

March 19, 1984. Tobias appears on the *Today* show to introduce *MYM* and the first 300 copies are shipped to dealers. A bug is uncovered and MECA replaces all 300 copies at its own expense. Another bug is uncovered and 500 copies are replaced. A WATS line with 12 customer support people is set up to answer questions and help users. MECA continues to improve *MYM* and to provide free updates to registered owners.

Summer 1984–Spring 1985. Tobias travels more than 60,000 miles promoting *MYM* in software stores, at trade associations, and on radio and TV talk shows. The program gets good reviews and IBM markets a cartridge version for the PCjr.

Summer 1985. Starting with a wish list compiled from customer suggestions, MECA begins work on version 2.0—a major update. Rubin, Martin, Russell, and four new programmers add 75 enhancements. Rubin and Russell spend weeks on an option to make the fiscal year different from the calendar year; Tobias writes an expanded 15,000-word manual. MECA engages a software-testing company which generates more than 120 Trouble Reports that ultimately have one of three resolutions: Already Fixed; To Be Fixed; and Not a Bug After All. The testing costs more than $50,000.

October 1985. *Andrew Tobias' Managing Your Money* version 2.0 ships to dealers. Current owners who signed up for the newsletter and warranty plan ($40/year) receive the update free. Other registered owners can purchase the update for $50.

(MYM 2.0 lists for $199.95 and requires an IBM PC or compatible with two disk drives and at least 192K of RAM; IBM markets version 1.0 in cartridge form for the Enhanced Model PCjr at the same price. MECA, 285 Riverside Avenue, Westport, CT 06880.) ©

What the world really needs is a 69 cent Double Sided, Double Density Diskette with a LIFETIME WARRANTY!

And DISK WORLD! has it.

Introducing Super Star Diskettes: the high quality diskette with the lowest price and the best LIFETIME WARRANTY!

In the course of selling more than a million diskettes every month, we've learned something: higher prices don't necessarily mean higher quality.

In fact, we've found that a good diskette manufacturer simply manufactures a good diskette...no matter what they charge for it. (By way of example, consider that none of the brands that we carry has a return rate of greater than 1/1,000th of 1 percent!)

In other words, when people buy a more expensive diskette, they aren't necessarily buying higher quality.

The extra money might be going toward flashier advertising, snazzier packaging or simply higher profits.

But the extra money in a higher price isn't buying better quality.

All of the good manufacturers put out a good diskette. Period.

How to cut diskette prices ...without cutting quality.

Now this discovery posed a dilemma: how to cut the price of diskettes without lowering the quality.

There are about 85 companies claiming to be "diskette" manufacturers.

Trouble is, most of them aren't manufacturers.

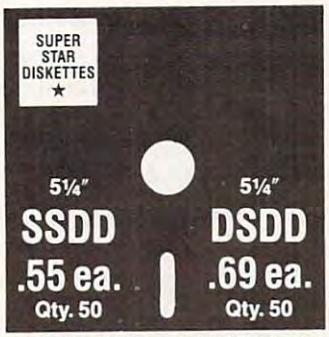
Rather they are fabricators or marketers, taking other company's components, possibly doing one or more steps of the processing themselves and pasting their labels on the finished product.

The new Eastman Kodak diskettes, for example, are one of these. So are IBM 5¼" diskettes. Same for DYSAN, Polaroid and many, many other familiar diskette brand names. Each of these diskettes is manufactured in whole or in part by another company!

So, we decided to act just like the big guys. That's how we would cut diskette prices...without lowering the quality.

We would go out and find smaller companies to manufacture a diskette to our specifications...specifications which are higher than most...and simply create our own "name brand" diskette.

Name brand diskettes that offered high quality at low prices.



Super Star diskettes are sold in multiples of 50 only. Diskettes are shipped with white Tyvec sleeves, reinforced hubs, user ID labels and write-protect tabs.

Boy, did we get lucky. Our Super Star Diskettes are the same ones you've been using for years...without knowing it.

In our search for the low priced, high quality diskette of our dreams, we found something even more interesting.

We found that there are several manufacturers who don't give a hoot about the consumer market for their diskettes. They don't spend millions of dollars in advertising trying to get you, the computer user, to use their diskettes.

Instead, they concentrate their efforts on turning out the highest quality diskettes they can...because they sell them to the software publishers, computer manufacturers and other folks who (in turn) put their name on them...and sell them for much higher prices to you!

After all, when a software publisher or computer manufacturer or diskette marketer puts their name on a diskette, they want it to work time after time, everytime. (Especially software publishers who have the nasty habit of copy-protecting their originals!)

Super Star Diskettes. You already know how good they are. Now you can buy them...cheap.

Well, that's the story.

Super Star diskettes don't roll off the boat from Pago-Pago or emerge from a basement plant just east of Nowhere.

Super Star diskettes have been around for years...and you've used them for years as copy-protected software originals, unprotected originals. Sometimes, depending on which computer you own, the system master may have been on a Super Star diskette. And maybe more than once, you've bought a box or two or more of Super Star diskettes without knowing it. They just had some "big" company's name on them.

Super Star Diskettes are good. So good that a lot of major software publishers, computer manufacturers and other diskette marketers buy them in the tens or hundreds of thousands.

We buy them in the millions.

And than we sell them to you.

Cheap.

When every little bit counts, it's Super Star Diskettes.

You've used them a hundred times...under different names.

Now, you can buy the real McCoy, the same diskette that major software publishers, computer manufacturers and diskette marketers buy...and call their own.

We simply charge less.

Super Special!

Order 50 Super Star Diskettes and we'll be happy to sell you an Amaray Media-Mate 50 for only $8.75, shipping included...a lot less than the suggested retail price of $15.95.

Regular DISK WORLD! price: $9.69 ea. + $2.00 Shpng.

DISKETTE STORAGE CASES

DISK CADDIES

The original flip-up holder for 10 5¼" diskettes. Beige or Grey only.

$1.65 ea. + .20 Shpng.

DISKETTE 70 STORAGE

Dust-free storage for 70 5¼" diskettes. Six dividers included. An excellent value.

$9.95 ea. +$3.00 Shpng.

HOURS:
Human: 8AM-6PM Central Time, Monday through Friday
Answering Machine: 6PM-8AM, All Times
MCI MAIL: 24 hours a day.

HOW TO ORDER:

ORDERS ONLY:
1-800-621-6827
(In Illinois: 1-312-256-7140)

INQUIRIES:
1-312-256-7140

FOR FASTEST SERVICE, USE NO-COST MCI MAIL: Our address is DISKORDER. It's a FREE MCI MAIL letter. No charge to you. (Situation permitting, we'll ship these orders in 24 hours or less.)

SHIPPING: 5¼" & 3½" DISKETTES—Add $3.00 per each 100 or fewer diskettes. **OTHER ITEMS:** Add shipping charges as shown in addition to other shipping charges. **PAYMENT:** VISA, MASTERCARD and Prepaid orders accepted. **COD ORDERS:** Add additional $5.00 special handling charge. **APO, FPO, AK, HI & PR ORDERS:** Include shipping charges as shown and additional 5% of total order amount to cover PAL and insurance. We ship only to United States addresses, except for those listed above. **TAXES:** Illinois residents, add 7% sales tax.

MINIMUM ORDER: $35.00.

The Super Star LIFETIME WARRANTY!

Super Star Diskettes are unconditionally warranted against defects in original material and workmanship so long as owned by the original purchaser. Returns are simple: just send the defective diskettes with proof of purchase, postage-paid by you with a short explanation of the problem, and we'll send you the replacements. (Incidentally, coffee stained diskettes and diskettes with staples driven through them don't qualify as "defective".)

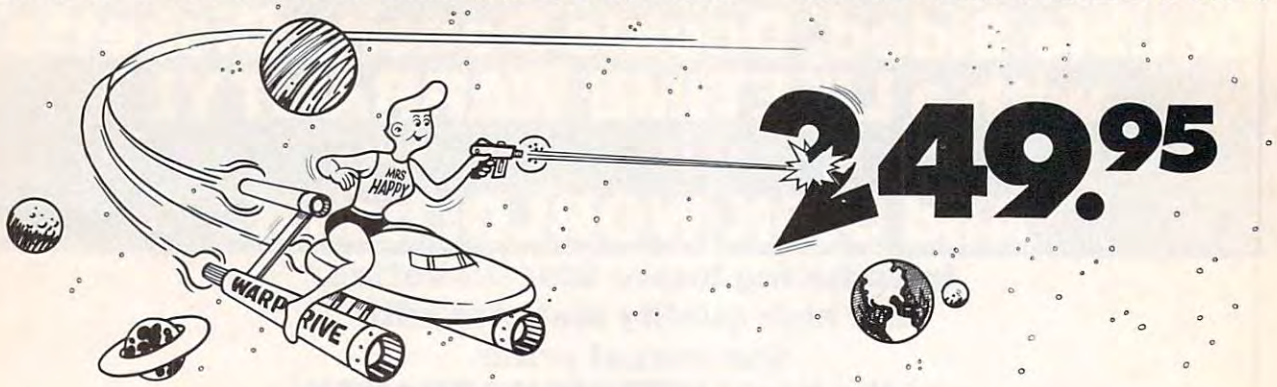
WE WILL MEET OR BEAT ANY NATIONALLY ADVERTISED PRICE ON THE SAME PRODUCTS AND QUANTITIES SUBJECT TO THE SAME TERMS AND CONDITIONS.

DISK WORLD!, INC.

629 Green Bay Road
Wilmette, Illinois 60091

ATARI DISK DRIVE OWNERS . . .

HAPPY BLASTS RETAIL PRICE—ORDER TOLL FREE!



THE FAMOUS HAPPY ENHANCEMENT NOW ONLY $149.95
for 1050 order number HC1C, for 810 order number HC8C

Makes your disk drive read and write faster, and allows you to execute the HAPPY WARP SPEED SOFTWARE. Available only for ATARI 1050 and 810 disk drives. 1050 version allows true double density plus the original single and enhanced density. PRICE INCLUDES WARP SPEED SOFTWARE BELOW, installation required.

HAPPY WARP SPEED SOFTWARE REV 7 (not sold separately)

Includes the famous HAPPY BACKUP and COMPACTOR which are the most powerful disk backup utilities available for your ATARI computer, plus MULTI DRIVE which allows high speed simultaneous reading and writing with up to 4 HAPPY ENHANCED drives, plus SECTOR COPIER which is the fastest disk copier that supports the 130XE RAMDISK, plus the WARP SPEED DOS which improves ATARI DOS 2.0s to allow fastest speed, plus HAPPY'S DIAGNOSTIC which allows comprehensive disk drive testing.

HAPPY 1050 CONTROLLER $64.95 order number HC2C

For use with HAPPY ENHANCED 1050 disk drives only. Allows easy access to HAPPY 1050 slow and fast speeds and ultimate control of disk drive write protect, including writing to disk back side and protecting valuable data disks. Printed circuit board has switches and write protect indicator LED, installation required.

GET YOUR FAVORITE HIGH SPEED DOUBLE DENSITY DOS

Both of these disk operating systems support the fastest speed with both HAPPY 810* and 1050, and with HAPPY 1050 you get true double density. WARP SPEED DOS XL is HAPPY's own version of OSS DOS XL, and includes under cartridge, under ROM and AXLON RAM disk version, and is order number HC4C at $29.95. TOP DOS version 1.5 from ECLIPSE SOFTWARE has more menu driven features, operates in all three densities, supports the 130XE RAMDISK, and is order number HC6C at $39.95. *Note: 810 requires upgrade below.

810 VERSION 7 UPGRADE $49.95 order number HU3C-XXXX

Allows older 810 HAPPIES to use newer software. Includes custom plug in IC and rev 7 WARP SPEED SOFTWARE. Same price for all HAPPY 810s registered or not. When ordering replace XXXX in part number with the serial number of your HAPPY COMPUTERS manufactured 810 board, or with a 2732 or 2532 which corresponds to the EPROM part number in your HAPPY 810 socket A102 of your side board modified HAPPY (not made by HAPPY COMPUTERS), installation required. Upgrade not needed for new 810 HAPPYS and serial number over 8000.

SUPER PACKAGE SPECIALS

Get a HAPPY 1050 ENHANCEMENT and CONTROLLER and WARP SPEED DOS XL for just $199.95 order number HS5C, or get the same with TOP DOS 1.5 instead of DOS XL for just $214.95 order number HS7C. If you already have the 1050 ENHANCEMENT you can get the HAPPY 1050 CONTROLLER and WARP SPEED DOS XL for $74.95 order number HXL9C, or get the HAPPY 1050 CONTROLLER and TOP DOS 1.5 for just $84.95 order number HTD9C. For other specials and dealer pricing call (408) 779-3830.

All prices include UPS shipping in USA, add $10.00 for shipment outside USA. California residents add sales tax. No extra charge for credit cards or COD, VISA or MASTERCARD accepted. Our toll free number is an order taking service, not our line. To ORDER ONLY call (800) 538-8157 outside California, or (800) 672-3470 inside California, ask for extension 817 and have your credit card, part number and quantities ready. Toll free hours 6 am to 12 pm Mon.–Fri., 8 am to 8 pm Sat. & Sun., Pacific Time. For answers to questions call HAPPY COMPUTERS at our number below. Office hours 9–5 Mon.–Fri. Pacific Time.

HAPPY COMPUTERS, INC. * P.O. Box 1268 * Morgan Hill, CA 95037 * (408) 779-3830

presenting . . . CAPTURE™

A NEW WAY TO UNLOCK THE POWER OF YOUR C64 OR C128*

- **CAPTURE** IS A CARTRIDGE THAT PLUGS INTO YOUR COMPUTER'S EXPANSION PORT.
- **CAPTURE** DOES NOTHING — UNTIL YOU PRESS ITS **CAPTURE** BUTTON. UNTIL THEN, A RUNNING PROGRAM CANNOT DETECT ITS PRESENCE.
- **CAPTURE** THEN TAKES CONTROL — NO IFS, ANDS OR BUTS — AND PRESENTS A MENU.
- **CAPTURE** WILL NEATLY SAVE EVERYTHING IN YOUR COMPUTER TO YOUR 1541 OR COMPATIBLE DISK DRIVE — ALL 64K OF RAM, CPU, VIC AND SID CHIP REGISTERS — EVERYTHING. IN EASY TO VIEW CHUNKS.
- **CAPTURE** WILL, IF YOU WANT, PRE-CONFIGURE YOUR COMPUTER'S RAM SO THAT ONLY MEMORY ALTERED BY YOUR PROGRAM NEED BE SAVED.
- **CAPTURE** WILL WRITE A BOOT ON YOUR DISK SO YOU CAN RELOAD AND BEGIN EXECUTION RIGHT WHERE YOU LEFT OFF.
- **CAPTURE** DOES ALL THIS AT A COST OF JUST **$39.95**

BUT HERE'S THE BEST PART

- **CAPTURE** WILL MAKE AN AUTO-START CARTRIDGE OF YOUR PROGRAM. IT'S EASY! JUST FOLLOW THE DIRECTIONS ON THE SCREEN. NOW PLUG IN YOUR CARTRIDGE AND TURN ON YOUR COMPUTER. IN LESS THAN TWO SECONDS YOUR PROGRAM BEGINS AGAIN AT PRECISELY THE POINT WHERE YOU **CAPTURE**'D IT. MAGIC!

  BESIDES **CAPTURE,** YOU NEED A *promenade* C1 AND A SUPPLY OF CPR3 CARTRIDGE KITS.

ORDERING INFORMATION

- **CAPTURE** CARTRIDGE — COMPLETE WITH INSTRUCTIONS ..... $ 39.95
- *promenade* C1 — EPROM PROGRAMMER WITH DISK SOFTWARE 99.50
- CPR3 CARTRIDGE KIT — PC BOARD, CASE AND 3 EPROMS 29.95
- DR—EPROM ERASER, TWO AT A TIME, 3 TO 10 MINUTES 34.95
- STARTER SET — **CAPTURE,** *promenade* C1 AND 1 CPR3 ..... 149.95
- DELUXE SET — **CAPTURE,** *promenade* C1, DR AND 2 CPR's 199.95

SHIPPING AND HANDLING — USA: UPS SURFACE $3.00
BLUE LABEL $5.00 NEXT DAY AIR $13.00
CANADA: AIR MAIL $7.00 OTHER FOREIGN AIR $12.00

CALIFORNIA RESIDENTS ADD APPLICABLE SALES TAX
COD ORDERS, USA ONLY, ADD $3.00

C64 AND C128 TM COMMODORE ELECTRONICS, LTD. *WHEN OPERATING IN 64 MODE

TO ORDER: TOLL FREE 800-421-7731
FROM CALIFORNIA 800-421-7748

TECHNICAL SUPPORT AND 408-287-0259
FROM OUTSIDE THE US: 408-287-0264

JASON-RANHEIM
580 PARROT STREET
SAN JOSE, CA USA 95112

PARTS / SERVICE FOR ATARI*COMPUTERS

Flat Service Rates Below Include Parts & Labor, 60-Day Warranty

800 Computer Repair	$49.50	810 Disk Drive Repair	$79.50
850 Interface Repair	$49.50	800XL Computer Repair	$49.50
600XL Computer Repair	$49.50	1050 Disk Drive Repair	$85.00
1200XL Computer Repair	$49.50	800 Keyboard Repair	$35.00

Above units repaired or exchanged with rebuildable exchange. Include $7.00 return shipping and insurance.

INTEGRATED CIRCUITS

GTIA Chip — C014805 upgrade with instructions	$11.50
10K Rev. B OS Upgrade — for 400/800 3-Chip ROM set with instructions	$10.00
C012294	$8.50
C012296	$9.50
C014795	$8.50
C014806	$9.50
C010745	$10.00
C010750	$9.50

MODULES/CIRCUIT BOARDS
complete with IC's

16KRAM Memory Module - CX853	$15.00
800 10K Rev. B OS Module	$15.00
800/400 CPU Board with GTIA	$19.50
800 Main Board	$24.50
400 Main Board	$20.00
800 Power Supply Board	$10.50
810 Data Separator Board upgrade with instructions	$25.00
810 Side Board w/o Sep. & 1771	$43.50
810 Rear Power Board	$25.00
810 Analog Board	$16.00
810 Rear Board/Analog Board Upgrade with 10-pin jumper and instructions	$37.50
800 OK Board Set	$65.00
810 Board Set	$99.50
800 48K Board Set	$79.50

BARE BOARDS

With parts lists	
850 INTERFACE BOARD	**$16.50**
Build your own interface!!	
810 Analog Board	$3.50
810 Rear Board	$5.00

DISK DRIVES, Etc.

810 Custom Disk Drive	**$145.00**
850 Custom Interface	**$79.50**
Replacement 810 Drive Mech.	$70.00
Replacement transformer for 800/400, 810, 1050, 1200XL, 1020	$15.00
800XL/600XL, 130XE Power Supply	$25.00
SAMS Service Manual for 800/400 or 800XL	$19.95
De Re Atari	$12.50
Inside Atari Basic	$6.50

SOFTWARE

Basic Cartridge	$15.00
Editor/Assembler	$15.00
Q*Bert Cartridge	$12.50
Popeye Cartridge	$12.50
Kindercomp Cart.	$10.00
Buck Rogers Cart.	$7.50
Jumbo Jet Pilot	$10.00
Crossfire Cart.	$5.00
Chicken Cartridge	$5.00

AMERICAN TV — 415 - 352-3787

Mail Order and Repair ..... **15338 Inverness St., San Leandro, CA 94579**
Retail Store ..... 1988 Washington Avenue, San Leandro, CA 94577

Terms: We accept money orders, personal checks or C.O.D.s. — VISA, MasterCard okay on orders over $20.00. No personal checks on C.O.D.

Shipping: $4.00 shipping and handling on orders under $150.00. Add $2.00 for C.O.D. orders. California residents include 6½% sales tax. Overseas shipping extra.

Prices subject to change without notice. We reserve the right to limit quantities. Sales limited to stock on hand. Foreign shipping extra.

Much more!
Sent SASE for free price list.

*Atari is a registered trademark of Atari Corp.

Super Cart

Copy Atari 400/800/XL Series Cartridges to Disk and run them from a Menu

ATARI CARTRIDGE-TO-DISK COPY SYSTEM $69.95

Supercart lets you copy *ANY* cartridge for the Atari 400/800/XL Series to diskette, and thereafter run it from your disk drive. Enjoy the convenience of selecting your favorite games from a "menu screen" rather than swapping cartridges in and out of your computer. Each cartridge copied by Supercart functions *exactly like the original*. Supercart includes:

- DISKETTE with:
  - COPY PROGRAM - Copies the cartridge to a diskette (up to 9 cartridges will fit on one disk.)
  - MENU PROGRAM - Automatically runs and displays a menu prompting user for a ONE keystroke selection of any cartridge on the disk.
- CARTRIDGE:
  - "Tricks" the computer into thinking that the original "copy protected" cartridge has been inserted.

To date there have been no problems duplicating and running all of the protected cartridges that we know of. However, FRONTRUNNER cannot guarantee the operation of all future cartridges. Supercart is user-friendly and simple to use and requires no modifications of your hardware. **PIRATES TAKE NOTE:** SUPERCART is not intended for illegal copying and/or distribution of copyrighted software. . .Sorry!!!

SYSTEM REQUIREMENTS:
Atari 400/800 or XL Series Computer / 48K Memory / One Disk Drive

Available at your computer store or direct from FRONTRUNNER. DEALER INQUIRIES ENCOURAGED
TOLL FREE ORDER LINE: (24 Hrs.) **1-800-648-4780** /In Nevada or for questions Call: **(702) 786-4600**
Personal checks allow 2-3 weeks to clear. M/C and VISA accepted.
Include $3.50 ($7.50 Foreign orders) for shipping.
FRONTRUNNER COMPUTER INDUSTRIES
316 California Ave., Suite #712, Reno, Nevada 89509 - (702) 786-4600
Others Make Claims. . .SUPERCART makes copies!!!
ATARI is a trademark of Warner Communications, Inc.

To receive additional information from advertisers in this issue, use the handy reader service cards in the back of the magazine.

WHITE HOUSE COMPUTER

P.O. Box 4025, Williamsport, PA 17701

"Where Prices Are Born, Not Raised"

TOLL FREE 1-800-351-3442

PA CALL 1-717-322-7700

— PA Residents FREE Shipping —

MONDAY - FRIDAY 9 AM - 6 PM
VISA & MC ACCEPTED 4%

POLICY: No deposit on COD orders. Free freight on all prepaid cash orders over $300 in the continental USA. APO & FPO add $5.00 per hundred. For priority mail add $8.00 per hundred. PA residents add 6% sales tax. Defective products must have Prior RA number. Schools net 15

PRINTERS

EPSON		**OKIDATA**	
RX 80	209.00	Okimate 10	179.95
RX 100	369.00	182	219.95
JX 80	479.00	84	640.95
FX 85	345.00	**192**	**349.95**
FX 185	499.00	193	525.95
LQ 1500 PAR	979.00	**COMMODORE**	
LQ 1500 SER	1039.00	MPS 801	125.00
LX 80	222.00	802	199.95
Homewriter	209.00	803	165.95
Comrex 220 Atari	199.00	DPS 1101	295.95
Comrex 220 Comm.	199.00	**STAR MICRONICS**	
LX 90	245.00	**SG-10**	**210.00**
SQ 2000	1525.00	SG-15	379.00
DX 10 Daisywheel	235.00	SD-10	321.95
DX 20 Daisywheel	319.00	SD-15	450.00
HS 80 Letterjet	359.00	SR-10	485.00
ATARI		SR-15	585.00
XTM 201	99.95	Powertype	307.00
XTC 201	109.95	**LEGEND**	
XDM 121	209.95	**1380**	**259.95**
XMM 801	169.95	**1385**	**293.95**
STC 504	139.95	1080	199.95
STD 121	219.95	808	159.95
SMM 801	279.95	**PANASONIC**	
CITIZEN		1090	187.00
MSP 10	275.00	1091	231.00
MSP 15	450.00	1092	385.00
MSP 20	450.00	1093	425.00
MSP 25	575.00	3151	455.00

INTERFACES

850 (Atari)	108.95
UPrint/port	49.95
UPrint/16k Buffer	**69.95**
UPrint/64k Buffer	89.95
Cardco G	39.95
G-Wiz	**48.95**
Apple Dumpling GX	**59.95**

PRINTER PAPER

2500 Sheets Lazor Edge	**24.95**
1000 Shts Lazor	14.95
500 Shts Lazor	9.95
Color Paper Assorted Pastels	
2500 Shts Lazor	42.95
1000 Shts Lazor	23.95
500 Shts Lazor	14.95

SOFTWARE

Print Shop	28.95
Graphics Library I	17.50
Graphics Library II	17.50

VIDEO CASSETTE RECORDER
$279.00

DISK DRIVES

ATARI	
Indus GT	**209.95**
1050	155.95
Happy 1050	319.95
Happy Enhancer	160.95
MSD Dual	459.00
C-64	
Indus GT	**198.00**
1541	195.00
1571	249.95
1572	375.95

MODEMS

C-1650	54.95
C-1670 300/1200	185.95
MPP 1064	54.95
Tele Learning	39.95
Hayes 300	149.95
Hayes 1200	385.95
Mitey Mo	59.95
Westridge	59.95
Compuserve	19.95
Micro Stuffer	95.95
MPP 1000E	**69.95**

DISKETTES

SKC	
SS/DD	10.95
DS/DD	14.95
BONUS	
SS/DD	9.50
DS/DD	13.50
MAXELL	
MD 1	15.95
MD 2	20.95

MONITORS

ZENITH	
122 A	**198.00**
123 G	**198.00**
AMDEK	
300 G	119.00
300 A	129.00
Color 300	**185.95**
Color 500	339.00
Color 600	399.00
Color 700	469.00
Color 710	539.00
310A	145.00
ATARI	
XC 141	199.95
SM 124	135.95
SC 1224	335.95
TEKNIKA	
MJ-10	**175.95**
MJ-22 RGB	**249.95**
SAKATA	
SC-100	179.00
COMMODORE	
C-1702	185.95
C-1902 RGB	259.95
C-1901	129.95

COMPUTERS

COMMODORE	
C-64	139.95
C-128	275.95
ATARI	
800 XL	89.95
130 XE	139.95
520 ST Monochrome & More	CALL
520 ST Color RGB & More	CALL

MUST LIQUIDATE At Far Below Dealer Cost!
TOTAL Personal Computer System

Factory Reconditioned with Factory Warranty!

Carries easily as a suitcase! Plugs into 115V outlet

GREAT GIFT IDEA!

Sorry, we're not permitted to PRINT the famous brand-name. BUT, we CAN "tell all" if you call us TOLL FREE: 1-800-328-0609!

THE COMPUTER

Snap-on computer keyboard! 64K RAM, 20K ROM. Full-size typewriter keyboard. Upper and lower case letters, numerals, symbols, reverse characters. 2 cursor control keys, 4 function keys, programmable to 8. Music synthesizer with 3 independent voices, each with 9 octave range. Input/output ports accommodate . . . user, serial, ROM cartridge, joysticks, external monitor, phone modem.

Built-in disk drive! Intelligent high speed unit with 5¼" floppy disk recorder. 170K formatted data storage; 35 tracks. 16K ROM. Uses single sided, single density disk. Serial interface. Second serial port to chain second drive or printer.

Built-in color monitor! Displays 40 columns x 25 lines of text on 5" screen. High resolution. 320 x 200 pixels. 16 background, character colors.

Built-in ROM cartridge port! Insert ROM program cartridge. Multitude of subjects available in stores across the nation!

Original List Price $995.00

Liquidation Priced At Only **$388**

Item H-764-63631-00 Ship, handling: $20.00

THE PRINTER

Print method: Bi-directional impact dot matrix.
Character matrix: 6 x 7 dot matrix.
Characters: Upper and lower case letters, numerals and symbols. All PET graphic characters.
Graphics: 7 vertical dots — maximum 480 columns. Dot addressable.
Character codes: CBM ASCII code.
Print speed: 60 characters per second.
Maximum columns: 80 columns.
Character spacing: 10 characters per inch.
Line feed spacing: 6 lines per inch in character mode or 8 lines per inch selectable. 9 lines per inch in graphics mode.
Line feed speed: 5 lines per second in character mode. 7.5 lines per second in graphics mode.
Paper feed: Friction feed.
Paper width: 4.5" to 8.5" width.
Multiple copies: Original plus maximum of two copies.
Dimensions: 13"W x 8"D x 3¼"H. Wt.: 6½ lbs. **Power:** 120V AC, 60 Hz.

Original List Price: $200.00

Liquidation Priced At **$118**

Item H-764-63681-00 Ship, handling: $7.00

THE SOFTWARE

"Easy Script" One of the most powerful word processors at any price! Cut re-typing, create documents from standard paragraphs, do personalized letters, see and change a document before it is printed. Instruction manual has extensive training section that simplifies use . . . even for someone who has never used a computer or word processor before!

"The Manager" A sophisticated database manager for business or home use. Business uses: accounts payable/receivable, inventory, appointments, task manager. Home uses: mailing lists, home inventory, recipes, collection organizer, investment tracking, checkbook balancing. School uses: research article index, gradebook.

Original List Price: $73.98

Liquidation Price **$24**

Item H-764-64011-03 Ship, handling: $3.00

BUY INDIVIDUAL UNITS OR GET THIS ULTRA-FAMOUS SYSTEM AT ONE LOW PACKAGE PRICE!

TOTAL Personal Computer System available at FAR BELOW dealer cost!

Original List Price $1,293.00

TOTAL PACKAGE PRICE **$488**

Item H-764-64011-02 Ship, handling: $24.00

Compatible with above Computer System (Not included in package price.)

JOYSTICKS (Set of 2)
Mfr. List: $59.90 pr.
Liquidation Price . . . . . **$18** pr.
Item H-764-63622-01 S/H: $6.00 pr.

64K MODEM (Factory Reconditioned w/Warranty)
Mfr. List: $124.95
Liquidation Price **$19**
Item H-764-63646-01 S/H: $4.00

VISA MasterCard

Credit card members can order by phone, 24 hours a day, 7 days a week.

Toll-Free: 1-800-328-0609

TeleCheck Your check is welcome! No delays when you pay by check!

C.O.M.B. CO.®
Authorized Liquidator
14605 28TH AVENUE NORTH
MINNEAPOLIS, MINNESOTA 55441-3397

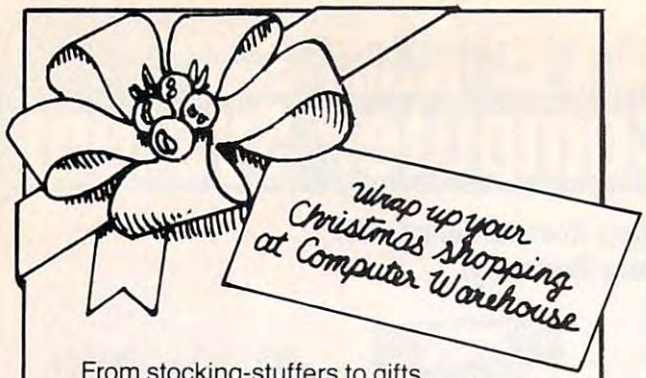
SEND ME THE ITEMS I HAVE LISTED BELOW
Price subject to change after 60 days. Sales outside continental U.S. are subject to special conditions. Please call or write to inquire.

Item No.	#	Item	Price	S/H
		TOTAL		

C.O.M.B. CO.® Item H-764
14605 28th Ave. N./Minneapolis, MN 55441-3397

Send the items indicated at left. (Minnesota residents add 6% sales tax. Please allow 3-4 weeks delivery. Sorry, no C.O.D.)
☐ My check or money order is enclosed. (No delays in processing orders paid by check, thanks to TeleCheck.)
Charge: ☐ MasterCard* ☐ VISA

Acct. No. _____ Exp. _____
PLEASE PRINT CLEARLY
Name _____
Address _____
City _____
State _____ ZIP _____
Phone _____
Sign here _____



From stocking-stuffers to gifts that'll light up their eyes, Computer Warehouse is *the* place to shop. You'll know you're getting the lowest prices: just see the list below and compare!

And even Santa couldn't get your order out faster! So call our hotline today, and wrap up your gift list at Computer Warehouse.

PRINTERS

Star SG-10	224.00
SG-10C	239.00
Panasonic 1091	259.00
Axiom Elite 5 (daisy wheel)	249.00
Okidata 182	269.00
Legend 808	179.00

HARDWARE

WYSE PC	
100% IBM Compatible	
256K	
2-360 K Drives	
3-I/O Ports	
2-Exp Slots	$1,550
Commodore 64	139.00
1541 Disk Drive	164.00
1702 Monitor	169.00
803 Printers	139.00
803 Tractor Feed	29.95
Modem 300/1660	79.95
Power Packs	29.95
1101 Printer (daisy wheel)	329.00
C-128	Call
1571 Disk Drive	Call
1670 Modem	Call
1902 Monitor	Call
Amiga	Call
1010 Recorder	39.95
1050 Disk Drives	165.00
Atari 800XL	89.00
Atari 130XE	149.00

MONITORS

Sakata 13" Color	179.00
Zenith 13" Green	89.00
Zenith 13" Amber	99.00
USI 12" Green	79.00

MODEMS

Westridge 6420	69.95
MPP 1064	79.95
Total Telecommunications	39.95
Mighty Mo	79.95

DISK DRIVES

Indus GT	249.95
MSD Single	289.95
MSD Dual	499.95

SOFTWARE

Commodore	
Printshop	34.95
Graphics Library	24.95
FastLoad	29.95
Fleet System 2	69.95
Zork I, II, III	29.95
Training Kit (C-64 Tutorial)	19.95
Wordpro 3 & 64	39.95
Exodus Ultima III	39.95
Rescue on Fractolus	29.95
Atari	
Atariwriter	24.95
Timewise	24.95
Visicalc	24.95
SynFile	49.95
SynStock	49.95
SynTrend	49.95
Paperclip	54.95

DISKETTES (10-PACKS)

Generic Disks DS/DD	10.00
SKC SS/SD	13.95
SKC SS/DD	15.95
Elephant SS/SD	16.95
Elephant SS/DD	19.95
Elephant DS/DD	24.95
Elephant Premium SS/DD	21.95
Maxell SS/DD	22.95
Maxell DS/DD	29.95
Bonus SS/DD (11-Pack)	12.95

ACCESSORIES

Floppiclene Head Cleaner	14.95
Numeric Keypad	34.95
Monitor Cables	9.00
6 foot I/O Cable	9.00
Paper—(1000 Sheets)	18.00
Joysticks	7.95
Surge Suppressor 6 outlet w/ 6 ft. cord Emz/Rfi filter	49.95
Diskcover 35 (Storage Box)	5.95
Printer Stand	19.95

* We also carry a full line of Hardware, Software, Covers, Printer Ribbons, Cables, Labels, Storage Cases for most popular computers. Call for new titles & prices.

COMPUTER WAREHOUSE

(In FL) 305-274-3680; 1-800-372-0214
7222 S.W. 117th Avenue,
Miami, FL 33183

Add 3% ($3 minimum) for shipping and handling charges. FPO's & APO's and overseas subject to additional shipping charges. Ad prices reflect 3% cash discount. Credit card orders should add 3%. Prices and availability subject to change without notice.

IT'S LIKE FREE DISKETTES

Your 5¼" single side disks are usable on the other side. You paid for one side, why not use the other... **IT'S FREE!**

Nibble Notch will **open** your **new** disk. It's easy... won't harm existing data. Try it!

nibble notch I
For Apple, Franklin, Commodore & Atari (w/Atari Drives); square notch.
only $14.95* PLUS P&H

nibble notch II
For all other computers; square notch & index hole.
only $21.90* PLUS P&H

DISK OPTIMIZER II©
Apple II Series Software
Pro DOS • DOS 3.3 • Pascal
Examines your **new** disk, locks out bad sectors and **certifies it 100% ERROR-FREE** in 30 seconds or less! Also checks drive speed... and more!

SUPER SAVER PACKAGE
Nibble Notch I and Disk Optimizer Combo (Optimizer alone reg. $24.95)
$29.95* FOR BOTH!

QUALITY DISKETTES low as 99¢

*add $2 ($5 frgn) for P & H. Fl. Res. add 5% Sales Tax

nibble notch computer products

Toll Free 1-800-642-2536
FL 1-305-748-3770
OR SEND CHECK OR MONEY ORDER TO:

VISA MasterCard

4211 NW 75th TERRACE, • DEPT. 662 LAUDERHILL, FL 33319

PROFESSIONAL HANDICAPPING SYSTEMS

• PRESENTED BY PROFESSOR JONES •

GLD. Thoroughbred "Gold" Edition™
A "Full" featured thoroughbred analysis designed for the professional and the serious novice. **$159.95** complete

EGLD. Enhanced "Gold" Edition™
"Gold" Edition with complete Master Bettor™ system integrated onto the same disk. This powerful program will transfer all horses and scores to the bet analysis with a "single keystroke." (Master Bettor™ included) **$199.95** complete

GLTD. Limited "Gold"™
Enables **Professional Handicappers** to assign specific values to the racing variables "they" feel are important. Create program weight based on a particular track and fine tune it for **maximum** win percentage. This program is designed for "ease of use". The user needs **no programming experience.**
(contains Integrated Bettor™) **$299.95** complete

GD. Gold Dog Analysis™
The "ONLY" professional greyhound analysis available that evaluates ALL variables. **$149.95** w/integrated Bettor **$199.95**
Limited Version **$299.95**

MHH. Master Harness Handicapper™
Professional software designed to provide a thorough analysis of all trotter and pacer races in North America and Canada. **$159.95** complete
$199.95 w/Integrated Bettor Limited **$299.95**

Professor Pix Football™
Complete STATISTICAL ANALYSIS on Data Base allowing "Designated" previous games to be evaluated. Statistical Series **$99.95**
w/Win-Loss Power Ratings **$149.95**

NBA. Basketball™ NBA **$99.95** w/college **$129.95**
w/power ratings **$149.95**

LOT. Lottery Analysis™
Statistical comparison program designed to detect subtle patterns in winning numbers and digits. Lottery (3-4 digit) **$79.95**
w/Lotto (Max. 99 Digit) **$99.95**

PC-3 Portable Computer (4k) with choice of Thoroughbred, Greyhound or Trotter™ **$249.95** (Includes portable computer and program.)

M-100 Portable (24k) w/choice of Thoroughbred, Greyhound, or Trotter. **$649.95** (Includes portable computer and program.)

BROCHURE AVAILABLE

IBM™
APPLE™
TRS-80™
CPM™
COMMODORE™

Prof. Jones
1940 W. State St.
Boise, ID 83702

VISA MasterCard

48 HR. FREE SHIPPING

CALL 208-342-6939

TERMS: FREE SHIPPING ALL SOFTWARE. Add $6.00 hardware / $6.00 C.O.D. / UPS Blue $6.00 / Out of Country $9.00 / ID Residents 4% / 3 weeks personal checks / Cash price only, add 2% Visa, MC / Prices subject to change.

Finally, something Apple and IBM owners can agree on:

The Sider™ 10 MB hard disk

Only $595 Thru December 31, 1985

from First Class Peripherals

Decisions, decisions. First you had to choose between Apple and IBM. Now you have to decide which hard disk subsystem to purchase—and they all seem about the same. *But are they?*

First Class Peripherals can make your hard disk decision a lot easier. Because whether you use an Apple II+ or IIe...or IBM PC* or XT...we offer a Sider 10 MB hard disk subsystem just right for all your storage needs.

The most reliable, affordable 10 MB hard disk on the market

The Sider features state-of-the-art Winchester disk technology. Direct booting without floppies. Self-contained power supply. And compatibility with the most popular Apple or IBM software.

In addition, the Sider is *plug and play.* Everything you need for quick, easy installation is included: cable, host adapter, software and manual.

Built to last by Xebec

The Sider has won rave reviews for its performance and reliability. That's because it's manufactured exclusively for First Class Peripherals by Xebec, the industry's leading manufacturer of computer disk drives and controllers. And it's sold *direct to you,* so there are no dealers or distributors to hike up the cost.

Full guarantee and free tech hotline

You can choose your Apple or IBM Sider with confidence. Simply order and use your Sider for 15 days. If you're not 100% satisfied, return it for a full refund. Keep it, and you'll enjoy a full one-year limited warranty...plus access to our toll-free hotline, should you ever have a technical or service question.

It's easy to order your Sider

The Sider is priced at just $595 for the Apple model...$595 for the IBM. *That's hundreds of dollars less than what you'd expect to pay for the comparable "big name" models.* To order, use the coupon below...or for faster service, order by phone using Visa, MasterCard or American Express. Call toll-free:

1 800 538-1307

Extension 702

*Please send me the Sider, including 10 megabyte hard disk drive, host adapter card, cable, complete installation software and documentation for my: ☐ **Apple II+ or IIe** ☐ **IBM PC or XT***

I prefer to pay as follows:

☐ I've enclosed my check or money order for $595* plus $15 shipping and handling, payable to First Class Peripherals.

☐ Please bill the following credit card account for $595* plus $15 shipping and handling:

☐ VISA ☐ MasterCard ☐ American Express

Card # *Exp. Date*

Signature

*Residents of CA, NV and PA, please add appropriate sales tax.

Name

Address

City

State *Zip*

Telephone (area code)

Mail to: First Class Peripherals
3579 Highway 50 East, Carson City, NV 89701

702

*Must contain hard disk ROM.

FIRST PERIPHERALS CLASS

3579 Highway 50 East, Carson City, NV 89701

Advertisers Index

This holiday season put your fellow PC users on line with

COMPUTE!

The magazine for Home, Education and Recreational Computing.

SPECIAL HOLIDAY GIFT SAVINGS

For each gift you give, save 25% off the regular $24 subscription rate. You pay only $18 per gift subscription.

A SEASONAL CARD WILL ANNOUNCE EACH GIFT

YOUR NAME _____ ☐ PAYMENT ENCLOSED ☐ PLEASE BILL ME

ADDRESS _____ APT # _____

CITY/STATE/ZIP _____

1. TO: _____

ADDRESS _____ APT # _____

CITY/STATE/ZIP _____

GIFT CARD TO READ _____

2. TO: _____

ADDRESS _____ APT # _____

CITY/STATE/ZIP _____

GIFT CARD TO READ _____

PRICES GOOD IN USA ONLY. FOR CANADA AND FOREIGN PLEASE ADD $6 PER SUBSCRIPTION. M1987

N

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 7478 DES MOINES, IOWA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPUTE!

PO BOX 10954
DES MOINES, IOWA 50347

COMPUTE!

Holiday Savings...
25% off the regular rate!

The magazine for Personal, Home, Educational and Recreational Computing.

SPECIAL SAVINGS

☐ Send me 1 year (12 issues) for just $18—25% off the regular $24 subscription rate.

☐ Bill me ☐ Payment enclosed

NAME _____

ADDRESS _____

CITY/STATE/ZIP _____

Please indicate make and model of computer you use:
APPLE ☐ 01 ATARI ☐ 02 64 ☐ 03 VIC 20 ☐ 04 IBM ☐ 05
OTHER _____ 99

FOR NEW SUBSCRIBERS ONLY

PRICE GOOD ONLY IN USA. FOR CANADA AND FOREIGN PLEASE ADD $6. J1072

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 7478 DES MOINES, IOWA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPUTE!

PO BOX 10954
DES MOINES, IOWA 50347

COMPUTE!'s FREE Reader Information Service

Use these cards to request FREE information about the products advertised in this issue. Clearly print or type your full name and address. Only one card should be used per person. Circle the numbers that correspond to the key number appearing in the advertisers index.

Send in the card and the advertisers will receive your inquiry. Although every effort is made to insure that only advertisers wishing to provide product information have reader service numbers, COMPUTE! cannot be responsible if advertisers do not provide literature to readers.

Please use these cards *only* for subscribing or for requesting product information. Editorial and customer service inquiries should be addressed to: COMPUTE!, P.O. Box 5406, Greensboro, NC 27403. Check the expiration date on the card to insure proper handling.

Use these cards and this address only for COMPUTE!'s Reader Information Service. Do not send with payment in any form.

COMPUTE!

101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117
118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134
135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151
152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168
169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185
186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202
203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219
220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236
237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253

Circle 101 for a one year new U.S. subscription to COMPUTE!: you will be billed for $24.

Please let us know. Do you own: plan to buy:

- ☐ 270 Apple _____ ☐ 271
- ☐ 272 Atari _____ ☐ 273
- ☐ 274 Commodore _____ ☐ 275
- ☐ 276 IBM _____ ☐ 277
- ☐ 278 TI-99/4A _____ ☐ 279
- ☐ 280 Other _____ (specify model) ☐ 281

Please print or type name and address.
Limit one card per person.

Name _____

Address _____

City _____

State/Province _____ Zip _____

Country _____

Please include zip code. Expiration date 1/31/86 C01285

Place
Stamp
Here

COMPUTE! Reader Service
P.O. Box 2141
Radnor, PA 19089

S'MORE
FOR YOUR 64

61,183 delicious bytes for your Commodore 64

S'MORE Memory, S'MORE Power, S'MORE Fun!

Now, 61K available memory
S'MORE frees up 61,183 bytes of C-64 RAM memory for un-restricted Basic programming (57% more than the standard C-64).

A bridge to C-128
The S'MORE command structure is similar to the C-128's new Basic 7.0, providing C-64 users advanced programming techniques.

S'MORE features for programming power

- Over 60 new and enhanced basic commands & functions
- No peeks or pokes (direct access to normally peeked/poked items)
- Full error trapping and automatic error helps
- Full up/down scrolling through program listings
- Structured programming
- Relative files
- Print using
- Formatted inputs
- Print at...and much, much more.

cardco, inc.

The Wizards from the Land of Oz Have done it Again!

CARDCO, Inc./300 S. Topeka/Wichita, KS 67202

HOW TO EVOLVE TO A HIGHER INTELLIGENCE.

1-800-

8890700

THE COMMODORE 128.

The first step is buying the Commodore 128™ Personal Computer. The smartest computer available for the price. It's like getting three computers for less than one usually costs. You can run CP/M® business software, the new programs written for the 128, and over 3,000 Commodore 64® programs. You start out with more software than most machines give you after years on the market.

THE COMMODORE 128 WORKS FASTER.

To run all that software and run it faster, you'll want the 1571 Disk Drive. You can't find a faster drive at the price. It transfers nearly 1,000 words a second (5200 cps), so you can load most programs instantly.

THE COMMODORE 128 GETS SMARTER.

Now try improving your memory. Plug in our 1750 RAM Expansion Module and your 128 moves up to a powerful 512K. That's enough to handle just about anything you can dish out, from complicated business forecasting to giant data bases.

THE COMMODORE 128 LEARNS TO COMMUNICATE.

There's no real intelligence without the ability to communicate. So you'll want our 1670 Modem/1200. It puts you in touch with a new world of shopping, banking, communications and information over your telephone line. And it operates at a lightning-fast 1200 baud to save on your phone bill.

THE COMMODORE 128 LEARNS TO WRITE.

Looking good in print could be your next move with the MPS 1000 Printer. It's a new dot matrix printer designed to make the most of the 128's speed and high-resolution graphics. The MPS turns out about 1200 words a minute (100 cps) of draft-quality printing, or gives you near-letter-quality at about 240 words a minute (20 cps).

THE COMMODORE 128 IMPROVES YOUR VISION.

Brains aren't enough without good looks, so improve your vision with Commodore's new 1902 RGB Color Monitor. The high-resolution screen gives you a sharper image and better color than your standard TV, so you can really appreciate the 128's great graphics.

All these evolutionary steps ahead won't set you back when it comes to paying for them. Additions to your Commodore 128 are available at a store near you and are as affordable as the 128 itself. We think that's a smart way to help you build a computer system.

COMMODORE 128 PERSONAL COMPUTER

A Higher Intelligence