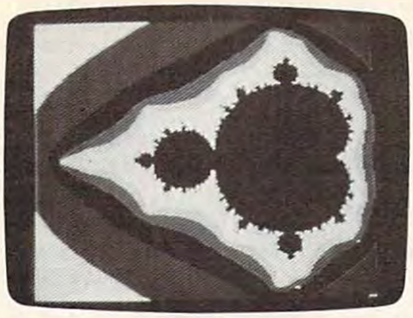
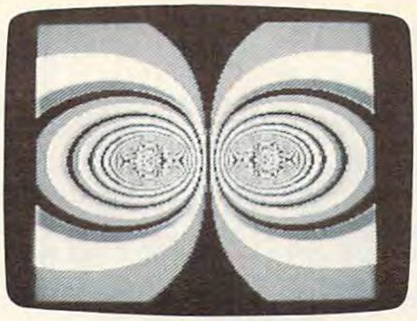


"Mandelbrot Graphics For Commodore" creates detailed hi-res images based on an unusual mathematics technique.



This image simulates the lines of electromagnetic force surrounding two electrical conductors.

This pattern is created by a program that represents the gravitational forces around the earth and the moon.

program). When that's done, save the program using a different filename from the one you used for Program 1.

Program 2 uses the color schedule to paint patterns that resemble the magnetic lines of force between two wires charged with electricity. Displays of this type make it easier to visualize forces such as electromagnetism, which are invisible to the naked eye, yet have a profound effect in the world around us. Of course, you can enjoy the displays even if you don't understand the scientific concepts involved in these simulations.

You can also modify Program 2 to produce different effects. Change the values of variables I1 and I2 to see what the magnetic pattern is like for various current levels. For instance, if you set I1 to 25 and I2 to −100, the program displays an opposed current flow with one current four times as large as the other. Larger current levels cause closer flux lines, more detail, and a greater chance of color distortion.

To enter Program 3, load Program 1, type the lines listed for Program 3, then save the entire program under a unique filename. Program 3 simulates another invisible, yet powerful force of nature—the interaction of gravitational forces between the earth and the moon. The black region on the right is the zone of gravity equal to that at the surface of the moon. The program accurately draws the moon to scale relative to the gravitational potentials in effect.

Line 120 of Program 3 positions the earth and moon relative to the upper left-hand corner of the screen, using an *x,y* coordinate scheme in which the screen has horizontal positions 0–159 and vertical positions 0–199. The earth can be shown by assigning it a value on the screen. You can create an interesting image by placing the earth at coordinates 53,99 and the moon at coordinates 106,99.

The variable F determines which color is plotted. F represents the gravitational force (measured in Newtons) on a stationary one-kilogram mass. Variable DE represents the distance from earth, and DM signifies the distance from the moon. The region close to the moon was made one color because it has such a steep gravitational gradient that there would be excessive color distortion if it were not specially treated.

Saving And Loading Screens

After you create a screen, it's easy to save it on disk for future use. Simply press S after the image is completely drawn. Before saving an image, make sure that the disk drive contains a disk with at least 40 blocks free. Three different memory areas must be saved in separate disk files to reconstruct the image accurately.

To avoid spoiling the image with a printed prompt, the program switches temporarily to a text screen to allow you to enter a filename (which should contain no more than 12 characters). The program then manipulates the name you enter to create distinct filenames for the three files necessary to store all the picture information. The prefix *M–* is added to the filename you specified for the first file. This file contains the hi-res bitmap of the image. For the next file, the suffix *S* is added in addition to the *M–* prefix. This file contains the screen memory for the image. Finally, for the third file, the suffix *SC* is added in addition to the *M–* prefix. This file contains the color memory for the image.

Choose your filenames carefully; the program erases any existing files with the specified filename before it stores the new files. Thus, if you accidentally reuse an existing name, you may overwrite a previously saved masterpiece.

After you enter the filename, the program switches back to the hi-res image while the screen is being saved.

Program 4 loads the stored image from disk and displays it on the screen. When you run this program, it asks you to enter a filename. Give the filename you specified when you saved the image with Programs 1–3. Don't worry about the prefixes and suffixes; Program 4 takes care of these and loads all three files necessary to recreate the image.

What Is A Mandelbrot Set?

Although Mandelbrot images have attracted wide interest as a means of generating graphics, the origin of these pictures lies deep in the realms of mathematics and engineering. In brief, certain engineering problems require the use of *complex numbers.* A complex number consists of two parts: a *real* part and an *imaginary* part. The real part of the number is like the numbers we use every day. For the imaginary part, special rules of multiplication apply. For instance, if you square the imaginary part of a complex number, the result becomes negative and real. Thus, imaginary numbers are often written in engineering calculations as a real number multiplied by the constant *i*, where *i* can be considered the square root of −1.

The real and imaginary parts of a complex number can be plotted on a coordinate grid known as the *complex plane*. Benoit B. Mandelbrot, a researcher at IBM, discovered that points inside a certain region of the complex plane behave strangely when they are repeatedly squared and the result of each squaring is added to the original point. Some points get large rapidly, while their near neighbors grow slowly. Other neighboring points don't grow at all. As the process is repeated, these tendencies are accentuated. The region where these strange results occur is called the *Mandelbrot set* after its discoverer.

While Mandelbrot sets have certain practical applications, they can produce striking results when you translate the numeric values into different-colored points on a computer screen. These programs color each point on the hi-res screen according to how fast it grows during the iterative (repetitive) process. Computing the entire set requires an enormous number of calculations, so all of these programs take some time to complete—many hours in some cases. You may want to start a program running at bedtime so the picture will be finished by the next morning. (There's no need to leave the monitor on all night, of course.)

If you'd like to photograph a finished image, shoot the photos in a darkened room to eliminate screen glare. Load a single-lens reflex camera with medium-speed film (ISO 100 works well), mount the camera on a tripod, attach a cable release, aim the viewfinder squarely at the screen, and use a very slow shutter speed—no faster than 1/4 second. If your camera has an automatic exposure mode or a built-in light meter, it should indicate a lens aperture of f/8 to f/16.

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

Program 1: Mandelbrot Graphics

```
QR 10 POKE56,32:POKE55,0:CLR:F
      ORA=828TO908:READB:POKEA
      ,B:NEXT
RR 20 DATA 162,2,160,4,208,4,1
      62,4,160,2,142,101,3,142
      ,127,3,140,103,3
GD 30 DATA 140,129,3,169,0,133
      ,2,133,4,169,216,133,3,1
      69,200,133,5,162,3
AE 40 DATA 160,0,177,2,145,4,1
      36,208,249,230,3,230,5,2
      02,16,242,169,4,133
JF 50 DATA 3,169,204,133,5,162
      ,3,160,0,177,2,145,4,136
      ,208,249,230,3,230
EK 60 DATA 5,202,16,242,96
AB 100 REM ------ MANDELZOOM-6
       4 ------
PH 120 SM=7:CT=40:REM DETERMIN
       ES DETAIL
CE 125 REM DEFINE REGION EXAMI
       NED BELOW
GK 130 XL=-2.2:XR=.58:YB=-1.25
BP 137 DX=(XR-XL):YT=YB+DX*.9
MS 140 DIMCO(15):FORI=0TO15:RE
       ADCO(I):NEXT
AC 150 GOSUB360:M=159/DX:B1=-M
       *XL
SJ 160 MM=199/(YB-YT):BB=-MM*Y
       T
MK 170 DATA 1,3,4,2,8,7,13,5,1
       4,6,4,15,14,6,4,15:REM
       {SPACE}<-- COLOR CODE S
       CHEDULE
DS 180 FOR X0=XL TO XR STEP(XR
       -XL)/159
PQ 190 FOR{2 SPACES}Y=YT TO YB
        STEP(YB-YT)/199
EF 200 A=X0*X0-Y*Y+X0:B=2*X0*Y
       +Y:C=0
RJ 210 R=A*A-B*B+X0:I=2*A*B+Y:
       C=C+1:S=USR(R*R+I*I):A=
       R:B=I:IFS<SMTHENIFC<CTT
       HEN210
QR 220 X1=INT(M*X0+B1+.5)
RQ 222 Y1=INT(MM*Y+BB+.5):IFC<
       6THENCO=CO((C/15-INT(C/
       15))*15):GOTO240
QF 225 IFC<CTTHENCO=1:GOTO240
XK 230 IFC=CTTHENCO=0
SB 240 GOSUB440:NEXT Y,X0
DP 250 REM SAVE PICTURE IF 'S'
        IS PRESSED
KX 260 GETA$:IFA$<>"S"THEN260
FB 261 SYS828:POKE53272,21:POK
       E53265,27:POKE53270,200
SF 262 PRINT"{CLR}{2 DOWN}ENTE
       R FILENAME (UP TO 12 CH
       AR)":INPUTF$:F$="M-"+LE
       FT$(F$,12)
SQ 270 POKE53272,29:POKE53265,
       59:POKE53270,216:SYS834
       :K=8192:E=16191:GOSUB30
       0
EM 280 F$=T$+"S":K=1023:E=2023
       :GOSUB300
XH 290 F$=T$+"C":K=55296:E=562
       95:GOSUB300:GOTO260
EM 300 F$="0:"+F$:OPEN15,8,15,
       "S"+F$:CLOSE15:T$=F$
SJ 310 ZK=PEEK(53)+256*PEEK(54
       )-LEN(T$):POKE782,ZK/25
       6:POKE781,ZK-PEEK(782)*
       256
HM 320 POKE780,LEN(T$):SYS6546
       9:POKE780,1:POKE781,8:P
       OKE782,1:SYS65466
EE 330 POKE254,K/256:POKE253,K
       -PEEK(254)*256:POKE780,
       253:K=E+1:POKE782,K/256
SD 340 POKE781,K-PEEK(782)*256
       :SYS65496:T$=RIGHT$(T$,
       LEN(T$)-2):RETURN
BX 350 REM----- SET COLOR GRAP
       HICS MODE AND CLEAR MEM
       ORY TO BE USED -----
XF 360 PRINT"{CLR}":FORI=49152
       TO49248:READJ:POKEI,J:N
       EXT:POKE53280,0:POKE532
       81,0
CJ 370 POKE251,0:POKE252,32:PO
       KE253,63:POKE254,63:POK
       E49169,0:SYS49166
AK 380 SYS49152:POKE785,39:POK
       E786,192
XD 390 BASE=8192:POKE53272,PEE
       K(53272)OR8
GB 400 POKE53265,PEEK(53265)OR
       32:POKE53270,PEEK(53270
       )OR16
HM 410 POKE251,0:POKE252,216:P
       OKE253,231:POKE254,219:
       SYS49166:RETURN
RQ 420 REM ---- PLOT X1,Y1,CO
       {SPACE}-------------
AH 430 REM{2 SPACES}0<=X1<160
       {2 SPACES}0<=Y1<200 0<=
       CO<15
AJ 440 IF CO=0 THEN RETURN:REM
        USE SCREEN COLOR CODE
       {SPACE}IN 53281
EJ 450 X=2*X1+1:RO=INT(Y1/8):C
       H=INT(X/8):LN=Y1AND7:BY
       =BA+RO*320+8*CH+LN
SJ 460 REM --- SET COLOR AND P
       IXEL ---
RG 470 SB=1024+RO*40+CH:REM SC
       REEN BYTE
EF 480 SE=PEEK(SB):S9=SEAND240
JK 490 REM USE HI NYBBLE OF SB
        IF OK
KC 500 IF(S9=0ORS9=CO*16)THEN
       {SPACE}POKE SB,SEOR(CO*
       16):GOTO580
GF 510 REM USE LO NYBBLE OF SB
        IF OK
XR 520 S8=SEAND15:IF(S8=0ORS8=
       CO)THEN POKE SB,SEORCO:
       X=X-1:GOTO580
SG 530 REM USE COLOR MEM. IF I
       T IS OK
KJ 540 CM=SB+54272:REM COLOR N
       YBBLE ADDR.
PD 550 C1=PEEK(CM):C3=C1AND15
GF 560 IF(C3=0ORC3=CO)THENPOKE
       CM,C1 ORCO:GOSUB580:X=X
       -1
EB 570 REM TOO MANY COLORS THI
       S CHAR.
AE 580 BI=7-(XAND7):POKEBY,PEE
       K(BY)OR(2↑BI):RETURN
HD 590 DATA 169,0,133,251,133,
       253,169,4,133,252,169,8
       ,133,254,160,0,169,27,1
       45,251
ES 600 DATA 230,251,208,2,230,
       252,165,251,197,253,208
       ,240,165,252,197,254,20
       8,234
DH 610 DATA 96,32,43,188,240,5
       2,16,3,76,72,178,32,199
       ,187,165,97,56,233,129,
       8,74
DF 620 DATA 24,105,1,40,144,2,
       105,127,133,97,169,4,13
       3,103,32,202,187,169,92
       ,160,0
FJ 630 DATA 32,15,187,169,87,1
       60,0,32,103,184,198,97,
       198,103,208,233,96
```

Program 2: Magnetic Forces

```
HX 100 REM------- MAGNETIC FIE
       LD -------
CR 120 DIMCO(15):FORI=0TO15:RE
       ADCO(I):NEXT
```

```
SG 130 REM WIRE LOCATIONS, CUR
       RENT LEVELS & PHYSICAL
       {SPACE}CONSTANTS
MM 140 XA=53:YA=99:XB=107:YB=9
       9:I1=100:I2=-100:K=2E-7
       :GOSUB360
GR 150 DATA 0,0,0,6,6,6,6,5,5,
       5,5,5,4,4,4,4:REM <----
        COLOR CODE SCHEDULE
DC 160 FOR X0=0 TO 159 : FOR Y
       =100 TO 199
PG 170 DA=USR((XA-X0)↑2+(YA-Y)
       ↑2)
SF 180 DB=USR((XB-X0)↑2+(YB-Y)
       ↑2)
FD 190 B=K*I1/DA+K*I2/DB
QJ 200 X1=X0:Y1=Y:CO=ABS(INT(B
       *1E8+.5)):CO=CO((CO/15-
       INT(CO/15))*15):GOSUB42
       0
GP 210 Y1=199-Y:GOSUB440:NEXT
       {SPACE}Y,X0
HH 220 REM
PH 230 REM
XJ 240 REM
```

Program 3: Forces Of Gravity

```
SR 5 REM----- EARTH-MOON SYSTE
     M-----
PD 100 DIMCO(15):FORI=0TO15:RE
       ADCO(I):NEXT
RR 110 REM BODY LOCATIONS & MA
       SSES
GC 120 XE=-111:YE=99:XM=80:YM=
       99:EM=6E24:MM=7.3E22:G=
       .667E-10:GOSUB360:XL=XE
       :YB=YM
KM 130 CF=3.8E8/(XM-XE)
EA 140 DATA14,7,11,2,9,8,7,13,
       5,3,14,6,4,2,9,8:REM <-
       --- COLOR CODE SCHEDULE
BB 150 REM CF CONVERTS SCREEN
       {SPACE}UNITS TO METERS
       {SPACE}AND 1.48 CORRECT
       S X-Y ASPECT
RX 160 FORX0=0TO159:FORY=0TO99
EX 170 D1=XE-X0:D2=(YE-Y)/1.48
       :D3=XM-X0:D4=(YM-Y)/1.4
       8
FM 180 DE=USR(D1*D1+D2*D2)*CF+
       1:DM=USR(D3*D3+D4*D4)*C
       F+1
XD 190 FE=G*EM/DE/DE:FM=G*MM/D
       M/DM:FX=FE*COS(D1/DE)+F
       M*COS(D3/DM)
KQ 200 FY=FE*SIN(D2/DE)+FM*SIN
       (D4/DM)
FM 210 F=USR(FX*FX+FY*FY)
QH 220 X1=X0:Y1=Y:CO=INT(F*5E3
       +.5):CO=CO(ABS((CO/15-I
       NT(CO/15))*15))
XH 222 REM
DJ 225 REM
SF 230 IFF>.0055THENIFX0>40THE
       NCO=4:IFF>.167THENNEXTX
       0
FH 240 GOSUB450:Y1=199-Y:GOSUB
       450:NEXT Y,X0
```

Program 4: Image Loader

```
PS 100 REM---MULTI-COLOR BIT M
       AP RECALL---
HF 110 INPUT"{CLR}{3 DOWN}ENTE
       R FILENAME";F$:F$="M-"+
       F$
PG 120 PRINT"{CLR}":POKE53272,
       (PEEK(53272)AND240)OR8:
       POKE53265,PEEK(53265)OR
       32
CR 130 POKE53270,PEEK(53270)OR
       16:POKE53280,0:POKE5328
       1,0
SR 140 F$="0:"+F$:GOSUB180
AR 150 F$=F$+"S":GOSUB180
EH 160 F$=F$+"C":GOSUB180
FX 170 GOTO170
AM 180 T$=F$:ZK=PEEK(53)+256*P
       EEK(54)-LEN(T$):POKE782
       ,ZK/256
MJ 190 POKE781,ZK-PEEK(782)*25
       6:POKE780,LEN(T$):SYS65
       469
EF 200 POKE780,1:POKE781,8:POK
       E782,1:SYS65466
SQ 210 POKE780,0:SYS65493:RETU
       RN
```

©

Fred D'Ignazio, Associate Editor

It Only Takes Two To Make Music

For 20 years Paul Lehrman has dreamt about becoming a composer. "But," he says, "serious composers often end up starving to death."

Paul has a lot of experience in the music industry—as a musician, recording engineer, and most recently as vice president of Southworth Systems, the makers of *Total Music*, a MIDI music program for the Apple Macintosh. "I'm not a serious composer," he says. "I'm a pop guy. But I want to help serious composers make music."

Paul is convinced that computers can help composers. To prove this, he recently did what no one has done before: He created an entire album of music with a personal computer. His album, *The Celtic Macintosh*, consists of traditional and contemporary Irish and Scottish jigs, reels, hornpipes, airs, and laments. It took him less than three weeks, start to finish, and cost him a little under $100. (He had to buy five floppy disks, a VHS videotape, and rent a digital tape converter.)

Why Irish and Scottish folk songs? St. Patrick's Day was coming, and Paul's Irish-American friend Sharon Kennedy, a storyteller, asked him to put together "something quick" that she could play over the PA system as background music at the annual St. Patrick's Day Concert in Brookline, Massachusetts. She was so happy with the result that Paul decided to make a whole album and sell it at the concert.

A Home Studio

Paul's recording studio was in his living room. Each morning he would get up at 10:00 a.m., shower, eat breakfast, walk the eight steps to his studio, and go to work—in jeans and a bathrobe. His instruments consisted of three synthesizers (a Kurzweil K250, a Yamaha DX7, and a Casio CZ-101), a drum machine (Roland PR-707), a digital effects processor (Lexicon PCM70), a keyboard mixer, and the Fat Mac running Southworth's *Total Music* software.

Paul did not look especially historic or impressive sitting alone in a bathrobe in his living room typing on a Mac and flanked by a few keyboards. But looks are deceiving. To really appreciate what he has done, you have to listen to his album. You hear woodwinds, an accordion, a hammer dulcimer, guitars, an acoustic piano, penny whistles, drums, trumpets, a harp, flutes, tympani, and many other instruments—some of which he invented. And, thanks to the PCM70 (which doubles as a reverberator), the songs sound as if they were played in all sorts of places—from a small, cozy bar to a grand concert hall. I was totally fooled. It is a masterful audio illusion. But it is also fine music.

Using the Mac, Paul called up instrument sounds on each of the keyboards, played them on the Kurzweil (which acted as his master keyboard), and recorded them as a single track on a 3½-inch floppy disk. He replayed the tracks, polished them, then overdubbed new tracks on top of the old. By layering the tracks together, he created the illusion of an entire band or orchestra.

With the press of a button, the computer played all the instruments while Paul taped them on a VHS video recorder. To make the recording sound professional, he used a Sony PCM (pulse code modulation) converter he rented for $10 a day to convert the analog sound signal into a digital signal. When recorded on the videotape, this digital audio equals the sound quality of a compact disc.

Next he went to a local record duplication house, which copied his master tape onto standard audio cassettes—50 at a time.

Not Machine Music

"People expect music made by machines to sound like machine music," says Paul. "I made this album to disprove that. I used the computer to do the things it does well, and made it play the kind of music I wanted to play. I deliberately left in things that some people might call mistakes—little timing things and grace notes. I could have fixed them, but I didn't. I wanted the music to sound like it was made by a human being. Without that human element the music becomes rhythmically perfect all the time. It sounds boring, robotic.

"I wanted to show that a composer could set up everything in his living room and not spend a lot of money. He can create and record his own music all on his own. A system like this cuts all the complications and red tape separating a composer from his audience. Now there is no one between them.

"When I was a kid I went to see Mary Martin do a concert in New York City. I remember she sang a song called 'It Takes Three to Make Music'—one to write, one to play, one to hear. That equation is changing. Now, in a very real sense, the person who writes and the person who plays can be the same person. So it only takes two to make music. And we're not talking about playing a piano, either. We're talking about having an entire orchestra at your disposal. Any kind of orchestra, with any kind of instruments."

To learn more about Paul's system, write to Paul D. Lehrman, 31 Maple Ave. Apt. #1, Cambridge, MA 02139. To get a tape of *The Celtic Macintosh*, send $10. To learn more about *Total Music*, write Southworth Music Systems, Inc., Box 275, RD 1, Harvard, MA 01451. ©

David D. Thornburg, Associate Editor

Metaphorical Computing

Every time we use a computer, we are working with metaphors. In some cases the metaphor is obvious, and in others it is not. A game, for example, might provide us with a model of a city through which we have to navigate without being caught by the bad guys. Rationally, we know there is no city or bad guys, but this metaphor lets us move beyond the computer to the application itself—the computer becomes the mirror of the mind's eye.

This aspect of the computer is what makes it so useful. Metaphors like desktops, menus, and windows can make computing appear more like "real" activities. Those who prefer command line interpreters to desktops are merely expressing a preference for one metaphor over another. None of this matters to the machine. As far as the computer is concerned, code is code and bytes are bytes.

High-level computer languages are programs and are therefore metaphors as well. As the noted computer scientist Edsger Dijkstra says in his book *A Discipline of Programming*: "There are two machines—the physical machine—the one that you pay for, that breaks down, that takes space on your desk. Then there is the abstract machine—defined by the functions the machine is to perform."

As programmers, we trust that the abstract machine is implemented in the physical machine. In fact, it might not be. For example, a program could accept 6502 machine language and convert it to run on a 68000 microprocessor. From the programmer's perspective, the computer is a 6502. The point here is that, even at the most basic level of programming, one cannot escape from metaphors.

Generations Of Languages

If computation is metaphorical at the machine level, it becomes even more so as we move to higher-level languages. In assembly language, for instance, the metaphor of the machine is often expanded to include instructions that the computer cannot execute directly. Programmers call these *macros*, or macro instructions. A macro that simulates a missing multiplication command greatly simplifies the programmer's task.

At the first two levels of computer language (machine and assembly), the metaphor pertains to the machine itself. At this level it is the program's purpose to instruct our computers. Once we move to high-level languages like BASIC, FORTRAN, COBOL, LISP, PROLOG, Forth, and Smalltalk, it becomes the computer's task to execute our programs. This may seem like a subtle distinction, but as we shall see, it is not.

I have come to believe that the major differences between various high-level languages are not to be found in their syntaxes, grammars, or vocabularies, but in their metaphors.

A Pascal programmer, for example, sees a program as a collection of procedures, each of which produces some effect. A LISP programmer sees a program as a collection of functions that output results in much the same manner as mathematical operations. A PROLOG programmer sees a program as a collection of assertions and rules that the computer can use as needed to respond to a query.

Now we can begin to see why it's so difficult to write major application programs. The programmer has the task of creating one metaphor out of another one. This is why multilingual programmers often spend some time deciding which language to use before writing any code.

The trick to making programming easy is to make the metaphor of the language match that of the application.

Construction Sets

Some languages do this quite well. The ease with which a user can create new pinball games with *Pinball Construction Set* is a result of the program's use of the same metaphor, whether a game is being constructed or played.

Other construction set languages (including *Loderunner* and any good spreadsheet program) have become immensely popular. Their popularity arises from ease of use, and this is a direct consequence of the consistency of the metaphor as one moves across the boundary from programming to executing a program.

While construction sets have great value, they also have a strong limitation. They are not general-purpose programming tools. One cannot build a word processor using *Pinball Construction Set* or a music program with *Multiplan*.

The closest we've come to general-purpose direct-manipulation languages is with products like *Filevision*, a Macintosh-based visual database program. I've used *Filevision* since 1984 as a tool for creating instructional software. But even *Filevision* (which was never intended to be a language) has its limitations.

The key to constructing a general-purpose direct-manipulation language is to find a "metametaphor"—a metaphor that encompasses all application metaphors. Does such a metaphor exist? This question is being asked by those of us who are exploring direct-manipulation programming. So far, the answer is not clear.

In the meantime, *metaphors be with you.* ©

Turning Apples Into Oranges

Sometimes it seems as if BASIC is overpopulated with a nearly endless collection of string-handling functions. We've already looked at LEFT$, MID$, RIGHT$, CHR$, ASC, and LEN, not to mention operators for adding strings together and making logical comparisons. Yet, we've barely scratched the surface of what can be done with strings in BASIC. The reason for all these functions is that programmers keep demanding more and more flexibility for manipulating text in their programs. One of BASIC's early predecessors, a language called FORTRAN, was very strong on mathematical functions, but rather weak when dealing with strings. From the very beginning, BASIC has sought to improve in this area.

Two interesting string functions we haven't covered so far are VAL and STR$. These two functions, common to virtually all BASICs, are opposites of each other. STR$ lets you convert a number into a string, and VAL lets you convert a string into a number. For example, look at this program:

```
10 DIM A$(10):REM This line for Atari
   only
20 A$="123"
30 PRINT A$
40 A=VAL(A$)
50 PRINT A
60 A$=STR$(A)
70 PRINT A$
```

When you run this program, the result is:

```
123
123
123
```

So what? you say. We could get the same result by deleting lines 30–70 and simply substituting PRINT A$: PRINT A$:PRINT A$.

But the result would *not* be the same. Although PRINT A$ in line 30 prints the number 123 on the screen, it's printing the number as a *character string*, not a *numeric value*. You can't tell one from another when they're printed on the screen, but the difference is important and matters very much to BASIC. If you attempt a statement such as A=A+"123" or A$=A$+123, BASIC reports some sort of type-mismatch error. Character strings and numeric values just don't mix.

That's why there's VAL and STR$. As shown in line 40, A=VAL(A$) takes the number "123" stored as a *character string* in A$, converts it into the corresponding *numeric value* 123, and stores it in the numeric variable A. Conversely, the statement A$=STR$(A) in line 60 takes the number stored as *numeric value* 123 in A, converts it into the *character string* "123," and stores it in the string variable A$. In short, VAL and STR$ let you turn apples into oranges and back again.

Easier Input

An interesting trick, you say, but what's the purpose? True, VAL and STR$ aren't exactly the most heavily used functions in the BASIC language. Instead, they're like Allen wrenches—once a year when you need them, nothing else will do.

For instance, recently I wrote a short program to experiment with a new computer's sound capabilities. The program asks the user to enter a phone number, then dials the number by playing Touch-Tones through the monitor speaker. To determine which Touch-Tone frequencies to play, I needed to extract each digit in the phone number as numeric data. But to make it as easy as possible for the user to enter the phone number, I wanted the program to accept the keyboard input as a character string. That way, people can type in the phone number any way they want: (919) 555-1212, or 919-555-1212, or 919 555 1212, etc. A statement such as INPUT A$ accepts all those variations. But if the program used INPUT A, the phone number would have to be entered as 9195551212, or an error would result.

So, VAL comes to the rescue. Once the phone number is entered in A$, the program loops through the string, converting digits from 0 to 9 into numbers with VAL. The numbers are then passed along to SOUND statements which play the tones. Spaces, hyphens, parentheses, and other characters are ignored. Here's a simplified version of the routine:

```
100 FOR N=1 TO LEN(A$)
110 IF ASC(MID$(A$,N,1))<48 THEN 150
120 IF ASC(MID$(A$,N,1))>57 THEN 150
130 A=VAL(MID$(A$,N,1))
140 REM SOUND statements to play
    Touch-Tones here...
150 NEXT N
```

This routine conveniently demonstrates several string-handling techniques we've covered in the past few columns. Line 100 sets up a FOR-NEXT loop by using the LEN function to measure the length of A$; this determines how many times the loop repeats (one loop for each character in A$). Lines 110 and 120 use the MID$ function to examine one character at a time in A$; if the ASC function discovers that the character is less than the ASCII value of 48 (the number 0) or greater than the ASCII value of 57 (the number 9), the program skips to line 150 and makes another pass through the loop.

If a character in A$ is a number 0–9, the program falls through to line 130. Here, the VAL function converts the character into a numeric value and stores it in A. Line 140 is where the SOUND statements to play the tones would be inserted. Line 150 then loops back to examine another character in A$.

You can adapt this trick to many of your own programs. Whenever you'd like to accept keyboard input as a string for maximum flexibility, just convert the string with VAL to the numeric input you're really looking for. ©

Arlan R. Levitan

Electronic Bulletin Boards: A Retrospective

Bulletin boards have been with us in one form or another for hundreds of years and will likely stay with us well into the future. Why? What's so special about bulletin boards, electronic or otherwise?

It's difficult to pinpoint when the first bulletin board appeared. Perhaps cave paintings were primitive bulletin boards. In the modern sense of a community communications media, the earliest bulletin board may have been the medieval practice of posting royal proclamations in the center of commerce, the town square.

The traditional bulletin board, with a wide variety of messages tacked to a freely accessible surface, abounds in our supermarkets, factories, offices, schools, laundromats, community centers, and city halls. These bulletin boards are more than just a way to give away kittens or sell tires. They make it possible for people with a message to reach out to the community as a whole.

Electronic Thumbtacks

The thousands of computer-based bulletin board systems (BBSs) which are online today offer the traditional message posting and a great deal more. Imagine trying to maintain a series of communications with other people using a regular bulletin board at a supermarket. Driving to the store every time you want to leave or read a message makes extended communication via corkboard and notecard extremely inconvenient. Even if you make the trip regularly, a less than careful search of the posted messages may miss the very reply that was sought.

The fact that a BBS can be accessed remotely, without leaving one's home, makes an ongoing dialog between many parties a simple matter. A computer dedicated to running the BBS manages the messages; in addition to numbering and indexing the messages, it also automatically notifies its many users of messages intended specifically for them.

The first BBS was born of necessity in 1978. Microcomputers were just getting off the ground, and the first micronauts were few and far between. The four major enclaves of personal computing were located in California, Illinois, Texas, and Massachusetts. Although the computer clubs in these areas exchanged newsletters regularly, there was a decided lack of spontaneous interaction between the major groups and even within the groups themselves.

Ward Christensen and Randy Suess, both members of the Chicago Area Computer Hobbyist Exchange (CACHE), came up with the answer. They developed a program to run on a computer that was equipped with a modem hooked up to a phone line. The program turned the computer into an automated message system. Callers to the Computerized Bulletin Board System (or CBBS, as its originators referred to it) could leave and retrieve messages at any time of day. The CBBS was a huge success, and other clubs began pressing personal computers into service as bulletin boards.

The Spread Of BBSs

CBBS was not a universal program. It was written for computers which used the CP/M operating system (Control Program for Microcomputers). Christensen and Suess wrote a widely publicized article describing the program and the structure of their system as it appeared to the person calling into the CBBS. Realizing that similar programs would be written for other types of computers, they proposed that the functions and commands used by the CBBS be standardized for all BBSs. This would make it unnecessary for people to learn a whole new set of commands for each type of board they accessed.

Sure enough, BBS software for other popular systems soon followed. Craig Vaughn and Bill Blue created a program for Apple II computers called the People's Message System (PMS). Close on their heels was Bill Abney, who produced Forum 80 for the Radio Shack TRS-80, and Tom Giese, father of the Atari Message & Information System (AMIS) for the Atari 400 and 800. Late in 1982, the first version of the Remote Bulletin Board System (RBBS) for the IBM was written by D. Thomas Mack and Jon Martin.

Aside from a message exchange, most BBSs offer a selection of public domain programs and other types of files. By using terminal software capable of receiving files via modem from a remote computer, callers can transfer (download) copies of these files from the BBS to their own machines.

Most of the free software available from BBSs consists of programs that computer enthusiasts like yourself have written and wish to share with other people. A plethora of games, word processors, spreadsheets, database managers, and terminal programs are available for the price of a phone call. Whatever your needs, you can acquire a respectable library of almost-free software that will handle all but the most demanding tasks.

Next Month: Current Trends in Bulletin Board Systems. ©

Bill Wilkinson

Tried And True Tools

In keeping with COMPUTE!'s programming languages theme for this month, I'd like to share some thoughts about programming in general and better use of the available languages in particular. I have long contended that, for most purposes, owners of Atari 400/800, XL, and XE computers have all the languages they need. You won't do parallel array processing with a 6502, no matter what language you use, but you *can* balance your checkbook, keep track of your mailing list, access online services via modem, write a book or two, and (of course) play some games. All of those applications and many more have been written with languages now available for the eight-bit Atari computers. What more can you ask for?

In a previous column I said it would be hard for most users to justify trading up to an ST, an Amiga, or whatever. If anything, I feel more strongly about that now. I still write this column using a good old Atari 1200XL (I like its keyboard best) and an Atari 825 printer (ancient history). Sometimes I wish for an 80-column screen or a hard disk drive—keeping track of 200 floppies is not my idea of fun—but I can't justify the expense for the extra convenience.

The same is true when it comes to programming languages. Admittedly, I'm a language junkie. I love learning new languages and/or tricks with old languages. So it would seem that the ST would be a dream machine for me. Despite its youth, the number of languages either available or coming soon is phenomenal: several varieties of BASIC, Logo, Pascal, C, LISP, Modula-2, COBOL, FORTRAN, Prolog, Forth, and 68000 machine language. There are probably others, too.

Old Machines, New Projects

But for owners of eight-bit Ataris, the situation is far from bleak. Though some of the language implementations are not as rich as those on the ST, we can enjoy Pascal, C, Logo, Action!, Forth, PILOT, 6502 machine language, and some extraordinarily easy-to-use BASICs.

Even though I've been using Ataris for six years now, I still see some interesting projects to do—projects that I've never done or which I think can be done better. A few examples: How about a terminal program written in Action! that is designed to work well with CompuServe's conference mode? Or a GEM-like interface for DOS? Or a combined spreadsheet/database written in BASIC XE and commented liberally so that even beginners can see the methods used? I know I'll never do all of these, but they are challenges I'd *like* to tackle.

Rethinking The Problem

Moving to new languages on new machines is not always an advantage. For instance, in ST BASIC, strings cannot exceed 255 characters in length. Atari BASIC strings can be up to 32,767 characters long, if you have the memory available. (Yes, ST BASIC allows string arrays, but so do BASIC XL, BASIC XE, and Atari Microsoft BASIC.) There are many other examples.

Another factor is that the speed and power of the newer machines is of little advantage for some applications. Other than missing an 80-column screen, I can use CompuServe or various bulletin boards just as well with my $100 computer as I can with the company's $1,000 machine. Besides, the modem for the $100 computer is cheaper. And by the time you read this, Atari may have released its 80-column adapter for the eight-bit line.

Suppose you're writing a program that *does* need more speed, however. What can you do other than buy a newer, faster computer? Well, you could buy a better, faster language. That's a lot cheaper than buying a new computer—for which you still might need an extra language or two. On the other hand, maybe you don't have to buy anything at all; maybe you just need to rethink your solution to the problem. Let me show you what I mean.

Program 1 is very similar to one which I found in a recent user group newsletter. The author was responding to a member's inquiry about writing a routine to shuffle a deck of cards. As you know, when BASIC gives you a random number, there's no guarantee it won't give you that same random number twice, perhaps even several times. For a quick example, type the following line and press RETURN:

```
FOR I=0 TO 9:PRINT INT(10*RND
(0)):NEXT I
```

This asks for ten random numbers in the range 0 to 9. Did you actually get ten different numbers? The odds are very much against it.

The Super Shuffle

Program 1 demonstrates this problem by dealing out an entire deck of 52 cards. As each card is dealt (by suit S and rank R, line 210), its spot in the C (card) array is marked. Then, if the random number generator picks that card again, the pick is ignored (line 230). The only things I added to the original routine are the counters (C and T) which count how many picks it takes to get each card and the entire deck. Can you guess how many picks it takes to get the entire deck? In 50 tests, it took a minimum of 128 picks and a maximum of 457, with the average around 220. The result, as you'll see if you run the program, is that it can take as long as five or six seconds to pick a card.

Now look at Program 2, which does exactly the same job but never takes more than one pick to get the next card in the deck. It works by using a single string (CARD$) to represent the entire deck. When it gets a random number from 1 to 52, the program removes the corresponding "card" from the "deck" (lines 400 and 410). The next time it picks a card, it gets a random number from 1 to 51. Each time the computer gets a card, the range of random numbers gets smaller. Simple. And it works by taking advantage of the string operations in Atari BASIC.

The point of this exercise is to show that sometimes the best way to fix a slow or inefficient program is to rethink it and then rewrite it. I'd be willing to bet that Program 2 on an eight-bit Atari runs faster than Program 1 on an Atari ST. If you have access to both machines, you might want to try it. And try improving your own programs. (Even while writing this column, I found a way to improve Program 2 even more. Can you find it?)

One last comment: Notice the readability of the two programs. Which one is cryptic and which one almost explains itself? Meaningful variable names can add a great deal of value to any program.

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

Program 1: Slow Shuffle

```
MD 100 DIM C(4,13)
OP 110 DIM R$(13):R$="A23456
       789TJQK"
IG 120 DIM S$(4):S$="{;}{,}
       {.}{P}"
AH 130 FOR S=1 TO 4:FOR R=1
       TO 13:C(S,R)=0
PC 140 NEXT R:NEXT S
CE 200 FOR I=1 TO 52:C=0
NC 210 S=INT(RND(0)*4)+1:R=I
       NT(RND(0)*13)+1
LD 220 C=C+1
PF 230 IF C(S,R)<>0 THEN 210
BJ 240 C(S,R)=1
OK 250 T=T+C
LB 260 PRINT I;:POKE 85,15-L
       EN(STR$(C)):PRINT C;"
        PICK(S) TO GET ";R$(
       R,R);" OF ";S$(S,S)
CB 270 NEXT I
PJ 280 PRINT :PRINT "TOTAL P
       ICKS: ";T
```

Program 2: Fast Shuffle

```
ED 100 REM === SET UP VARIAB
       LES,ETC. ===
GC 110 DIM CARD$(52)
BA 120 DIM SUITS$(8*4)
CA 130 SUITS$="SPADES  HEART
       S  CLUBS{3 SPACES}DIA
       MONDS"
GB 140 DIM SUIT$(8)
PH 150 DIM RANKS$(4*5)
EE 160 RANKS$="ACE  JACK KIN
       G QUEEN"
EI 170 DIM RANK$(5)
GN 200 REM === SET UP THE DE
       CK ===
AM 210 FOR CARD=1 TO 52
OG 220 CARD$(CARD)=CHR$(CARD
       )
OO 230 NEXT CARD
IM 240 DECKSIZE=52
MP 300 REM === DEAL 52 CARDS
        ===
AN 310 FOR CARD=1 TO 52
HC 320 PICK=INT(DECKSIZE*RND
       (0))+1
LP 330 PICKED=ASC(CARD$(PICK
       ))-1
JI 340 SUIT=INT(PICKED/13)
LB 350 RANK=PICKED-13*SUIT
IF 360 SUIT$=SUITS$(SUIT*8+1
       ,SUIT*8+8)
HD 370 IF RANK<4 THEN RANK$=
       RANKS$(RANK*5+1,RANK*
       5+5)
LK 380 IF RANK>=4 THEN RANK$
       =STR$(RANK-2)
DK 390 PRINT "Picked: ";RANK
       $;" OF ";SUIT$
AH 400 IF PICK<DECKSIZE THEN
        CARD$(PICK)=CARD$(PI
       CK+1)
EA 410 IF PICK=DECKSIZE THEN
        CARD$(PICK)=""
NF 420 DECKSIZE=DECKSIZE-1
PA 430 NEXT CARD
```

©

INSIGHT: ST

Bill Wilkinson

Odd Facets Of GEM

This month we're going to explore a handful of quirks in the GEM desktop: things you can do to make your system more useful...things you can do to make your system crash! Since I'm a natural-born pessimist, let's start with the crash.

The bug I'm about to demonstrate infests the ROM (Read Only Memory) version of the TOS operating system. Even so, if you don't already have the TOS ROMs, get them today. The difference in overall system performance and capability is only a little short of great.

To see this bug, simply boot your system and bring up the Control Panel. Select either the date or time field. Then type an underline character (SHIFT-hyphen). Watch your system bomb. The only recovery is to press the reset button or turn off the power.

Problem: The Control Panel is a form of dialog box, and it uses what are known as *editable text fields* to display and let you modify the date and time. An editable text field is designed to restrict the user to typing certain characters. For example, the date and time fields of the Control Panel are editable fields which allow only numbers to be typed. Unfortunately, somehow a bug crept into the ROM-based TOS. Anytime you edit a numeric-only field, typing the underline causes something nasty to occur. Editable fields for filenames have a similar, though usually nonfatal, problem.

Solution: A GEM application program that needs to accept numeric-only input from the user has two choices: (1) Use an editable field which allows *any* character and then validate the user's input after the dialog box returns; (2) Retrieve keystrokes one at a time, checking them on the fly, and print only the valid ones on the screen. The former solution is kind of ugly because the user doesn't get immediate response to incorrect input. The latter solution is a lot of work. Take your pick.

Modifying DESKTOP.INF

Many of you already know how to customize the GEM desktop so your preferences appear automatically when you boot up the ST. When you select the *Save Desktop* item under the *Options* menu, GEM saves a file to the disk in drive A called DESKTOP.INF which stores these preferences. You can rearrange the icons on the screen, change screen colors, resize the windows, and so on, and GEM remembers it all for you.

DESKTOP.INF is an ordinary ASCII file, so it can be modified with most text editors and word processors. This lets you personalize GEM even more. (See "ST Hints & Tips," COMPUTE!, June 1986.)

The first thing we'll do is the easiest. Using a text editor or word processor that handles ASCII files, load and examine DESKTOP.INF. You should see one or two lines which contain the words FLOPPY DISK (among other things). These are the labels which appear beneath the disk icons. I usually rename the labels *—Top—Disk—* and *Bottom—Disk*. (I've used dashes here to show where I typed a space—magazine typesetting sometimes makes it hard to indicate spaces.)

Save the modified file back on disk in ASCII format. The next time you boot from that disk, the names should appear as you have modified them. Just for fun, sometimes I change the name of the trash can to *Junk!* or *Garbage* or something equally silly.

Rearranging Files

There are even more interesting things you can do with DESKTOP.INF. If, like me, you have a disk or subdirectory in which you do most of your work, you'll soon find that you can't see all of the filenames or icons on the screen at once. Although it's a minor nuisance, it always seems that the files (or, more likely, programs) which I want the most are always off the screen. How can we force them back on the screen? Preferably in the upper-left position?

One solution, since the default display mode under the *Show* menu is *Sort by Name*, is to name your favorite files AARDVARK.PRG or AAABASIC.PRG. But that's kind of messy. A better method might be to choose *Sort by Date* if you could change the file's creation date. But I think Mark Rose (of Optimized Systems Software) has hit upon the best scheme.

First, he chooses *Sort by Type*. Second, he renames his most-used programs so they have no type (filename extension) at all. Third, he loads DESKTOP.INF and adds a line or so. To figure out exactly what to add, look for a line in DESKTOP.INF similar to this:

```
#G 03 FF *.PRG@ @
```

This line tells the desktop that all files which match the *.PRG specifier are GEM (G) program files. Now, let's say the program you want to appear at the top-left of the screen was called PASCAL.PRG and has been renamed to simply PASCAL. You would add this line to the end of the DESKTOP.INF file:

```
#G 03 FF PASCAL.@ @
```

This tells GEM that PASCAL is actually a GEM-based program. Neat, huh? What's more, you can do this for several files. However, I *do not* recommend using the *. wildcard in such a line—general untyped files end up looking like programs, a dangerous practice. ©

Programming the TI

C. Regena

A Beginning Reading Program

When my youngest son was learning to read, I wrote lists of words for him to practice with. The word lists used a certain word ending coupled with various beginning letters—such as AT, BAT, CAT, FAT, HAT, MAT, PAT, RAT, SAT, TAT, VAT, and THAT. After I wrote the same list several times, I realized this was another idea for a computer application.

Not only can the TI-99/4A print the words nicely on the screen, but with the TI Speech Synthesizer it can also speak the words aloud. This month's example program, "Reading Practice," takes advantage of this feature. Therefore, it requires both the TI Speech Synthesizer and the Terminal Emulator 2 command module. Insert the module and select 1 for TI BASIC to gain access to speech.

Reading Practice

Notice line 154 in the program listing below. This OPENs the channel for speech, so any subsequent PRINT #1 statement will speak the word. The words I selected for examples are all pronounced correctly by the speech synthesizer as they are spelled. If you add words, check their pronunciation; you may need to add a routine so the word is pronounced correctly.

If you want to run this program without speech, omit all statements that refer to file 1 or insert a REM in front of each of these commands. Another alternative is to make speech an option; insert IF-THEN statements before the speech commands.

Due to space limitations, I had to keep Reading Practice rather short. Feel free to add graphics, sound, and additional word lists. I included just a few for examples. You could also modify the program to offer the student various groupings of words.

When you run Reading Practice, you'll see a word ending in lowercase letters. To hear the word pronounced aloud, press the space bar. To go to the next word, press ENTER. The screen clears and you'll see a word with a beginning letter and that word ending.

The words are in the DATA statements starting at line 198. First there is a word ending, then all the beginning letters that go with the word ending and create real words. ZZZ indicates no more beginning letters for that ending. At the end of all the lists, @@@ indicates the end of data.

Lines 126–146 redefine characters to draw larger lowercase letters. Be careful typing the DATA statements; they contain the character definitions. If you have my program for lowercase letters (COMPUTE!, August 1983), you can take a shortcut. First, load that program and delete all the PRINT statements. Type RES 126,2 to renumber the lines. Then add the rest of the Reading Practice program.

Lines 20–110 contain subroutines to draw the 26 letters of the alphabet. The variable R is the main row number and C is the column number. If a lowercase letter has an ascender or descender, more than one character is required to draw the letter. Line 174 goes to the proper subroutine for each letter in the word. Since there are 26 possible letters, I needed low line numbers to use a single ON-GOSUB statement (otherwise more complex logic and several ON-GOSUB statements would be required). Therefore, the subroutines are near the beginning of the program, and the program starts with line 2 and increments by 2. To type in this program using the automatic numbering feature, use NUM 2,2.

TI BASIC String-Handling

Reading Practice uses several string functions. It reads the letters from the DATA statements into X$ for the word ending and B$ for the beginning letters. W$ holds the word to be printed and read. LEN(W$) returns the length of the word (or word ending). Lines 170–178 contain a FOR-NEXT loop that executes for each letter of the word. SEG$ gets one character at a time from the word, and ASC returns the ASCII value of that letter. In TI BASIC, the ampersand symbol (&) is used to combine strings. (See "The Beginner's Page," June 1986.)

Line 194 combines the beginning letter B$ with the word ending X$ to form the word W$. Line 186 adds a period to the word to change its inflection for the speech synthesizer. You may prefer to simply use W$.

Versions of BASIC on other computers generally use the string functions RIGHT$, LEFT$, and MID$ for extracting sections of strings. The equivalent in TI BASIC is SEG$. SEG$(W$,A,B) looks at the string W$, starts with character number A, and returns B number of characters. For example, let's assume that W$="RICHARD". The Microsoft BASIC statement LEFT$(W$,4) is translated into TI BASIC as SEG$(W$,1,4), which yields RICH. The statement RIGHT$(W$,4) is translated as SEG$(W$,LEN(W$)−4+1,4), which yields HARD. And the statement MID$(W$,3,2) is translated as SEG$(W$,3,2), which yields CH. (See "The Beginner's Page," April 1986.)

In the Reading Practice program, SEG$(W$,P,1) is in a loop where P starts at 1 and goes to L, which is the number of characters in the word. This function, then, gets one letter at a time, in order, from the word. The variable A is the ASCII value of that character minus 64 to yield numbers from 1 to 26 for the relative letter. Line 174 then uses ON-GOSUB to go to the proper subroutine to draw the corresponding lowercase letter.

If you want to save typing effort, you may get a copy of this program by sending a blank cassette or disk, a stamped, self-addressed mailer, and $3 to:

C. Regena
P.O. Box 1502
Cedar City, UT 84720

Be sure to specify the title "Reading Practice." This program is available for the TI computer only.

Reading Practice

```
2 REM  READING
4 REM  REQUIRES TERMINAL
6 REM    EMULATOR 2
8 REM  REQUIRES SPEECH
10 REM    SYNTHESIZER
12 REM  ** WITHOUT SPEECH--

14 REM    REMOVE STATEMENTS
16 REM    CONTAINING "#1"
18 GOTO 112
20 CALL HCHAR(R,C,A+96)
22 RETURN
24 CALL HCHAR(R-1,C,104)
26 GOTO 20
28 CALL HCHAR(R-1,C,100)
30 CALL HCHAR(R,C,97)
32 RETURN
34 CALL HCHAR(R-1,C,102)
36 CALL HCHAR(R,C,108)
38 RETURN
40 CALL HCHAR(R,C,97)
42 CALL HCHAR(R+1,C,103)
44 RETURN
46 CALL HCHAR(R-1,C,104)
48 CALL HCHAR(R,C,110)
50 RETURN
52 CALL HCHAR(R-1,C,105)
54 CALL HCHAR(R,C,108)
56 RETURN
58 GOSUB 52
60 CALL HCHAR(R+1,C,106)
62 RETURN
64 CALL HCHAR(R-1,C,104)
66 GOTO 20
68 CALL VCHAR(R-1,C,108,2)
70 RETURN
72 CALL HCHAR(R,C,110)
74 C=C+1
76 CALL HCHAR(R,C,109)
78 RETURN
80 CALL HCHAR(R,C,98)
82 CALL HCHAR(R+1,C,112)
84 RETURN
86 CALL HCHAR(R,C,97)
88 CALL HCHAR(R+1,C,113)
90 RETURN
92 CALL HCHAR(R-1,C,116)
94 CALL HCHAR(R,C,108)
96 RETURN
98 CALL HCHAR(R,C,118)
100 C=C+1
102 CALL HCHAR(R,C,119)
104 RETURN
106 CALL HCHAR(R,C,118)
108 CALL HCHAR(R+1.C.121)
110 RETURN
112 CALL CLEAR
114 PRINT "** READING PRACT
    ICE **"
116 PRINT ::::"READ THE WOR
    D ON THE SCREEN."
118 PRINT ::"PRESS THE SPAC
    E BAR TO HEAR"
120 PRINT ::"THE WORD."
122 PRINT ::"PRESS <ENTER>
    TO GO TO THE"
124 PRINT ::"NEXT WORD.":::
126 FOR C=97 TO 122
128 READ C$
130 CALL CHAR(C,C$)
132 NEXT C
134 REM  REDEFINE LOWERCASE

136 DATA 3D4381818181433D,B
    CC2818181C2BC,3C42808
    080423C,000001010101
    01,3C4281FF8080423C
138 DATA 06090808080883E,0
    1010141221C,00008080
    080808,00000008,0808080
    808887,88900A0C0A0908884
140 DATA 0808080808080808,7
    884020202020202,BCC2818
    18181818,3C428181814
    23C,808080808080,010101
    0101
142 DATA BCC281808080808,3C
    42403C0202423C,00000808
    08087F08,818181818143
    3D,41412222141408808,040
    48888505020202
144 DATA 824428102844828,1
    010202040,7F0204081020
    407F
146 REM
148 PRINT "NOW PRESS <ENTER
    > TO START."
150 CALL KEY(0,K,S)
152 IF K<>13 THEN 150
154 OPEN #1:"SPEECH",OUTPUT
156 R=10
158 READ X$
160 IF X$="@@@" THEN 216
162 W$=X$
164 CALL CLEAR
166 L=LEN(W$)
168 C=12
170 FOR P=1 TO L
172 A=ASC(SEG$(W$,P,1))-64
174 ON A GOSUB 20,24,20,28,
    20,34,40,46,52,58,64,68
    ,72,20,20,80,86,20,20,9
    2,20,20,98,20,106,20
176 C=C+2
178 NEXT P
180 CALL KEY(0,K,S)
182 IF K=13 THEN 190
184 IF K<>32 THEN 180
186 PRINT #1:W$&"."
188 GOTO 180
190 READ B$
192 IF B$="ZZZ" THEN 158
194 W$=B$&X$
196 GOTO 164
198 DATA AT,B,C,F,H,M,P,R,S
    ,T,V,TH,ZZZ
200 DATA AN,B,C,F,M,P,R,T,V
    ,TH,ZZZ
202 DATA ED,B,F,L,R,W,ZZZ
204 DATA IN,B,F,K,P,S,T,W,Z
    ZZ
206 DATA IT,B,F,H,K,L,P,S,W
    ,ZZZ
208 DATA OG,C,D,F,H,J,L,ZZZ
210 DATA UG,B,D,H,J,L,M,R,T
    ,ZZZ
212 DATA AND,B,H,L,S,ZZZ
214 DATA @@@
216 CALL CLEAR
218 CLOSE #1
220 END
```

©

Programming In Modula-2

There are a plethora of programming languages for the Amiga, giving programmers and developers a wide choice of programming styles and systems. There are two versions of BASIC (MetaComCo's ABasiC and Microsoft's Amiga BASIC), two C compilers (Manx Aztec C and Lattice C), a macro assembler/editor, two versions of Pascal, and even an implementation of LISP. Numerous programming tools, such as editors and debuggers, are also available.

A relative newcomer to the scene, TDI Modula-2, is now getting some attention. Some programmers consider it easier to learn and use than C—since it shares many of the high-level aspects of Pascal—while still retaining a machine-level interface for maximum efficiency.

Modula-2 is a descendent of the language Modula, which in turn is a descendent of Pascal. Nicklaus Wirth, the inventor of Pascal, designed Modula from the roots of Pascal, but purposely kept it very simple so that it could be used with very small computers—primarily for controlling hardware devices such as robot arms. The original Modula had little application outside a very specialized world, so Wirth put back most of the features of Pascal to create Modula-2. TDI has worked directly with Wirth to implement versions of Modula-2 for the Amiga and Atari ST.

Software Chips

The concept of Modula-2 is echoed in its name. It is a language designed specifically for the techniques of modular programming, just as Pascal was designed to make structured programming convenient and elegant. Modular programming—the art of breaking a large, complex problem into small, independent tasks—is at the heart of all programming, but Modula-2 tries to bring to software the modularity inherent in computer hardware, based on off-the-shelf chips and components. With "software chips," Wirth envisioned, software technology could advance apace with the remarkable speed of hardware evolution.

If software chips are possible, they have to be based on program modules that can be truly independent, hence, individually testable. You can compile a module without having to recompile the entire program. A module, once developed, becomes a "black box" routine that accepts input and/or provides output. You no longer need to know how this module works internally to use it—you just plug it in and go. Writing a program becomes a task of putting together these building blocks in the right way without ever needing to reinvent the wheel. Why solve a problem when someone else has already found the solution?

Modula-2 comes with a standard library containing modules for input/output, math routines, and access to special machine features. You use only the routines you're interested in, and only these routines (and the underlying routines they are based on) need to be included in your compiled code. This lets you control the size of your final program.

You can easily add your own library modules. First, you write the *definition module*, which simply contains the procedure headers that specify the inputs and outputs of a module. The definition module primarily specifies the names of these procedures. It compiles to a symbol file for use by the compiler. The *implementation module* contains the actual code of the module. You compile the definition module separately from the implementation module.

You can change and recompile the implementation module without changing the definition module, as long as your procedure headings remain the same. When you're referencing library modules, the compiler can check the compact, compiled symbol file rather than the full-length definition module, speeding up compilation. After compilation, a linker combines your main program with the compiled implementation modules to create the final executable program.

Reminiscent Of Pascal

One of the best ways to learn about a language is to study an example program. The program accompanying this column is written to demonstrate some of the features of Modula-2 without getting bogged down in tricky algorithms. It's a simple guess-my-number game. The RandomNumbers module thinks of a number from 1 to 100. The program then gives you ten tries to guess the number, helping out with hints. If you guess too high, the program recommends that you try a smaller number. If you guess too low, you should try a higher number.

Here's how the program works. The first line declares the name of the module. Next, the IMPORT statements specify which external library calls we'll be using. Then we declare the variables. We define the procedure SkipEOL, used to strip away the rest of a line after getting a single-character response. The main loop follows, enclosed by the keywords BEGIN and END. (All Modula-2 keywords must be typed in uppercase, which can be annoying.)

Most of the program looks very much like Pascal, especially the use of := for assignments and the required semicolon at the end of each logical line. Also, you won't find GOTO anywhere in this or any Modula-2 program. Instead, you can control looping and program execution with statements like LOOP-EXIT-END, WHILE-END, REPEAT–UNTIL, and IF-THEN-ELSE-END.

You might be interested to know that this program compiles in 35 seconds when the source code is stored in the RAM disk; it takes 37 seconds to compile when the source code is stored on a floppy disk. Linking takes 45 seconds from the RAM disk, and just one minute from a floppy disk. This is quite a bit faster than Lattice C and compares well with Aztec C.

There's much more to Modula-2 than this discussion can encompass. The language even permits procedures to run as multitasking programs. Our example doesn't show how easily Modula-2 can take advantage of the Amiga operating system—even a small program would be too large to demonstrate here—but the interface is similar to C's, using Pascal-style RECORDs instead of C structures. It's possible to develop modules that support the Amiga operating system on a higher level, using calls like Screen(320,200,5) to open a custom screen as opposed to filling in the blanks of a NewScreen structure, opening the Intuition library, and calling OpenScreen( ). Some high-level modules are included in the library. When these modules are developed and shared between Modula-2 programmers, Amiga programming in Modula-2 can seem almost as easy as in BASIC, but with every advantage of a modern compiled language.


```
MODULE Example;

FROM InOut IMPORT EOL,Read,ReadInt,ReadString,
     WriteLn,WriteString,WriteInt;
FROM RandomNumbers IMPORT Random;

VAR
 MyNum, Guess, Tries : INTEGER;
 Again : CHAR;

(* Skips until end of line is reached *)
PROCEDURE SkipEOL;
  VAR temp : CHAR;
BEGIN
  REPEAT
    Read(temp);
  UNTIL temp=EOL;
END SkipEOL;

(* The main loop *)
BEGIN
  LOOP
    MyNum := Random(100)+1;
    WriteString("I'm thinking of a ");
    WriteString("number from 1 to 100");
    WriteLn;
    Tries :=0;
    LOOP
      Tries := Tries+1;
      IF Tries>10 THEN EXIT; END;
      WriteString("Guess #");
      WriteInt(Tries,2);
      WriteString("? ");
      ReadInt(Guess);
      WriteLn;
      IF (Guess=MyNum) OR (Guess=0) THEN EXIT; END;
      IF Guess<MyNum THEN
        WriteString("Try a larger number.");
      ELSE
        WriteString("Try a smaller number.");
      END; (* IF *)
      WriteLn; WriteLn;
    END; (* LOOP *)
    IF Tries>10 THEN
      WriteString("You only get 10 tries!");
    END; (* IF *)
    IF Guess=MyNum THEN
      WriteString("You guessed my number!");
      WriteLn;
      WriteString("After ");
      WriteInt(Tries,2);
      WriteString(" tries.");
    END; (* IF *)
    WriteLn; WriteLn;
    WriteString("Play again? (Y/N): ");
    Read(Again);
    SkipEOL; (* skip ahead to next line *)
    IF (Again='N') OR (Again='n') THEN EXIT; END;
  END; (* LOOP *)
END Example.
```

©

Donald B. Trivette

Hard Disks And The Home PC

Technology marches on: The price of a hard disk drive—a $2,000 luxury just a few years ago—has fallen to a point where it's becoming affordable for home-based computers. Mail-order houses are offering internal hard disks for less than $400, and prices of hard disk cards are coming down as well. If you're thinking about upgrading your IBM PC with one of these super storage devices, here are some points to consider.

Hard disk, fixed disk, and *Winchester disk* are all names for the same thing—a device with a rigid magnetic disk permanently sealed in a box. Because it's sealed from airborne contaminants and has a hard rather than a flexible surface, it can record larger amounts of data than a floppy disk, and the data can be read or written much faster. The smallest hard disks commonly in use store 10 megabytes of data—that's 10,240 kilobytes, or 10,485,760 characters. In comparison, a standard IBM floppy disk stores only 360 kilobytes—368,640 characters. Hard disks are available for the PC with capacities of 20, 50, and even 100 megabytes, although a home user isn't likely to need more than 10 megabytes.

Hard disks come in two forms: the internal type that fits into one of the spaces once occupied by a floppy disk drive (half-height or full-height), and a newer configuration called a hard disk card that squeezes the disk onto a printed circuit board which plugs into one of the PC's expansion slots.

Can You Spare A Slot?

Every hard disk must have a hard disk controller. An ordinary internal hard disk requires a separate controller board connected via cables to the drive. One advantage of the hard disk cards is that both the disk and the controller are a single compact unit. Either way, one expansion slot is used. But there is an alternative for internal models. For about $50 extra, you can order a controller that runs existing floppy disk drives as well as the hard drive. This allows you to remove (and scrap) your floppy disk controller board and free up one expansion slot. None of the hard disk cards introduced so far can control floppy disk drives.

Another factor to consider is power consumption. Hard disk cards are designed to work with the 63-watt power supply found in the IBM PC; many of the internal hard disks were designed for the PC-XT, which has a larger power supply. Frankly, if your PC is already brimming with boards—parallel, serial, color/graphics, and game ports as well as memory expansion—you should strongly consider replacing the old power supply no matter which type of hard disk you install. The operation is as simple and safe as removing some screws and unplugging some wires. A new 135-watt power supply can be purchased for as low as $90.

Of course, speed and reliability are prime considerations when investing in a hard disk. Although reliability is difficult to measure without an industry-wide mean-time-between-failure test, there are some things you can check. An oxide coating on the disk surface is more stable and thus more reliable than plated or sputtered media. The type of actuator that moves the read/write heads across the surface of the platter not only affects speed, but also influences accuracy. A voice-coil actuator is faster and more reliable (and more expensive) than the more common stepper-motor actuator. Therefore, look for a hard disk with an oxide coating and voice-coil actuator.

Fast, Faster, Fastest

Speed varies greatly depending on the make and model. Consider the results of a test which measures a mixture of 1,000 sequential and random accesses—a test that is typical of how real computer programs use a hard disk. The times range from 12 milliseconds for a high-performance internal drive like Core International's AT line, to more then 100 milliseconds for some of the inexpensive hard disk cards. A hard disk for home use—where you don't need top performance—should have a sequential/random access time in the 30- to 60-millisecond range.

Cost is usually a major consideration when selecting a hard disk for home use. Hard disk cards cost as little as $550 for a 10-megabyte unit to as much as $1,200 for 20 megabytes. Internal 20-megabyte drives, including the controller board, are generally in the $400–$600 range; 10-megabyte models cost about $100 less. The best advice here is not to choose a disk by cost alone. Consider all the factors.

My own PC has a 20-megabyte internal Seagate drive, a new 135-watt power supply, and a controller that also runs two half-height floppy drives. It cost about $650 and took two hours to install. An alternative I'd be comfortable with is Plus Development's 10-megabyte Hardcard; it's designed for those who want a quick, simple installation and who don't want to fool with a new power supply. The Hardcard, at about $900, is a well-engineered solution to upgrading your computer's mass storage capabilities.

One final caveat: Whether you spend ten minutes or two hours installing the hardware, plan to spend lots more time learning to use the DOS commands that are necessary to manage files on a hard disk. ©

MLX Machine Language Entry Program For Commodore 64

Ottis Cowper, Technical Editor

"MLX" is a labor-saving utility that allows will help you enter machine language program listings without error. MLX is required to enter all Commodore 64 machine language programs published in COMPUTE!.

Type in and save some copies of MLX (you'll want to use it to enter future ML programs from COMPUTE!, COMPUTE!'s GAZETTE, and COMPUTE! books). When you're ready to enter an ML program, load and run MLX. You'll be asked for a starting address and an ending address. These addresses should appear in the article accompanying the MLX-format program listing you're typing.

If you're unfamiliar with machine language, the addresses (and all other values you enter in MLX) may appear strange. Instead of the usual decimal numbers you're accustomed to, these numbers are in *hexadecimal*—a base 16 numbering system commonly used by ML programmers. Hexadecimal—hex for short—includes the numerals 0–9 and the letters A–F. But don't worry—even if you know nothing about ML or hex, you should have no trouble using MLX.

After you enter the starting and ending addresses, you'll be offered the option of clearing the workspace. The data you enter with MLX is kept in a special reserved area of memory; clearing this workspace fills the reserved area with zeros, which makes it easier to find where you left off typing if you enter the listing in several sessions. Choose this option if you're starting to enter a new listing. If you're continuing a listing that's partially typed from a previous session, there's no point in clearing the workspace, since the data you load in will fill the area with whatever values were in workspace memory at the time of the last Save.

At this point, functions menu will appear. If you're just starting to type in a program, pick the first option, ENTER DATA, by pressing the E key. You'll be asked for an address; type the four-digit number at the start of the first line of the program listing. If you've already typed in part of a program, be sure to load the partially completed program before you resume entry, then choose the ENTER DATA option and type the line number where you left off typing at the end of the previous session. In any case, make sure the address you enter corresponds to the address of a line in the listing. Otherwise, you'll be unable to enter the data correctly. If you pressed E by mistake, you can return to the command menu by pressing RETURN alone when asked for the address. (You can get back to the menu from most options by pressing RETURN with no other input.)

Entering A Listing

Once you're in Enter mode, MLX prints the address for each program line for you. You then type in all nine numbers on that line, beginning with the first two-digit number after the colon (:). Each line represents eight data bytes and a checksum. Although an MLX-format listing appears similar to the "hex dump" listings from a machine language monitor program, the extra checksum number on the end allows MLX to check your typing.

When you enter a line, MLX recalculates the checksum from the eight bytes and the address and compares this value to the number from the ninth column. If the values match, you'll hear a bell tone, the data will be added to the workspace area, and the prompt for the next line of data will appear. But if MLX detects a typing error, you'll hear a low buzz and see an error message. The line will then be redisplayed for editing.

Invalid Characters Banned

Only a few keys are active while you're entering data, so you may have to unlearn some habits. You *do not* type spaces between the columns; MLX automatically inserts these for you. You *do not* press RETURN after typing the last number in a line; MLX automatically enters and checks the line after you type the last digit.

Only the numerals 0–9 and the letters A–F can be typed in. If you press any other key (with some exceptions noted below), you'll hear a warning buzz. MLX checks for transposed characters. If you're supposed to type in A0 and instead enter 0A, MLX will catch your mistake. There is one error that can slip past MLX: Because of the checksum formula used, MLX won't notice if you accidentally type FF in place of 00, and vice versa. And there's a very slim chance that you could garble a line and still end up with a combination of characters that adds up to the proper checksum. However, these mistakes should not occur if you take reasonable care while entering data.

Editing Features

To correct typing mistakes before finishing a line, use the INST/DEL key to delete the character to the left of the cursor. (The cursor-left key also deletes.) If you mess up a line really badly, press CLR/HOME to start the line over. The RETURN key is also active, but only before any data is typed on a line. Pressing RETURN at this point returns you to the command menu. After you type a character of data, MLX disables RETURN until the cursor returns to the start of a line. Remember, you can press CLR/HOME to quickly get to a line number prompt.

More editing features are available when correcting lines in which MLX has detected an error. To make corrections in a line that MLX has redisplayed for editing, compare the line on the screen with the one printed in the listing, then move the cursor to the mistake and type the correct key. The cursor left and right keys provide the normal cursor controls. (The INST/DEL key now works as an alternative cursor-left key.) You cannot move left beyond the first character in the line. If you try to move beyond the rightmost character, you'll reenter the line. During editing, RETURN is active; pressing it tells MLX to recheck the line. You can press the CLR/HOME key to clear the entire line if you want to start from scratch, or if you want to get to a line number prompt to use RETURN to get back to the menu.

Display Data

The second menu choice, DISPLAY DATA, examines memory and shows the contents in the same format as the program listing (including the checksum). When you press D, MLX asks you for a starting address. Be sure that the starting address you give corresponds to a line number in the listing. Otherwise, the checksum display will be meaningless. MLX displays program lines until it reaches the end of the program, at which point the menu is redisplayed. You can pause the display by pressing the space bar. (MLX finishes printing the current line before halting.) Press space again to restart the display. To break out of the display and get back to the menu before the ending address is reached, press RETURN.

Other Menu Options

Two more menu selections let you save programs and load them back into the computer. These are SAVE FILE and LOAD FILE; their operation is quite straightforward. When you press S or L, MLX asks you for the filename. You'll then be asked to press either D or T to select disk or tape.

You'll notice the disk drive starting and stopping several times during a load or save. Don't panic; this is normal behavior. MLX opens and reads from or writes to the file instead of using the usual LOAD and SAVE commands. Disk users should also note that the drive prefix 0: is automatically added to the filename (line 750), so this should *not* be included when entering the name. This also precludes the use of @ for Save-with-Replace, so remember to give each version you save a different name.

Remember that MLX saves the entire workspace area from the starting address to the ending address, so the save or load may take longer than you might expect if you've entered only a small amount of data from a long listing. When saving a partially completed listing, make sure to note the address where you stopped typing so you'll know where to resume entry when you reload.

MLX reports the standard disk or tape error messages if any problems are detected during the save or load. (Tape users should bear in mind that Commodore computers are never able to detect errors during a save to tape.) MLX also has three special load error messages: INCORRECT STARTING ADDRESS, which means the file you're trying to load does not have the starting address you specified when you ran MLX; LOAD ENDED AT *address*, which means the file you're trying to load ends before the ending address you specified when you started MLX; and TRUNCATED AT ENDING ADDRESS, which means the file you're trying to load extends beyond the ending address you specified when you started MLX. If you see one of these messages and feel certain that you've loaded the right file, exit and rerun MLX, being careful to enter the correct starting and ending addresses.

The QUIT menu option has the obvious effect—it stops MLX and enters BASIC. The RUN/STOP key is disabled, so the Q option lets you exit the program without turning off the computer. (Of course, RUN/STOP–RESTORE also gets you out.) You'll be asked for verification; press Y to exit to BASIC, or any other key to return to the menu. After quitting, you can type RUN again and reenter MLX without losing your data, as long as you don't use the clear workspace option.

The Finished Product

When you've finished typing all the data for an ML program and saved your work, you're ready to see the results. The instructions for loading and using the finished product vary from program to program. Some ML programs are designed to be loaded and run like BASIC programs, so all you need to type is LOAD *"filename"*,8 for disk or LOAD *"filename"* for tape, and then RUN. Such programs will usually have a starting address of 0801. Other programs must be reloaded to specific addresses with a command such as LOAD *"filename"*,8,1 for disk or LOAD *"filename"*,1,1 for tape, then started with a SYS to a particular memory address. The most common starting address for such programs is 49152, which corresponds to MLX address C000. In either case, you should always refer to the article which accompanies the ML listing for information on loading and running the program.

An Ounce Of Prevention

By the time you finish typing in the data for a long ML program, you may have several hours invested in the project. Don't take chances—use our "Automatic Proofreader" to type MLX, and then test your copy *thoroughly* before first using it to enter any significant amount of data. Make sure all the menu options work as they should. Enter fragments of the program starting at several different addresses, then use the Display option to verify that the data has been entered correctly. And be sure to test the Save and Load options several times to ensure that you can recall your work from disk or tape. Don't let a simple typing error in MLX cost you several nights of hard work.

MLX

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

```
EK 100 POKE 56,50:CLR:DIM IN$,
       I,J,A,B,A$,B$,A(7),N$
DM 110 C4=48:C6=16:C7=7:Z2=2:Z
       4=254:Z5=255:Z6=256:Z7=
       127
CJ 120 FA=PEEK(45)+Z6*PEEK(46)
       :BS=PEEK(55)+Z6*PEEK(56
       ):H$="0123456789ABCDEF"
SB 130 R$=CHR$(13):L$="{LEFT}"
       :S$=" ":D$=CHR$(20):Z$=
       CHR$(0):T$="{13 RIGHT}"
CQ 140 SD=54272:FOR I=SD TO SD
       +23:POKE I,0:NEXT:POKE
       {SPACE}SD+24,15:POKE 78
       8,52
FC 150 PRINT"{CLR}"CHR$(142)CH
       R$(8):POKE 53280,15:POK
       E 53281,15
EJ 160 PRINT T$" {RED}{RVS}
       {2 SPACES}{8 @}
       {2 SPACES}"SPC(28)"
       {2 SPACES}{OFF}{BLU} ML
       X II {RED}{RVS}
       {2 SPACES}"SPC(28)"
       {12 SPACES}{BLU}"
FR 170 PRINT"{3 DOWN}
       {3 SPACES}COMPUTE!'S MA
       CHINE LANGUAGE EDITOR
       {3 DOWN}"
JB 180 PRINT"{BLK}STARTING ADD
       RESS{4}";:GOSUB300:SA=A
       D:GOSUB1040:IF F THEN18
       0
GF 190 PRINT"{BLK}{2 SPACES}EN
       DING ADDRESS{4}";:GOSUB
       300:EA=AD:GOSUB1030:IF
       {SPACE}F THEN190
KR 200 INPUT"{3 DOWN}{BLK}CLEA
       R WORKSPACE [Y/N]{4}";A
       $:IF LEFT$(A$,1)<>"Y"TH
       EN220
PG 210 PRINT"{2 DOWN}{BLU}WORK
       ING...";:FORI=BS TO BS+
       EA-SA+7:POKE I,0:NEXT:P
       RINT"DONE"
DR 220 PRINTTAB(10)"{2 DOWN}
       {BLK}{RVS} MLX COMMAND
       {SPACE}MENU {DOWN}{4}":
       PRINT T$"{RVS}E{OFF}NTE
       R DATA"
BD 230 PRINT T$"{RVS}D{OFF}ISP
       LAY DATA":PRINT T$"
       {RVS}L{OFF}OAD FILE"
JS 240 PRINT T$"{RVS}S{OFF}AVE
        FILE":PRINT T$"{RVS}Q
       {OFF}UIT{2 DOWN}{BLK}"
JH 250 GET A$:IF A$=N$ THEN250
HK 260 A=0:FOR I=1 TO 5:IF A$=
       MID$("EDLSQ",I,1)THEN A
       =I:I=5
FD 270 NEXT:ON A GOTO420,610,6
       90,700,280:GOSUB1060:GO
       TO250
EJ 280 PRINT"{RVS} QUIT ":INPU
       T"{DOWN}{4}ARE YOU SURE
        [Y/N]";A$:IF LEFT$(A$,
       1)<>"Y"THEN220
EM 290 POKE SD+24,0:END
JX 300 IN$=N$:AD=0:INPUTIN$:IF
       LEN(IN$)<>4THENRETURN
KF 310 B$=IN$:GOSUB320:AD=A:B$
       =MID$(IN$,3):GOSUB320:A
       D=AD*256+A:RETURN
PP 320 A=0:FOR J=1 TO 2:A$=MID
       $(B$,J,1):B=ASC(A$)-C4+
       (A$>"@")*C7:A=A*C6+B
JA 330 IF B<0 OR B>15 THEN AD=
       0:A=-1:J=2
GX 340 NEXT:RETURN
CH 350 B=INT(A/C6):PRINT MID$(
       H$,B+1,1);:B=A-B*C6:PRI
       NT MID$(H$,B+1,1);:RETU
       RN
RR 360 A=INT(AD/Z6):GOSUB350:A
       =AD-A*Z6:GOSUB350:PRINT
       ":";
BE 370 CK=INT(AD/Z6):CK=AD-Z4*
       CK+Z5*(CK>Z7):GOTO390
PX 380 CK=CK*Z2+Z5*(CK>Z7)+A
JC 390 CK=CK+Z5*(CK>Z5):RETURN
QS 400 PRINT"{DOWN}STARTING AT
       {4}";:GOSUB300:IF IN$<>
       N$ THEN GOSUB1030:IF F
       {SPACE}THEN400
EX 410 RETURN
HD 420 PRINT"{RVS} ENTER DATA
       {SPACE}":GOSUB400:IF IN
       $=N$ THEN220
```

```
JK 43Ø  OPEN3,3:PRINT
SK 44Ø  POKE198,Ø:GOSUB36Ø:IF F
         THEN PRINT IN$:PRINT"
        {UP}{5 RIGHT}";
GC 45Ø  FOR I=Ø TO 24 STEP 3:B$
        =S$:FOR J=1 TO 2:IF F T
        HEN B$=MID$(IN$,I+J,1)
HA 46Ø  PRINT"{RVS}"B$L$;:IF I<
        24THEN PRINT"{OFF}";
HD 47Ø  GET A$:IF A$=N$ THEN47Ø
FK 48Ø  IF(A$>"/"ANDA$<":")OR(A
        $>"@"ANDA$<"G")THEN54Ø
MP 49Ø  IF A$=R$ AND((I=Ø)AND(J
        =1)OR F)THEN PRINT B$;:
        J=2:NEXT:I=24:GOTO55Ø
KC 5ØØ  IF A$="{HOME}" THEN PRI
        NT B$:J=2:NEXT:I=24:NEX
        T:F=Ø:GOTO44Ø
MX 51Ø  IF(A$="{RIGHT}")ANDF TH
        ENPRINT B$L$;:GOTO54Ø
GK 52Ø  IF A$<>L$ AND A$<>D$ OR
        ((I=Ø)AND(J=1))THEN GOS
        UB1Ø6Ø:GOTO47Ø
HG 53Ø  A$=L$+S$+L$:PRINT B$L$;
        :J=2-J:IF J THEN PRINT
        {SPACE}L$;:I=I-3
QS 54Ø  PRINT A$;:NEXT J:PRINT
        {SPACE}S$;
PM 55Ø  NEXT I:PRINT:PRINT"{UP}
        {5 RIGHT}";:INPUT#3,IN$
        :IF IN$=N$ THEN CLOSE3:
        GOTO22Ø
QC 56Ø  FOR I=1 TO 25 STEP3:B$=
        MID$(IN$,I):GOSUB32Ø:IF
         I<25 THEN GOSUB38Ø:A(I
        /3)=A
PK 57Ø  NEXT:IF A<>CK THEN GOSU
        B1Ø6Ø:PRINT"{BLK}{RVS}
        {SPACE}ERROR: REENTER L
        INE {4}":F=1:GOTO44Ø
HJ 58Ø  GOSUB1Ø8Ø:B=BS+AD-SA:FO
        R I=Ø TO 7:POKE B+I,A(I
        ):NEXT
QQ 59Ø  AD=AD+8:IF AD>EA THEN C
        LOSE3:PRINT"{DOWN}{BLU}
        ** END OF ENTRY **{BLK}
        {2 DOWN}":GOTO7ØØ
GQ 6ØØ  F=Ø:GOTO44Ø
QA 61Ø  PRINT"{CLR}{DOWN}{RVS}
        {SPACE}DISPLAY DATA ":G
        OSUB4ØØ:IF IN$=N$ THEN2
        2Ø
RJ 62Ø  PRINT"{DOWN}{BLU}PRESS:
         {RVS}SPACE{OFF} TO PAU
        SE, {RVS}RETURN{OFF} TO
         BREAK{4}{DOWN}"
KS 63Ø  GOSUB36Ø:B=BS+AD-SA:FOR
        I=BTO B+7:A=PEEK(I):GOS
        UB35Ø:GOSUB38Ø:PRINT S$
        ;
CC 64Ø  NEXT:PRINT"{RVS}";:A=CK
        :GOSUB35Ø:PRINT
KH 65Ø  F=1:AD=AD+8:IF AD>EA TH
        ENPRINT"{DOWN}{BLU}** E
        ND OF DATA **":GOTO22Ø
KC 66Ø  GET A$:IF A$=R$ THEN GO
        SUB1Ø8Ø:GOTO22Ø
EQ 67Ø  IF A$=S$ THEN F=F+1:GOS
        UB1Ø8Ø
AD 68Ø  ONFGOTO63Ø,66Ø,63Ø
CM 69Ø  PRINT"{DOWN}{RVS} LOAD
        {SPACE}DATA ":OP=1:GOTO
        71Ø
PC 7ØØ  PRINT"{DOWN}{RVS} SAVE
        {SPACE}FILE ":OP=Ø
RX 71Ø  IN$=N$:INPUT"{DOWN}FILE
        NAME{4}";IN$:IF IN$=N$
        {SPACE}THEN22Ø
PR 72Ø  F=Ø:PRINT"{DOWN}{BLK}
        {RVS}T{OFF}APE OR {RVS}
        D{OFF}ISK: {4}";
FP 73Ø  GET A$:IF A$="T"THEN PR
        INT"T{DOWN}":GOTO88Ø
HQ 74Ø  IF A$<>"D"THEN73Ø
HH 75Ø  PRINT"D{DOWN}":OPEN15,8
        ,15,"IØ:":B=EA-SA:IN$="
        Ø:"+IN$:IF OP THEN81Ø
SQ 76Ø  OPEN 1,8,8,IN$+",P,W":G
        OSUB86Ø:IF A THEN22Ø
FJ 77Ø  AH=INT(SA/256):AL=SA-(A
        H*256):PRINT#1,CHR$(AL)
        ;CHR$(AH);
PE 78Ø  FOR I=Ø TO B:PRINT#1,CH
        R$(PEEK(BS+I));:IF ST T
        HEN8ØØ
FC 79Ø  NEXT:CLOSE1:CLOSE15:GOT
        O94Ø
GS 8ØØ  GOSUB1Ø6Ø:PRINT"{DOWN}
        {BLK}ERROR DURING SAVE:
        {4}":GOSUB86Ø:GOTO22Ø
MA 81Ø  OPEN 1,8,8,IN$+",P,R":G
        OSUB86Ø:IF A THEN22Ø
GE 82Ø  GET#1,A$,B$:AD=ASC(A$+Z
        $)+256*ASC(B$+Z$):IF AD
        <>SA THEN F=1:GOTO85Ø
RX 83Ø  FOR I=Ø TO B:GET#1,A$:P
        OKE BS+I,ASC(A$+Z$):IF(
        I<>B)AND ST THEN F=2:AD
        =I:I=B
FA 84Ø  NEXT:IF ST<>64 THEN F=3
FQ 85Ø  CLOSE1:CLOSE15:ON ABS(F
        >Ø)+1 GOTO96Ø,97Ø
SA 86Ø  INPUT#15,A,A$:IF A THEN
         CLOSE1:CLOSE15:GOSUB1Ø
        6Ø:PRINT"{RVS}ERROR: "A
        $
GQ 87Ø  RETURN
EJ 88Ø  POKE183,PEEK(FA+2):POKE
        187,PEEK(FA+3):POKE188,
        PEEK(FA+4):IFOP=ØTHEN92
        Ø
HJ 89Ø  SYS 63466:IF(PEEK(783)A
        ND1)THEN GOSUB1Ø6Ø:PRIN
        T"{DOWN}{RVS} FILE NOT
        {SPACE}FOUND ":GOTO69Ø
CS 9ØØ  AD=PEEK(829)+256*PEEK(8
        3Ø):IF AD<>SA THEN F=1:
        GOTO97Ø
SC 91Ø  A=PEEK(831)+256*PEEK(83
        2)-1:F=F-2*(A<EA)-3*(A>
        EA):AD=A-AD:GOTO93Ø
KM 92Ø  A=SA:B=EA+1:GOSUB1Ø1Ø:P
        OKE78Ø,3:SYS 63338
JF 93Ø  A=BS:B=BS+(EA-SA)+1:GOS
        UB1Ø1Ø:ON OP GOTO95Ø:SY
        S 63591
AE 94Ø  GOSUB1Ø8Ø:PRINT"{BLU}**
         SAVE COMPLETED **":GOT
        O22Ø
XP 95Ø  POKE147,Ø:SYS 63562:IF
        {SPACE}ST>Ø THEN97Ø
FR 96Ø  GOSUB1Ø8Ø:PRINT"{BLU}**
         LOAD COMPLETED **":GOT
        O22Ø
DP 97Ø  GOSUB1Ø6Ø:PRINT"{BLK}
        {RVS}ERROR DURING LOAD:
        {DOWN}{4}":ON F GOSUB98
        Ø,99Ø,1ØØØ:GOTO22Ø
PP 98Ø  PRINT"INCORRECT STARTIN
        G ADDRESS (";:GOSUB36Ø:
        PRINT")":RETURN
GR 99Ø  PRINT"LOAD ENDED AT ";:
        AD=SA+AD:GOSUB36Ø:PRINT
         D$:RETURN
FD 1ØØØ PRINT"TRUNCATED AT END
         ING ADDRESS":RETURN
RX 1Ø1Ø AH=INT(A/256):AL=A-(AH
        *256):POKE193,AL:POKE1
        94,AH
FF 1Ø2Ø AH=INT(B/256):AL=B-(AH
        *256):POKE174,AL:POKE1
        75,AH:RETURN
FX 1Ø3Ø IF AD<SA OR AD>EA THEN
        1Ø5Ø
HA 1Ø4Ø IF(AD>511 AND AD<4Ø96Ø
        )OR(AD>49151 AND AD<53
        248)THEN GOSUB1Ø8Ø:F=Ø
        :RETURN
HC 1Ø5Ø GOSUB1Ø6Ø:PRINT"{RVS}
        {SPACE}INVALID ADDRESS
         {DOWN}{BLK}":F=1:RETU
        RN
AR 1Ø6Ø POKE SD+5,31:POKE SD+6
        ,2Ø8:POKE SD,24Ø:POKE
        {SPACE}SD+1,4:POKE SD+
        4,33
DX 1Ø7Ø FOR S=1 TO 1ØØ:NEXT:GO
        TO1Ø9Ø
PF 1Ø8Ø POKE SD+5,8:POKE SD+6,
        24Ø:POKE SD,Ø:POKE SD+
        1,9Ø:POKE SD+4,17
AC 1Ø9Ø FOR S=1 TO 1ØØ:NEXT:PO
        KE SD+4,Ø:POKE SD,Ø:PO
        KE SD+1,Ø:RETURN
```

©

New Products From Batteries Included

An extensive array of new software programs for Apple, Atari, Commodore, and IBM computers has been announced by Batteries Included. The list includes the following:

*I*S Talk* is a full-featured GEM-based telecommunications program for the Atari ST and the IBM PC. The program is part of the company's Integral Solutions line of GEM-based products. *I*S Talk* features a built-in 50,000-word spelling checker, three levels of automation functions, keystroke macros, and an unlimited capture buffer including X-modem file transfers. Other features include Replay, and a unique session recorder. A custom keyboard interface is built in, allowing you to use the mouse or function keys and keyboard.

Suggested retail prices are $79.95 for the ST version and $99.95 for the IBM version.

The popular *HomePak* three-in-one word processor, database manager, and telecommunications package should be available by the time you read this in versions for the Apple Macintosh, the Atari ST, and the Commodore 128 computers. *HomePak* sells for $49.95, and is already available for the Commodore 64, Apple II, IBM PC/PCjr, and Atari XL/XE computers.

The *PaperClip II* word processor for the Commodore 128 is designed to take advantage of the 128's memory, speed, and power. The program includes a built-in telecommunications module for use with online services, a 20,000-word spell checker, macro capability, multiple columns, reverse video scroll, word wrap, chaptering, and an expanded maximum document size. *PaperClip II*, priced at $79.95, is completely compatible with *PaperClip* text files on the Commodore 64. Also new (or planned for release soon) are *PaperClip With SpellPack* for the Atari 130XE ($59.95); *PaperClip* for the Apple II, II+, IIc, and IIe computers ($59.95); and *I*S PaperClip Elite* for the IBM PC and compatibles ($129.95), Atari ST ($79.95), and Commodore Amiga ($129.95). *PaperClip Elite* includes all of the earlier *PaperClip* features, plus advanced functions such as a real-time spelling checker, idea processing, independent linked windows, integrated text and graphics, and other capabilities.

*I*S Degas Elite*, an upgraded version of the original *Degas* graphics and design program for the Atari ST, will be available soon for the IBM PC and compatibles, Amiga, and ST. The program includes everything in the original, plus features like FLIP, SCALE, ROTATE, and the ability to cut and paste between pictures on multiple work screens. *DEGAS Elite* files can also be integrated with *PaperClip Elite* text files, and all *Degas Elite* files are compatible with the original *Degas*.

Batteries Included will also be marketing *The Isgur Portfolio System (IPS)* for the IBM PC ($249.95), the Atari ST ($199.95), the Amiga ($249.95), and the Apple Macintosh ($249.95). This is an investment management program designed by Lee Isgur, Wall Street analyst and first vice president of Paine Webber in New York. *IPS* lets you update your portfolios automatically from online services, and provides a variety of analytical tools for more profitable decision making.

BTS The Spreadsheet is a $69.95 spreadsheet for the Macintosh, Atari ST, and Amiga that includes key math, statistical, and financial functions, logical operators, formatting enhancements, and other features. Maximum worksheet size is 1000 rows by 1000 columns. Desk accessory version is included on the same Macintosh and ST disks.

TimeLink is an electronic diary program for planning and record-keeping, available for the Macintosh and Atari ST for $49.95 each. *I*S Time And Billing* is a professional office administration program planned for the fourth quarter of 1986 for IBM and Atari ST computers. *I*S Consultant*, also planned for the fourth quarter, is an enhanced Atari ST version of the original *Consultant* database. The new program will use the GEM interface and contain additional features. *B/GRAPH Elite* (a fourth quarter release) is a $69.95 graphics/charting and statistical analysis package for the Atari ST.

Soon to be available is *Thunder*, a $39.95 writer's accessory that can be called for use from within any GEM-based Atari ST application. Features include a 50,000-word spelling checker with three different error-catching modes and an immediate word replacement capability. An Abbreviation function will even supply you with the remainder of the word, or words, you want when you type in just the first few letters. And a built-in report analyzer gives you details on everything from word count to the document's relative readability.

Batteries Included, 30 Mural St., Richmond Hill, Ontario, Canada L4B 1B5.
Circle Reader Service Number 200.

Abacus Software, Book

Abacus Software has introduced several Atari ST software applications packages, each priced at $39.95. *ST TextPro* is a word processor that features multicolumn output, automatic indexing and table of contents, scrolling, definable function keys, sideways output (to Epson printers), and printer drivers for other printers. The program allows full-screen editing with either mouse or keyboard commands.

ST Text Designer is a pagemaking package for creating layouts from word processing files. The program reads text files from *ST TextPro* and other ASCII word processors, and allows block operations. Graphics can be merged into the layout, and borders and lines can be added. Output is to Epson-compatible printers.

ST DataPro is a data management package that lets you input data through screen templates. Record length is unlimited, with a maximum of 64,000 records available. The package also supports a RAM disk as well as a floppy disk. *ST Forth/MT* is a multitasking implementation of the Forth programming language, based on Forth 83. The program supports 32-bit arithmetic, has more than 1500 commands, and includes a full-screen editor, Forth macro assembler, and monitor and disk monitor.

Abacus is also announcing *ST PaintPro*, a drawing program, and *ST AssemPro*, an assembly language development package.

For the Commodore 128 computer, Abacus recently released another volume in its Commodore 128 Reference Library. The $19.95 title is *Commodore 1571 Internals.*

Abacus Software, P.O. Box 7211, Grand Rapids, MI 49510.

Circle Reader Service Number 201.

64 Software From Firebird

Firebird has released several new software packages for the Commodore 64 and 128.

The Music System is a comprehensive music development package that includes full sonic tailoring of the computer's SID chip, multi-voicing, mono or polyphonic modes, editing, recording, and storage and playback of sound settings and compositions.

There is an advanced version of *The Music System* which includes the features of the standard version and adds MIDI capabilities (compatible with either S.I.E.L. or Passport Designs MIDI interfaces). The advanced version enables the user to link and edit sequences, control six MIDI tracks or devices simultaneously, print sheet music, and also permits automatic transpositions and automatic tempo conforming and correcting. You can upgrade the standard version to the advanced version at a nominal cost.

Gerry The Germ and *Microcosm* are the newest in the Firebird Super Silver Disk Series. Both games feature music, sound effects, graphics, and animation. *Gerry The Germ* is a journey through the human body with Gerry as the guide. In *Microcosm,* the player must defend the agricultural cargo of a crippled interstellar freightliner against a hoard of mutant insects. The games are on one *flippy* disk and require either joystick or keyboard control.

The standard version of *The Music System* has a suggested retail price of $39.95 and the advanced version is $79.95. The *Gerry The Germ/Microcosm* Super Silver Disk Series package retails for $19.95.

Firebird, P.O. Box 49, Ramsey, NJ 07446.

Circle Reader Service Number 202.

New Games From Intellicreations

Intellicreations (formerly Datasoft) has announced the release of three new games.

The Never Ending Story is an adventure/fantasy game based on the book and film of the same name. Cast as the hero, Atreyu, and aided by Falkor, the luck dragon, you face the trials and terrors of the ever-consuming "nothing" in an illustrated and imaginative computer adventure. *The Never Ending Story* is now available as a *flippy* for Commodore and Atari 8-bit machines (Commodore version on one side, Atari on the flip side), and for the Apple II computers. All versions retail for $29.95.

Mind Pursuit is a test of intelligence, knowledge, and trivia with three levels of difficulty. At the simplest level are true/false questions; next are multiple choice; and finally, and most difficult, are fill-in-the-blanks. The game is designed with questions for both adults and children, and has three difficulty levels, so the whole family can play together. Music and graphics clues are also used for variety, and provide further challenges to the game play. Additional game disks are available. Commodore and Apple versions retail for $29.95; add-on disks are $14.95.

221B Baker Street is the address of fictional super-sleuth Sherlock Holmes. In the game *221B Baker Street,* you start at that address and travel through the streets and alleys of London, gathering clues that will lead to the solution of some of the most intriguing cases ever faced by Holmes and his assistant, Dr. Watson. The initial game will include 30 different cases; two additional game disks will be available later, each containing 30 more cases. Atari, Commodore, and Apple versions retail for $29.95, with follow-up disks costing $14.95 each.

Intellicreations, 19808 Nordhoff Pl., Chatsworth, CA 91311.

Circle Reader Service Number 203.

128 Telecommunications And 64 Graphics

Progressive Peripherals & Software has announced two new products for the Commodore 64 and 128.

For the Commodore 128, *BobsTerm Pro-128* is a menu-driven communications package that lets you edit files while it reads, writes, uploads, and downloads to any disk type (including CP/M). The package supports VT-100 and VT-52 80 ADM-31 (CP/M type) terminal emulation, and includes a full-screen text editor, on-screen status display, remote mode, macro and answer-back string functions, and a manual. The program is compatible with most modems and works with high speed 1571 disk drives, the SFD-1001, and Commodore and MSD dual drives.

With *Picasso's Revenge,* you can draw circles, squares, triangles, and many other geometric figures with the Commodore 64. This graphics package has 35 predefined textures, five-level focus, and a ZOOM command that magnifies the screen area eight times. The program supports high-resolution drawings made with Koala, Suncom, Paint Majic, and other commonly used graphics programs. It's compatible with most dot-matrix printers and prints in nine shades of gray. The *Picasso's Revenge* package includes a free light pen and a user's manual.

BobsTerm Pro-128 retails for $79.95 and *Picasso's Revenge* for $59.95.

Progressive Peripherals & Software, 464 Kalamath St., Denver, CO 80204.

Circle Reader Service Number 204.

Amiga Fonts

Classic Concepts FUTUREWARE has announced new type fonts for Amiga users who need larger fonts for video titling, graphics, desktop publishing, and other applications. *FUTUREWARE FONTS* can be used to mix titles with video images (with Genlock) and are more clearly legible in high resolution mode than fonts with the *Workbench* disk. The fonts are compatible with *Notepad, Deluxe Paint, Aegis Images,* and other common Amiga software

Each package includes a disk utility installation program, 13 new fonts, and a font reference booklet.

Retail price is $14.95, plus $1 shipping and handling.

FUTUREWARE FONTS, P.O. Box 94276, Richmond, B.C., Canada V6Y 2A6.

Circle Reader Service Number 205.

Apple II Science Software

Science Toolkit—Master Module lets you perform real experiments with your Apple computer and two sensory probes. Each package contains an interface box, light probe, temperature probe, light probe stand, light guard, two lab notebook labels, and a 125-page *User's Manual & Experiment Guide* that details how to use the probes and the on-screen instruments—thermometer, light meter, timer, and strip chart.

Science Toolkit—Master Module is for the Apple IIe and IIc with 64K memory. Printer is optional and the interface box is included. Users should be 12 years old and up.

There is a school version which includes a teacher's guide in addition to the material provided in the standard package. For use in grades 4–12.

The standard version retails for $69.95 and the school version retails for $89.95. Anyone with the standard version can buy the teacher's guide separately for $20.

Brøderbund Software, Inc., 17 Paul Dr., San Rafael, CA 94903-2101.

Circle Reader Service Number 206.

©

From the publishers of *COMPUTE!*

July 1986 *COMPUTE!* Disk

All the exciting programs from the past three issues of *COMPUTE!* are on timesaving, error-free floppy disks that are ready to load on your IBM PC and PCjr or Commodore 64 and 128. The July 1986 *COMPUTE! Disks* contain the entertaining and useful Commodore or IBM programs from the May, June, and July 1986 issues of *COMPUTE!*.

The July 1986 *COMPUTE! Disk* costs $12.95 plus $2.00 shipping and handling and is available only from COMPUTE! Publications. Please specify whether you need a Commodore or IBM disk.

For added savings and convenience, you may also subscribe to the *COMPUTE! Disk.* At a cost of only $39.95 a year (a $12.00 savings), you'll receive four disks, one every three months. Each disk will contain all the programs for your machine from the previous three issues of *COMPUTE!*.

This is an excellent way to build your software library while you enjoy the quality programs from *COMPUTE!*.

Disks and subscriptions are available for Apple, Atari, Commodore 64 and 128, and IBM personal computers. Call for details.

For more information or to order the July 1986 *COMPUTE! Disk*, call toll free 1-800-346-6767 (in NY 212-265-8360) or write *COMPUTE! Disk*, P.O. Box 5038, F.D.R. Station, New York, NY 10150.

COMPUTE!™ Publications, Inc. abc
Part of ABC Consumer Magazines, Inc.
One of the ABC Publishing Companies
825 7th Avenue, 6th Floor, New York, NY 10019
Publishers of COMPUTE!, COMPUTE!'s Gazette, COMPUTE!'s Gazette Disk, COMPUTE! Books, and COMPUTE!'s Apple Applications

Atari COMPUTE! DISK

The first quarterly COMPUTE! DISK for Atari, containing the programs from the January–March 1986 issues, has caused some confusion among those who received it. First, the disk *does not* contain a disk operating system (DOS) and therefore cannot be used to start up the computer. As the disk label instructs, you must boot your system using your own DOS disk, make sure you are in BASIC, then put the COMPUTE! DISK in the drive and type RUN "D:MENU". The disk is in Atari DOS 2.0S format, which can also be read by Atari DOS 2.5 and most third-party operating systems such as DOS XL and OS/A+. Atari DOS 3.0 has an option to convert DOS 2.0 files to 3.0 format. However, if you're still using DOS 3.0 you should seriously consider switching to DOS 2.5 (despite the confusing numbering, version 2.5 is newer than 3.0). DOS 2.5 is available free from many Atari dealers and user groups. It is not possible to write DOS files to the COMPUTE! DISK because it is write protected and because there isn't enough room on the disk for any more files.

Note, however, that while the COMPUTE! DISK is write protected, it is not copy protected. When you receive the disk, you should first make a backup copy. (With Atari DOS 2.0 or 2.5, use option J—Duplicate Disk). Programs such as *SpeedScript* and *SpeedCalc* will be more useful if copied to a disk that contains the DOS system files. (With Atari DOS 2.0 or 2.5, use option O—Duplicate File).

There is one program on the January–March Atari disk that cannot be used as-is with Atari DOS. "High Rise" was designed to be a boot disk file if typed in from the magazine using MLX. The verison on the disk was intended to be executed as a binary file, but we failed to explain that and also failed to put the program on the disk in the proper format for the Atari DOS L (Load Binary File) option. The problem is easy to correct, though. Copy the HIRISE.FEB program from the COMPUTE! DISK to another disk (preferably one that contains DOS) using the O option in Atari DOS 2.0/2.5, then enter and run the following one-line program:

```
10 OPEN #1,9,0,"D:HIRISE.
   FEB":PUT #1,224:PUT #1
   ,2:PUT #1,225:PUT #1,2
   :PUT #1,0:PUT #1,48:CL
   OSE #1
```

This fixes the file so the game can be played by going to the Atari DOS 2.0/2.5 menu and using the L option. Specify HIRISE.FEB as the file to load; it begins running automatically after loading. You could also use the DOS menu's E option to rename the fixed file to AUTORUN.SYS. In that case, the program will load and run automatically when the disk is booted. If you're using OS/A+ DOS or DOS XL, you can run High Rise simply by typing HIRISE.FEB at a DOS D1: prompt, so the above correction is not necessary.

The January–March Atari disk does not contain the programs "MLX" and "Automatic Proofreader" because the disk was too full for them to be included. They appear on the April–June Atari disk and should be on all future Atari disks when space allows. We know of no special problems with any of the programs on the April–June disk.

64 Autobooter

The automatic loading technique used in this program from the May issue (p. 92) should work fine for the first program autobooted after the computer is turned on. However, when autobooting more than one program in a session, the routine may fail if a later autoboot attempts to load a program shorter than the one previously autobooted. To correct this, change the following lines:

```
GA 140 IF S<>51988 THEN PRINT:
       PRINT"MISTYPED DATA":EN
       D
DF 200 IF B>7936 THEN PRINT CH
       R$(18);STR$(B-7936)"
       {2 SPACES}BYTE OVERFLOW
        ":GOTO 160
CG 550 DATA 57,224,32,176,12,1
       89,16,255
BE 710 DATA 104,208,11,32,51,1
       65,162,54
AS 720 DATA 160,2,134,122,132,
       123,162,31
EF 730 DATA 169,0,157,16,245,2
       02,16,250
RM 740 DATA 169,13,76,210,255,
       138,0,0
```

Minding IBM Memory

Line 100 in Program 3 (p. 86) of this memory management utility in the June issue contains an error that prevents the deallocation routine from properly releasing allocated blocks of memory. The &h85 in that line should be &h8b, as shown in the assembly listing for the deallocation routine (Program 2).

Using PALETTE USING

In Program 2 from this article in the May issue (p. 71), the AS$="N" at the end of line 80 should instead be AN$="N".

Commodore Loading and Linking, Part 4

In the example program in the lower-right corner of page 75 in the June issue, omit the question mark in the value 128 in line 140. The question mark was caused by a printer malfunction. ©

COMPUTE!'s Guide To Typing In Programs

Computers are precise—type the program *exactly* as listed, including necessary punctuation and symbols, except for special characters noted below. We have provided a special listing convention as well as a program to check your typing—"The Automatic Proofreader."

Programs for the IBM, TI-99/4A, and Atari ST models should be typed exactly as listed; no special characters are used. Programs for Commodore, Apple, and Atari 400/800/XL/XE computers may contain some hard-to-read special characters, so we have a listing system that indicates these control characters. You will find these Commodore and Atari characters in curly braces; *do not type the braces.* For example, {CLEAR} or {CLR} instructs you to insert the symbol which clears the screen on the Atari or Commodore machines. A complete list of these symbols is shown in the tables below. For Commodore, Apple, and Atari, a single symbol by itself within curly braces is usually a control key or graphics key. If you see {A}, hold down the CONTROL key and press A. This will produce a reverse video character on the Commodore (in quote mode), a graphics character on the Atari, and an invisible control character on the Apple.

Graphics characters entered with the Commodore logo key are enclosed in a special bracket: [<A>]. In this case, you would hold down the Commodore logo key as you type A. Our Commodore listings are in uppercase, so shifted symbols are underlined. A graphics heart symbol (SHIFT-S) would be listed as S̲. One exception is {SHIFT-SPACE}. When you see this, hold down SHIFT and press the space bar. If a number precedes a symbol, such as {5 RIGHT}, {6 S̲}, or [<8 Q>], you would enter five cursor rights, six shifted S's, or eight Commodore-Q's. On the Atari, inverse characters (white on black) should be entered with the inverse video

Atari 400/800/XL/XE

When you see	Type	See	
{CLEAR}	ESC SHIFT <		Clear Screen
{UP}	ESC CTRL -	↑	Cursor Up
{DOWN}	ESC CTRL =	↓	Cursor Down
{LEFT}	ESC CTRL +	←	Cursor Left
{RIGHT}	ESC CTRL *	→	Cursor Right
{BACK S}	ESC DELETE		Backspace
{DELETE}	ESC CTRL DELETE		Delete character
{INSERT}	ESC CTRL INSERT		Insert character
{DEL LINE}	ESC SHIFT DELETE		Delete line
{INS LINE}	ESC SHIFT INSERT		Insert line
{TAB}	ESC TAB		TAB key
{CLR TAB}	ESC CTRL TAB		Clear tab
{SET TAB}	ESC SHIFT TAB		Set tab stop
{BELL}	ESC CTRL 2		Ring buzzer
{ESC}	ESC ESC		ESCape key

Commodore PET/CBM/VIC/64/128/16/+4

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

```
180 IF VAL(LEFT$(L$,2))=Ø AND
   MID$(L$,3,1)=" " THEN L$=M
   ID$(L$,4)
200 IF ASC(L$)>57 THEN 260 'no
    line number, therefore co
   mmand
205 BL=INSTR(L$," "):IF BL=Ø T
   HEN BL$=L$:GOTO 206 ELSE B
   L$=LEFT$(L$,BL-1)
206 LNUM=VAL(BL$):TEXT$=MID$(L
   $,LEN(STR$(LNUM))+1)
210 IF TEXT$="" THEN GOSUB 54Ø
   :IF LNUM=LNUM(P) THEN GOSU
   B 56Ø:GOTO 15Ø ELSE 15Ø
220 CKSUM=Ø:FOR I=1 TO LEN(L$)
   :CKSUM=(CKSUM+ASC(MID$(L$,
   I))*I) AND 255:NEXT:LOCATE
    Y,1:PRINT CHR$(65+CKSUM/1
   6)+CHR$(65+(CKSUM AND 15))
   +" "+L$
230 GOSUB 54Ø:IF LNUM(P)=LNUM
   THEN L$(P)=TEXT$:GOTO 15Ø
   'replace line
240 GOSUB 58Ø:GOTO 15Ø 'insert
    the line
260 TEXT$="":FOR I=1 TO LEN(L$
   ):A=ASC(MID$(L$,I)):TEXT$=
   TEXT$+CHR$(A+32*(A>96 AND
   A<123)):NEXT
270 DELIMITER=INSTR(TEXT$," ")
   :COMMAND$=TEXT$:ARG$="":IF
    DELIMITER THEN COMMAND$=L
   EFT$(TEXT$,DELIMITER-1):AR
   G$=MID$(TEXT$,DELIMITER+1)
    ELSE DELIMITER=INSTR(TEXT
   $,CHR$(34)):IF DELIMITER T
   HEN COMMAND$=LEFT$(TEXT$,D
   ELIMITER-1):ARG$=MID$(TEXT
   $,DELIMITER)
280 IF COMMAND$<>"LIST" THEN 4
   1Ø
290 OPEN "scrn:" FOR OUTPUT AS
    #1
300 IF ARG$="" THEN FIRST=Ø:P=
   MAX-1:GOTO 34Ø
310 DELIMITER=INSTR(ARG$,"-"):
   IF DELIMITER=Ø THEN LNUM=V
   AL(ARG$):GOSUB 54Ø:FIRST=P
   :GOTO 34Ø
320 FIRST=VAL(LEFT$(ARG$,DELIM
   ITER)):LAST=VAL(MID$(ARG$,
   DELIMITER+1))
330 LNUM=FIRST:GOSUB 54Ø:FIRST
   =P:LNUM=LAST:GOSUB 54Ø:IF
   P=Ø THEN P=MAX-1
340 FOR X=FIRST TO P:N$=MID$(S
   TR$(LNUM(X)),2)+" "
350 IF CKFLAG=Ø THEN A$="":GOT
   O 37Ø
360 CKSUM=Ø:A$=N$+L$(X):FOR I=
   1 TO LEN(A$):CKSUM=(CKSUM+
   ASC(MID$(A$,I))*I) AND 255
   :NEXT:A$=CHR$(65+CKSUM/16)
   +CHR$(65+(CKSUM AND 15))+"
   "
370 PRINT #1,A$+N$+L$(X)
380 IF INKEY$<>"" THEN X=P
390 NEXT :CLOSE #1:CKFLAG=Ø
400 GOTO 13Ø
410 IF COMMAND$="LLIST" THEN O
   PEN "lpt1:" FOR OUTPUT AS
   #1:GOTO 3ØØ
420 IF COMMAND$="CHECK" THEN C
   KFLAG=1:GOTO 29Ø
430 IF COMMAND$<>"SAVE" THEN 4
   5Ø
440 GOSUB 6ØØ:OPEN ARG$ FOR OU
   TPUT AS #1:ARG$="":GOTO 3Ø
   Ø
450 IF COMMAND$<>"LOAD" THEN 4
   9Ø
460 GOSUB 6ØØ:OPEN ARG$ FOR IN
   PUT AS #1:MAX=Ø:P=Ø
470 WHILE NOT EOF(1):LINE INPU
   T #1,L$:BL=INSTR(L$," "):B
   L$=LEFT$(L$,BL-1):LNUM(P)=
   VAL(BL$):L$(P)=MID$(L$,LEN
   (STR$(VAL(BL$)))+1):P=P+1:
   WEND
480 MAX=P:CLOSE #1:GOTO 13Ø
490 IF COMMAND$="NEW" THEN INP
   UT "Erase program - Are yo
   u sure";L$:IF LEFT$(L$,1)=
   "y" OR LEFT$(L$,1)="Y" THE
   N MAX=Ø:LNUM(Ø)=65536!:GOT
   O 13Ø:ELSE 13Ø
500 IF COMMAND$="BASIC" THEN C
   OLOR 7,Ø,Ø:ON ERROR GOTO Ø
   :CLS:END
510 IF COMMAND$<>"FILES" THEN
   52Ø
515 IF ARG$="" THEN ARG$="A:"
   ELSE SEL=1:GOSUB 6ØØ
517 FILES ARG$:GOTO 13Ø
520 PRINT"Syntax error":GOTO 1
   3Ø
540 P=Ø:WHILE LNUM>LNUM(P) AND
    P<MAX:P=P+1:WEND:RETURN
560 MAX=MAX-1:FOR X=P TO MAX:L
   NUM(X)=LNUM(X+1):L$(X)=L$(
   X+1):NEXT:RETURN
580 MAX=MAX+1:FOR X=MAX TO P+1
    STEP -1:LNUM(X)=LNUM(X-1)
   :L$(X)=L$(X-1):NEXT:L$(P)=
   TEXT$:LNUM(P)=LNUM:RETURN
600 IF LEFT$(ARG$,1)<>CHR$(34)
    THEN 52Ø ELSE ARG$=MID$(A
   RG$,2)
610 IF RIGHT$(ARG$,1)=CHR$(34)
    THEN ARG$=LEFT$(ARG$,LEN(
   ARG$)-1)
620 IF SEL=Ø AND INSTR(ARG$,".
   ")=Ø THEN ARG$=ARG$+".BAS"
630 SEL=Ø:RETURN
640 CLOSE #1:CKFLAG=Ø:PRINT"St
   opped.":RETURN 15Ø
650 PRINT "Error #";ERR:RESUME
    15Ø
```

Program 3: Commodore Proofreader

By Philip Nelson, Assistant Editor

```
10 VEC=PEEK(772)+256*PEEK(773)
   :LO=43:HI=44
20 PRINT "AUTOMATIC PROOFREADE
   R FOR ";:IF VEC=42364 THEN
   {SPACE}PRINT "C-64"
30 IF VEC=50556 THEN PRINT "VI
   C-20"
40 IF VEC=35158 THEN GRAPHIC C
   LR:PRINT "PLUS/4 & 16"
50 IF VEC=17165 THEN LO=45:HI=
   46:GRAPHIC CLR:PRINT"128"
60 SA=(PEEK(LO)+256*PEEK(HI))+
   6:ADR=SA
70 FOR J=0 TO 166:READ BYT:POK
   E ADR,BYT:ADR=ADR+1:CHK=CHK
   +BYT:NEXT
80 IF CHK<>20570 THEN PRINT "*
   ERROR* CHECK TYPING IN DATA
   STATEMENTS":END
90 FOR J=1 TO 5:READ RF,LF,HF:
   RS=SA+RF:HB=INT(RS/256):LB=
   RS-(256*HB)
100 CHK=CHK+RF+LF+HF:POKE SA+L
   F,LB:POKE SA+HF,HB:NEXT
110 IF CHK<>22054 THEN PRINT "
   *ERROR* RELOAD PROGRAM AND
   {SPACE}CHECK FINAL LINE":EN
   D
120 POKE SA+149,PEEK(772):POKE
    SA+150,PEEK(773)
130 IF VEC=17165 THEN POKE SA+
    14,22:POKE SA+18,23:POKESA+
    29,224:POKESA+139,224
140 PRINT CHR$(147);CHR$(17);"
    PROOFREADER ACTIVE":SYS SA
150 POKE HI,PEEK(HI)+1:POKE (P
    EEK(LO)+256*PEEK(HI))-1,0:N
    EW
160 DATA 120,169,73,141,4,3,16
    9,3,141,5,3
170 DATA 88,96,165,20,133,167,
    165,21,133,168,169
180 DATA 0,141,0,255,162,31,18
    1,199,157,227,3
190 DATA 202,16,248,169,19,32,
    210,255,169,18,32
200 DATA 210,255,160,0,132,180
    ,132,176,136,230,180
210 DATA 200,185,0,2,240,46,20
    1,34,208,8,72
220 DATA 165,176,73,255,133,17
    6,104,72,201,32,208
230 DATA 7,165,176,208,3,104,2
    08,226,104,166,180
240 DATA 24,165,167,121,0,2,13
    3,167,165,168,105
250 DATA 0,133,168,202,208,239
    ,240,202,165,167,69
260 DATA 168,72,41,15,168,185,
    211,3,32,210,255
270 DATA 104,74,74,74,74,168,1
    85,211,3,32,210
280 DATA 255,162,31,189,227,3,
    149,199,202,16,248
290 DATA 169,146,32,210,255,76
    ,86,137,65,66,67
300 DATA 68,69,70,71,72,74,75,
    77,80,81,82,83,88
310 DATA 13,2,7,167,31,32,151,
    116,117,151,128,129,167,136
    ,137
```

Program 4: Apple Proofreader

By Tim Victor, Editorial Programmer

```
10 C = Ø: FOR I = 768 TO 768 +
   68: READ A:C = C + A: POKE I
   ,A: NEXT
20 IF C < > 7258 THEN PRINT "ER
   ROR IN PROOFREADER DATA STAT
   EMENTS": END
30 IF PEEK (190 * 256) < > 76 T
   HEN POKE 56,Ø: POKE 57,3: CA
   LL 1002: GOTO 5Ø
40 PRINT CHR$ (4);"IN#A$3ØØ"
50 POKE 34,Ø: HOME : POKE 34,1:
    VTAB 2: PRINT "PROOFREADER
    INSTALLED"
60 NEW
100 DATA 216,32,27,253,201,141
110 DATA 208,6Ø,138,72,169,Ø
120 DATA 72,189,255,1,201,16Ø
130 DATA 24Ø,8,1Ø4,1Ø,125,255
140 DATA 1,1Ø5,Ø,72,2Ø2,2Ø8
150 DATA 238,1Ø4,17Ø,41,15,9
160 DATA 48,2Ø1,58,144,2,233
170 DATA 57,141,1,4,138,74
180 DATA 74,74,74,41,15,9
190 DATA 48,2Ø1,58,144,2,233
200 DATA 57,141,Ø,4,1Ø4,17Ø
210 DATA 169,141,96
```

key (Atari logo key on 400/800 models).

Whenever more than two spaces appear in a row, they are listed in a special format. For example, {6 SPACES} means press the space bar six times. Our Commodore listings never leave a single space at the end of a line, instead moving it to the next printed line as {SPACE}.

Amiga program listings contain only one special character, the left arrow (←) symbol. This character marks the end of each program line. Wherever you see a left arrow, press RETURN or move the cursor off the line to enter that line into memory. Don't try to type in the left arrow symbol; it's there only as a marker to indicate where each program line ends.

The Automatic Proofreader

Type in the appropriate program listed below, then save it for future use. The Commodore Proofreader works on the Commodore 128, 64, Plus/4, 16, and VIC-20. Don't omit any lines, even if they contain unfamiliar commands or you think they don't apply to your computer. When you run the program, it installs a machine language program in memory and erases its BASIC portion automatically (so be sure to save several copies before running the program for the first time). If you're using a Commodore 128, Plus/4 or 16, do *not* use any GRAPHIC commands while the Proofreader is active. You should disable the Commodore Proofreader before running any other program. To do this, either turn the computer off and on or enter SYS 64738 (for the 64), SYS 65341 (128), SYS 64802 (VIC-20), or SYS 65526 (Plus/4 or 16). To reenable the Proofreader, reload the program and run it as usual. Unlike the original VIC/64 Proofreader, this version works the same with disk or tape.

On the Atari, run the Proofreader to activate it (the Proofreader remains active in memory as a machine language program); you must then enter NEW to erase the BASIC loader. Pressing SYSTEM RESET deactivates the Atari Proofreader; enter PRINT USR(1536) to reenable it.

The Apple Proofreader erases the BASIC portion of itself after you run it, leaving only the machine language portion in memory. It works with either DOS 3.3 or ProDOS. Disable the Apple Proofreader by pressing CTRL-RESET before running another BASIC program.

The IBM Proofreader is a BASIC program that simulates the IBM BASIC line editor, letting you enter, edit, list, save, and load programs that you type. Type RUN to activate. Be sure to leave Caps Lock on, except when typing lowercase characters.

Once the Proofreader is active, try typing in a line. As soon as you press RETURN, either a hexadecimal number (on the Apple) or a pair of letters (on the Commodore, Atari, or IBM) appears. The number or pair of letters is called a *checksum*.

Compare the value displayed on the screen by the Proofreader with the checksum printed in the program listing in the magazine. The checksum is given to the left of each line number. Just type in the program a line at a time (without the printed checksum), press RETURN or Enter, and compare the checksums. If they match, go on to the next line. If not, check your typing; you've made a mistake. Because of the checksum method used, do not type abbreviations, such as ? for PRINT. On the Atari and Apple Proofreaders, spaces are not counted as part of the checksum, so be sure you type the right number of spaces between quote marks. The Atari Proofreader does not check to see that you've typed the characters in the right order, so if characters are transposed, the checksum still matches the listing. The Commodore Proofreader catches transposition errors and ignores spaces unless they're enclosed in quotation marks. The IBM Proofreader detects errors in spacing and transposition.

IBM Proofreader Commands

Since the IBM Proofreader replaces the computer's normal BASIC line editor, it has to include many of the direct-mode IBM BASIC commands. The syntax is identical to IBM BASIC. Commands simulated are LIST, LLIST, NEW, FILES, SAVE, and LOAD. When listing your program, press any key (except Ctrl-Break) to stop the listing. If you enter NEW, the Proofreader prompts you to press Y to be especially sure you mean yes.

Two new commands are BASIC and CHECK. BASIC exits the Proofreader back to IBM BASIC, leaving the Proofreader in memory. CHECK works just like LIST, but shows the checksums along with the listing. After you have typed in a program, save it to disk. Then exit the Proofreader with the BASIC command, and load the program as usual (this replaces the Proofreader in memory). You can now run the program, but you may want to resave it to disk. This will shorten it on disk and make it load faster, but it can no longer be edited with the Proofreader. If you want to convert an existing BASIC program to Proofreader format, save it to disk with SAVE *"filename"*,A.

Program 1: Atari Proofreader

By Charles Brannon, Program Editor

```
100 GRAPHICS 0
110 FOR I=1536 TO 1700:REA
    D A:POKE I,A:CK=CK+A:N
    EXT I
120 IF CK<>19072 THEN ? "E
    rror in DATA Statement
    s.  Check Typing.":END

130 A=USR(1536)
140 ? :? "Automatic Proofr
    eader Now Activated."
150 END
160 DATA 104,160,0,185,26,
    3,201,69,240,7
170 DATA 200,200,192,34,20
    8,243,96,200,169,74
180 DATA 153,26,3,200,169,
    6,153,26,3,162
190 DATA 0,189,0,228,157,7
    4,6,232,224,16
200 DATA 208,245,169,93,14
    1,78,6,169,6,141
210 DATA 79,6,24,173,4,228
    ,105,1,141,95
220 DATA 6,173,5,228,105,0
    ,141,96,6,169
230 DATA 0,133,203,96,247,
    238,125,241,93,6
240 DATA 244,241,115,241,1
    24,241,76,205,238
250 DATA 0,0,0,0,0,32,62,2
    46,8,201
260 DATA 155,240,13,201,32
    ,240,7,72,24,101
270 DATA 203,133,203,104,4
    0,96,72,152,72,138
280 DATA 72,160,0,169,128,
    145,88,200,192,40
290 DATA 208,249,165,203,7
    4,74,74,74,24,105
300 DATA 161,160,3,145,88,
    165,203,41,15,24
310 DATA 105,161,200,145,8
    8,169,0,133,203,104
320 DATA 170,104,168,104,4
    0,96
```

Program 2: IBM Proofreader

By Charles Brannon, Program Editor

```
10 'Automatic Proofreader Vers
   ion 3.0 (Lines 205,206 adde
   d/190 deleted/470,490 chang
   ed from V2.0)
100 DIM L$(500),LNUM(500):COLO
    R 0,7,7:KEY OFF:CLS:MAX=0:
    LNUM(0)=65536!
110 ON ERROR GOTO 120:KEY 15,C
    HR$(4)+CHR$(70):ON KEY(15)
     GOSUB 640:KEY (15) ON:GOT
    O 130
120 RESUME 130
130 DEF SEG=&H40:W=PEEK(&H4A)
140 ON ERROR GOTO 650:PRINT:PR
    INT"Proofreader Ready."
150 LINE INPUT L$:Y=CSRLIN-INT
    (LEN(L$)/W)-1:LOCATE Y,1
160 DEF SEG=0:POKE 1050,30:POK
    E 1052,34:POKE 1054,0:POKE
     1055,79:POKE 1056,13:POKE
     1057,28:LINE INPUT L$:DEF
     SEG:IF L$="" THEN 150
170 IF LEFT$(L$,1)=" " THEN L$
    =MID$(L$,2):GOTO 170
```

COMMODORE ★ ATTARI ★ IBM ★ APPLE

MONITORS

$119.00

(RGB's Available)

MODEMS

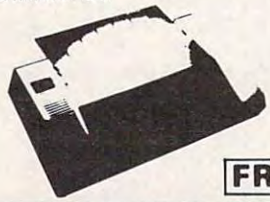
300 BAUD **$28.00**

Sale!

1200 BAUD **$98.00**

PRINTER

EPSON WARRANTED



NOW **$88.00** RETAIL $200.00

FREE TRIAL PERIOD

NLQ PRINTER

EPSON WARRANTED

NEAR LETTER QUALITY

NOW **$139.00** RETAIL $299.00

COMREX CR20-A

DISK DRIVES

1541 COMMODORE

$149.00

PLUS/4

COMMODORE COMPUTER

INCLUDES BUILT-IN SOFTWARE

for word processing, file management, spreadsheets and 128 color graphics!

64 POWER SUPPLY

$39.00

ELECTRIC TYPEWRITER

FULL WARRANTY

IBM SELECTRA II

NOW **$499.00** RETAIL $1200.00

FREE TRIAL PERIOD

PRO-TECH-TRONICS

6870 Shingle Creek Parkway #103 • Minneapolis, MN 55430 • (612) 560-6603

American Express · MasterCard · VISA

*IN STOCK ITEM

CALL TODAY

—NEXT-DAY DELIVERY*—

SCHOOL P.O.'s Accepted!

1-800-345-5080

Expert Back-up for Atari® Users.

SPECIALIZING IN BACKUP HARDWARE & SOFTWARE

THE 1050 DUPLICATOR™: The most powerful diskdrive copy system ever developed for the ATARI.

The only Copy system You will ever need.

What will it do?

▶ ***The main purpose of the Duplicator is to copy disks!*** *You will be able to copy just about any disk! The copies you make will run on any Atari drive. The Duplicator need not be present to run your backup copies. The Duplicator is fully automatic. You need only insert source and destination disks. Custom formats will be read and in turn reproduced on the backup copy disk. Our device will reproduce any custom format or heavily copy guarded scheme, bad sectors, double sectors, 19 through 24 sector format will present no problem to the Duplicator.*

▶ ***You will still have single density, density and one half, and double density.*** *When you have a Duplicator installed in a 1050 drive that drive will be turned into true double density. You will have twice the disk storage. Your drive will be compatible with other double density drives as The Rana Indus. Percom, etc.*

HARDWARE POWER

Fully Compatible with the XL & New XE Series.

▶ ***High speed read & write.*** *Your disk drive will read and load all of your software, saving wear and tear on your drive. The 1050 drive now reads one sector at a time. This is slow and inefficient. With the duplicator installed you will be able to read eighteen sectors in the time it takes standard, unenhanced drives to read one.*

▶ ***Included with every Duplicator will be user friendly disk software.*** *A simple, menu driven program will allow you to copy all of your software. A Duplicator enhanced drive will be a SMART drive. We plan to write many new and exciting programs that can only be run on an enhanced drive, eg. sending a copy-guarded disk over the phone. Since the drive is now fully programmable, future upgrades can be made available to you on disks, should the need arise. No further hardware changes will ever be needed. The Duplicator comes with a full hardware and software guarantee.*

Only **$149.95** *Plus $3.50 for Shipping & Handling Add 7% outside U.S.A.*

**Duplicator price may increase due to component shortage and price increases.*

N.Y. State Residents add 7½% Sales Tax

**Dealer inquiries are welcome, call for quantity price quote.*

EASY 5 MINUTE INSTALLATION

NO HARM TO YOUR DRIVE OR INCOMPATIBILITY PROBLEMS CAN ARISE AS A RESULT OF THE INSTALLATION OF OUR DUPLICATOR.

520 ST Copy Program

THE ST DUPLICATOR™

Our famous 1050 DUPLICATOR has been converted to service the **ATARI™** 520 ST. It is a software only disc copy system.

You can now back up heavily copy-guarded and protected discs. Presently, we can't find anything the ST DUPLICATOR will not copy. And if new forms of software protection should appear on the market, we will provide software upgrades.

ST DUPLICATOR™ INCLUDES USER FRIENDLY SOFTWARE AND INSTRUCTIONS **$59.95**

Plus $3.50 Shipping Add 7% Outside U.S.A.

THE HACKER'S TREASURE CHEST On Disk

18 Utility Programs on disk. Each program is specifically designed to aid you in backing-up your software collection. You will be able to duplicate **disks, cartridges** and **cassettes.** Any one program is worth the price of all 18. It has taken us over one year to put together this fine collection on the **Hacker's Treasure Chest** disk. Some of the programs you will receive are: **Cartridge Copy, Bootape Maker, Tape to Disk, SectorCopy, The Unprotector, Sector Disassembler, Bad Sector Finder, Modem Program...** plus more. All of these programs plus 10 more on this disk. You will also receive a **menu** that will run basic and binary flies just by typing the number of the program. Any file on any disk will load automatically from this menu. ALL FOR ONLY **$34.95**

Plus $3.50 Shipping Add 7% outside U.S.A.

DENSITY "DOUBLER"

1050

DOUBLE THE POWER...

TRIPLE THE SPEED...

Get TRUE double density, full compatibility with any DOS. Now store twice as much data on each disk. Read and write up to 3X faster in single or double density (Whole Track Buffering). Includes ultra-speed software, simple plug-in PC board No soldering or cutting required. **$59.95**

Plus $3.50 Shipping Add 7% outside U.S.A.

WRITE-RIGHT

This device will allow you to write to side 2 of any disk. Install this box to your **ATARI™** 1050 in 5 minutes. Just plug in one cable - no cutting or soldering required. Push a button and a LED will light, allowing you to write to a disk without notching out a hole in the disk. Easy plug-in installation. Instructions included. Fully tested & assembled. **$29.95**

Plus $3.50 Shipping Add 7% outside U.S.A.

CART CLONE™

COPY ANY ATARI™ CARTRIDGE

A MUST FOR ALL ATARI™ USERS **CART CLONE** will backup and transfer any 8 or 16k cartridge to disk or tape. The contents of the cartridge will become a file which you can transfer, rename or delete. They will execute from DOS. No need to run a special menu or program to run these files it goes in the left cartridge slot enabling it to work in all ATARI™ Home Computers including the XL series. You can get **CART CLONE™** with software for **$59.95**

Plus $3.50 Shipping Add 7% outside U.S.A.

"Constantly Working on New Products and Software."

DUPLICATING TECHNOLOGIES inc.

Formerly Gardner Computing

99 Jericho Tpke., Suite 302A Jericho, N.Y. 11753

TECHNICAL INFO ONLY **(516) 333-5504, 5950**

WEEKDAY ORDERS ONLY **(516) 333-5805, 5807, 5808**

EVE. & WKEND. ORDERS ONLY **(516) 333-5950**

TERMS: *We accept American Express, Visa, MasterCard and C.O.D. orders. Foreign orders must be in U.S. dollars. All personal checks allow 14 days to clear. Most items shipped within 24 hours.*

Classified

SOFTWARE

LOTTO PICKER. Improve your chances for those Million Dollar Jackpots! Picks LOTTO, WIN-4 and Daily Numbers. All USA & CAN. games incl'd. Expandable! IBM/C64/TI99 $29.95. Order Now! 1-800-341-1950 ext. 77. Mail Orders: Ridge, 170 B'way, #201C, NY, NY 10038. Catalog.

FREE SOFTWARE for C64, C128, IBM, & CPM. For info send self-addressed, stamped envelope to: PUBLIC DOMAIN USERS GROUP, PO Box 1442-A1, Orange Park, FL 32067

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

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

DISCOUNT SOFTWARE: Amiga/Apple/Atari/C64-128/IBM PC-PCjr/TRS-80/Timex/Sinclair. Free Catalog: WMJ DATA SYSTEMS, 4 Butterfly Dr., Hauppauge, NY 11788

Genealogy Programs for the 64 & 128. **Family Tree** will produce Pedigree Charts, Family Group Records, Individual Files, Indexes, Searches of Ancestors. LDS version available. $49.95 Genealogy Software, POB 1151, Port Huron, MI 48061, (519) 344-3990

FREE SOFTWARE CATALOG
for APPLE, ATARI, C64, IBM, ST - 1/3 OFF retail prices. We carry SSI, Infocom, Elec. Arts, Microprose, and many more. TEVEX, Dept. C, 1710 Wilwat Dr. #E, Norcross, GA 30093

LINCAD, CALCAD, FFT-SPECTRUM ANALYSIS. VALUABLE PROGRAMS FOR ELECTRONIC CIRCUIT OR SIGNAL ANALYSIS & DESIGN. LINCAD FOR LINEAR CIR.-C64:$49, IBM-PC:$99. CALCAD FOR LOGIC ANALYSIS + SYNTHESIS - IBM-PC:$179. FFT ANALYZER - C64:$49, IBM-PC:$99. +$3.50 S/H. OH RES. ADD 5.5% TAX. CHECK OR M.O. WRITE FOR DETAILS. SOFCAD ELECTRONICS, INC., PO BOX 21845, COLUMBUS, OH 43221

HEY AMIGO! PD Software for Amiga. Games, graphics, utilities, more! Over 15 disks available only $7.95 each! SASE for list. Casaware, PO Box 1646, Orange Park, FL 32067

WIN AT CASINO BLACKJACK! Disk includes game, play analyzer and powerful point count system. Proves itself on your machine. For IBM-PC (no graphics card req'd), Atari 800 (48K)/XL/XE. Specify. $19.95. BISONSOFT, Box 2136, Bloomington, IN 47402

$ LUCKY LOTTO $ — Lotto pgm for all states. Over 2 yr. R&D. For 64K + disk. IBM, CoCoII, C64, Tan.1000, $49.95 + $5 s/h to: Delta, POB 968, Concord, NC 28026 (spec st.)

TANDY 1000 PROGRAMS AND NEWSLETTER Send for free information on educational & entertainment programs & newsletter. Soda Pop Software, POB 653, Kenosha, WI 53141

REAL PINOCHLE. Double-deck, 4-handed, partnership for one player. For C64, PC, PCjr. On disk $20. Jim Bernard, 301 Forest Dr., Bellevue, NE 68005

BRIDGE GAME PROGRAM $39.95 demo disk $5. 1 to 4 players for IBM, Apple, TI99, TRS80, C64, 128, +4, 16, VIC. Authors John & Lynda Allan, Box 313, Azilda, Ontario, Canada P0M 1B0

Animal Records maintained with **Petigree** for the C64. Produces Litter, Awards, Breeding, Show, Individual Records, Pedigree Charts. $69.95. Genealogy Software, POB 1151, Port Huron, MI 48061, (519) 344-3990

FREE! PC PUBLIC DOMAIN SOFTWARE! MS-DOS, IBM & Compatibles - Save $$$ @ $2.50 per disk! Free info: AP-JP, Inc., PO Box 1155, W. Babylon, NY 11704

HARDWARE

ASSEMBLE YOUR OWN JOYSTICKS & SAVE $$
Kits available for IBM, Apple, Commodore, Atari, TI99/4A. Schematics, plastics, cables, components included. Send $24.95 for Apple or IBM, $12.95 for TI99/4A, $9.95 for Commodore/Atari. JOYSTICK KITS, BOX 8248, LONGVIEW, TX 75607 214-758-0040. Specify Computer type.

REPAIR YOUR OWN JOYSTICKS & SAVE $$
Repair kits available for P.P.I. & TG Product Joysticks. Send $1 and SASE for Parts & Price List. JOYSTICK KITS, BOX 8248, LONGVIEW, TX 75607 214-758-0040

MISCELLANEOUS

IBM PCjr. REPORT: THE NATIONAL NEWS-LETTER PCjr.-specific articles, reviews, Public Domain from across the nation. $18/yr. PCjr. CLUB, POB 95067, Schaumburg, IL 60195

MIDWEST COMMODORE CONFERENCE/EXPO
C64, C128, AMIGA - AUG. 9 & 10, 1986 - $20/person. Greater Omaha Commodore Users Group. PO Box 24115, Omaha, NE 68124

FLIGHT SIMULATOR II CONTROL CONSOLE
Complete set of plans for aircraft-style controls and panel. C64. $15 to: FLIGHT SYSTEMS, Rte 2, Box 121, Gaston, OR 97119

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.) Inquire about display rates.
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.

Advertisers Index

To Our Readers:

COMPUTE! Publications is a part of the ABC Consumer Magazines group of ABC Publishing, Inc. and recently we consolidated many of our operations and moved our Customer Service Department to the New York ABC headquarters. If you have any questions regarding back issues, disk orders, book orders, or how to place an order, call toll free **1-800-346-6767**. New York residents should call 212-887-8525.

If you want to order a subscription to COMPUTE!, COMPUTE!'s GAZETTE, COMPUTE!'s GAZETTE DISK, or the COMPUTE! DISK, call **1-800-247-5470** or in Iowa call 1-800-532-1272.

Our Editorial Offices remain in Greensboro, North Carolina. If you wish to submit an article for publication, write us at COMPUTE! Publications, Inc., P.O. Box 5406, Greensboro, NC 27403.

We thank you for your interest and continued support of COMPUTE! Publications.

COMPUTE! Publications, Inc. abc

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

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 Disk

SPECIAL INTRODUCTORY OFFER

YES! I want to save time and money. Please enter my quarterly subscription to the following COMPUTE! Disk:

☐ Commodore ☐ Apple ☐ Atari ☐ IBM

☐ Save 33% off the single issue price. 1 year subscription, $39.95

☐ Save even more! 2 year subscription, $69.95

All *Disk* orders must be prepaid.

☐ Payment enclosed (check or money order)
☐ Charge ☐ MasterCard ☐ Visa

Acct. No. ______________________ Exp. Date __________
Signature ______________________
Name ______________________
Address ______________________
City ______________ State ________ Zip ________

Save 33%

(Outside U.S. and Canada, add $9.00 per year for shipping and handling.)

S

J1809

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 7551 DES MOINES, IA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPUTE!'s DISK
P.O. BOX 10036
DES MOINES, IA 50347

Flight Simulator II Scenery Disks

The Challenge of Accomplished Flight

With a realism comparable to (and in some ways even surpassing) $100,000 aircraft flight simulators, Flight Simulator II includes full flight instrumentation and avionics, and provides a full-color out-the-window view. Instruments are arranged in the format standard to modern aircraft. All the radios needed for IFR flight are included. Front, rear, left, right, and diagonal views let you look in any direction. Program features are clearly documented in a 96-page Pilot's Operating Handbook.

For training in proper flight techniques, Flight Simulator II includes another 96-page instruction manual, compiled by two professional flight instructors with over 8,000 hours flight time and 12,000 hours of aviation teaching experience. You'll learn correct FAA-recommended flight procedures, from basic aircraft control through instrument approaches. To reward your accomplishments, the manual even includes a section on aerobatic maneuvers.

The Realism and Beauty of Flight

Go sight-seeing over detailed, realistic United States scenery. High-speed graphic drivers provide an animated out-the-window view in either day, dusk, or night flying modes.

Flight Simulator II features over 80 airports in four different scenery areas: New York, Chicago, Seattle, and Los Angeles. Six additional Scenery Disks covering the entire Western half of the United States are now available in IBM and C64/128 disk formats.

Apple and Atari versions will be released soon. Each disk covers a geographical region of the country in detail, and is very reasonably priced.

The Pure Fun of "World War I Ace"

When you think you're ready, you can test your flying skills with the "World War I Ace" aerial battle game. This game sends you on a bombing run over heavily-defended enemy territory. Six enemy fighters will attempt to engage you in combat as soon as war is declared. Your aircraft can carry five bombs, and your machine guns are loaded with 100 rounds of ammunition.

See Your Dealer. Flight Simulator II is available on disk for the Apple II, Atari XL/XE, and Commodore 64/128 computers for $49.95. Scenery Disks for the C64 and IBM PC (Jet or Microsoft Flight Simulator) are $19.95 each. A complete Western U.S. Scenery six-disk set is also available for $99.95. For additional product or ordering information, call (800) 637-4983.

subLOGIC
Corporation
713 Edgebrook Drive
Champaign IL 61820
(217) 359-8482 Telex: 206995

Order Line: (800) 637-4983
(except in Illinois, Alaska, and Hawaii)