Mousify Your Applesoft Programs

Lee Swoboda

Here's the first installment of a two-part tutorial on interfacing and using a mouse with your Apple II computer. Though a mouse is preferred, you can use the techniques shown here to substitute a joystick or game paddles for the mouse.

Allow me to introduce a new word—*mousify*. Don't try to look that up in a dictionary—it's not there, at least not yet. Though Apple didn't invent the mouse, their Macintosh was the first popular home computer that used one; and it has changed the way that we interface with computers. So the need for the word *mousify*—to add mouse control to computer hardware or software—may grow.

A Mouse For Apple II

Apple II computers don't come with a mouse, but it's now possible to add one. If you have an Apple II, II+, or IIe, you'll need an interface card (Apple product A2M2050, $149.95 including the mouse). The interface is built into the IIc, so you only need to buy the mouse device (Apple product A2M4015, $99.95).

Attaching a mouse to your Apple II is only half the battle. To your computer, a mouse is merely another input device, like a joystick. Many new programs have the ability to accept mouse input. But there are thousands of existing programs that were written before the mouse appeared on the scene. Most commercial programs are written in machine language and protected from unauthorized access so, unless you are an expert machine language programmer, you won't be able to mousify them. However, Applesoft BASIC programs can easily be mousified.

Pages 35–40 of the *AppleMouse II User's Manual*, which comes with the mouse, give a brief description of how to use the mouse in a BASIC program. But the two examples are trivial and the text fails to mention some of the "features" that make mousifying an Applesoft program difficult. For example, you must use the INPUT statement to obtain mouse position parameters. Unfortunately, the INPUT command erases one line of screen text. In addition, your program must set the computer to receive input, which disconnects the keyboard. Mixing mouse and keyboard input requires switching input control between the mouse and the keyboard. Not only that, you have to use GET statements for keyboard input, which can be exasperating. Using these techniques quickly turns your BASIC program into a Rube Goldberg contraption of INPUTs, GETs and PRINT D$'s. But there's a better way to handle the mouse.

Fortunately, Apple's input and output (I/O) are *memory-mapped*, meaning that the keyboard and each character on the 40-column screen are at specific locations in Apple's main memory. Applesoft's PEEK and POKE commands let us examine and change characters on the screen without having to use INPUTs, GETs or PRINTs. This is especially important when using the mouse, since it lets you zoom quickly to any point on the screen to change a character, rather than perform complex string manipulations.

Practical Application

Let's see how this works. Type in and save Programs 1 and 2, then load and run Program 2, which creates a text file for Program 1 to read. When that's done, load and run Program 1. You'll see an input screen, typical of what might be used in an address book program that lets you store and recall names and addresses. (Of course, this program is just for demonstration; you can't use it to create a real address file.)

In a typical program, the computer would make you reenter an entire line to correct a misspelling or other error. Program 1 lets you point directly at an error with the mouse, and change only the incorrect character. The initial screen display should contain this information:

ENTER INFORMATION

FIRST NAME ...	COMPUTE!
LAST NAME	REEDER SERVICE
STREET	P.O. BOX 2141
CITY	RANDOR
STATE	PA
ZIP	19089
TELEPHONE	1-800-334-0868

ERASE QUIT DONE HELP

Notice that the word READER is misspelled REEDER. That's the mistake you'll correct. Also notice there are three flashing rectangles on the screen. The rapidly flickering rectangle in the upper-right corner is produced when Program 1 obtains mouse input (lines 10150–10160). This effect is always present, but is unimportant to the present discussion. The flashing C at the beginning of the word COMPUTE! is the cursor. The blinking reflex (^) in the upper left corner of the screen is the mouse pointer. The mouse moves the mouse pointer around the screen.

Move the mouse so the mouse pointer replaces the second E in REEDER and press the mouse button. The computer immediately moves the cursor to the same spot. Now type the letter A (upper or lower case) to correct the spelling mistake. That's all you need to do to correct the error. You can also use the arrow keys to move the cursor (Apple II uses the CTRL-J and CTRL-K keys to simulate the up and down arrow keys of the IIe and IIc). But the mouse moves the cursor much more rapidly, and is far more intuitive to use.

Now move the pointer to the word DONE in the strip menu at the bottom of the screen, and press the button. The computer reads the information from screen memory and, in this case, redisplays the updated information. Of course, in a working program you would replace lines 30120–30190 with routines that store the data for later recall. You can move the mouse pointer or cursor anywhere on the screen, but line 10710 of the program prevents you from typing anything outside the text area.

How The Program Works

Let's take a closer look at the significant portions of Program 1. Lines 130 and 10000–10830 are the most important. Line 130 sets the sensitivity of the mouse. When MI has a value of 20, the pointer moves to any part of the 40-column screen as the mouse moves within a 5 × 8 inch area. Give MI a larger value to make the mouse less responsive.

Lines 10070–10090 activate the mouse. Input control is transferred from the keyboard to the mouse, until line 20030 returns control to the keyboard. Lines 10150–10270 calculate the horizontal and vertical position of the mouse and determine whether the mouse button has been pushed. Line 10170 and lines 10440–10760 handle input from the keyboard. Note that input control remains with the mouse at this point: The program does *not* use the statement PRINT D$"IN#0" to return control to the keyboard. This is the key to the simplicity of Program 1, since it avoids the problems normally encountered when using GET and PRINT commands with DOS.

Line 10220 and lines 10230–10270 move the cursor to the same position as the mouse pointer when you press the mouse button. Lines 10320–10390 position the mouse pointer and return to line 10150 to read the mouse again. If you don't press a key or the mouse button, the computer stays in the loop from 10150 to 10390, reading the mouse and repositioning the mouse pointer. Lines 10590–10620 position the cursor. This routine is activated only when you press an arrow key or the mouse button. Lines 10640–10690 change all upper- and lower-case and all inverse characters to flashing.

Substituting A Joystick Or Game Paddles

If you don't have a mouse, you can use a joystick (or, less conveniently, game paddles) to achieve the same effects. With a few modifications, Program 1 can be made to accept joystick or paddle input. Here are the steps to follow:

1. Delete lines 120, 130, 10001–10090 and 20220.
2. Modify the following lines as shown:

```
CB 1Ø15Ø XØ=PDL(Ø)
04 1Ø16Ø YØ=PDL(1)
A0 1Ø17Ø IFBØ>127THEN1Ø44Ø
A9 1Ø18Ø YØ=INT(YØ/1Ø)+1
47 1Ø2ØØ XØ=INT(XØ/6)+1
60 1Ø22Ø IFBØ<128THEN1Ø32Ø
32 2ØØ3Ø REM
```

3. Add the following lines:

```
6D 1Ø165 BØ=PEEK(-16384)
54 1Ø215 BØ=PEEK(-16287)
```

After making these changes, resave Program 1, using a different filename to distinguish it from the original version. When you run it, the joystick moves the mouse pointer around the screen and the button works just like the mouse button. At this point you might wonder why anyone would buy a mouse, since a joystick or game paddle seems to work as a substitute. Part of the reason is simply preference—many people find that a real mouse "feels" better and is therefore more convenient than a joystick. More significantly, most commercial programs that accept mouse input do not recognize input from a joystick or paddles. If you're writing programs strictly for your own use, a joystick may serve the purpose; but if you buy commercial software that requires a mouse, you may have no choice.

Using a mouse is a new experience for many Apple II owners. I hope this program inspires you to mousify some of your own programs. In Part 2 of this article we'll expand the capabilities of Program 1 to let you use the mouse to delete and insert blocks of text.

Program 1. Apple II Mouse Demonstration

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

```
BD 12Ø SØ = 2: REM SLOT CONTAINI
       NG MOUSE
D7 13Ø MI = 2Ø: REM MOUSE SENSIT
       IVITY
5A 14Ø D$ = CHR$ (4)
8C 15Ø REM
07 16Ø REM READ DATA FILE
90 17Ø REM
CB 18Ø PRINT D$"OPEN TEXT"
32 19Ø PRINT D$"READ TEXT"
6D 2ØØ INPUT NF$,NL$,AD$,CI$,ST$
       ,ZI$,TE$
CD 21Ø PRINT D$"CLOSE TEXT"
87 22Ø REM
25 23Ø REM DATA ENTRY SCREEN
8B 24Ø REM
4F 25Ø HOME
4D 26Ø Y1 = 4:X1 = 15:CØ = 16Ø
35 27Ø INVERSE
D7 28Ø PRINT "                ENTER
       INFORMATION            "
2C 29Ø VTAB 24: PRINT " MENU:  E
       RASE  QUIT  DONE  HELP
          ";
C6 3ØØ NORMAL
31 31Ø VTAB 4: HTAB 1
F4 32Ø PRINT "FIRST NAME .."
C6 33Ø PRINT "LAST NAME ..."
3C 34Ø PRINT "STREET ......"
D6 35Ø PRINT "CITY ........"
1F 36Ø PRINT "STATE ......."
36 37Ø PRINT "ZIP ........."
17 38Ø PRINT "TELEPHONE ..."
3A 39Ø VTAB 19: HTAB 1Ø: INVERSE
        : PRINT "^";: NORMAL
81 4ØØ PRINT " IS MOUSE POINTER"
3C 41Ø VTAB 21: HTAB 14: INVERSE
        : PRINT " ";: NORMAL
38 42Ø PRINT " IS CURSOR"
26 43Ø VTAB 4
5E 44Ø HTAB 15: PRINT NF$
```

```
66 45Ø HTAB 15: PRINT NL$
D9 46Ø HTAB 15: PRINT AD$
E1 47Ø HTAB 15: PRINT CI$
F6 48Ø HTAB 15: PRINT ST$
71 49Ø HTAB 15: PRINT ZI$
59 5ØØ HTAB 15: PRINT TE$
73 9999 REM #1ØØØØ
19 1ØØØØ REM
E6 1ØØØ1 REM ---------------
29 1ØØ1Ø REM MOUSE ROUTINES
E6 1ØØ2Ø REM ---------------
39 1ØØ4Ø REM
A4 1ØØ5Ø REM TURN MOUSE "ON"
49 1ØØ6Ø REM
A8 1ØØ7Ø PRINT D$"PR#"SØ: PRINT CHR$ (1)
CB 1ØØ8Ø PRINT D$"PR#Ø"
Ø9 1ØØ9Ø PRINT D$"IN#"SØ
17 1Ø1ØØ GOTO 1Ø59Ø
25 1Ø11Ø REM
65 1Ø12Ø REM DETERMINE POSITION
91 1Ø13Ø REM OF MOUSE
3D 1Ø14Ø REM
1C 1Ø15Ø VTAB 1: HTAB 4Ø
77 1Ø16Ø INPUT "";XØ,YØ,BØ
7D 1Ø17Ø IF BØ < Ø THEN 1Ø44Ø: REM KEY PRESSED?
DØ 1Ø18Ø YØ = INT (YØ / MI) + 1
7B 1Ø19Ø IF YØ > 24 THEN YØ = 24
64 1Ø2ØØ XØ = INT (XØ / MI) + 1
75 1Ø21Ø IF XØ > 4Ø THEN XØ = 4Ø
ØB 1Ø22Ø IF BØ > 1 THEN 1Ø32Ø: REM BUTTON PRESSED?
B9 1Ø23Ø IF YØ = 24 THEN 2ØØ3Ø
63 1Ø24Ø Y1 = YØ:X1 = XØ
78 1Ø25Ø POKE VØ,CØ
4B 1Ø26Ø CØ = C2
F2 1Ø27Ø GOSUB 1Ø8ØØ
F2 1Ø28Ø GOTO 1Ø62Ø
69 1Ø29Ø REM
ED 1Ø3ØØ REM POSITION MOUSE POINTER
2D 1Ø31Ø REM
86 1Ø32Ø IF VØ = V1 THEN C2 = C1
BØ 1Ø33Ø POKE V1,C2
A2 1Ø34Ø V1 = 1Ø23 + 128 * (YØ - 1) + XØ
3F 1Ø35Ø IF YØ > 8 THEN V1 = V1 - 984
9C 1Ø36Ø IF YØ > 16 THEN V1 = V1 - 984
27 1Ø37Ø C2 = PEEK (V1)
64 1Ø38Ø POKE V1,16Ø
ØA 1Ø39Ø IF C2 = 16Ø THEN POKE V1,3Ø
C2 1Ø4ØØ GOTO 1Ø15Ø
31 1Ø41Ø REM
Ø1 1Ø42Ø REM KEYBOARD INPUT
41 1Ø43Ø REM
F9 1Ø44Ø C3 = PEEK ( - 16384)
77 1Ø45Ø POKE - 16368,Ø
DC 1Ø455 IF C3 > 223 THEN C3 = C3 - 32: REM CONVERT TO UPPER CASE
48 1Ø46Ø IF C3 > 159 THEN 1Ø71Ø
CD 1Ø47Ø IF C3 = 141 THEN X1 = 15:Y1 = Y1 + 1: IF Y1 > 1Ø THEN Y1 = 4: REM RETURN KEY
69 1Ø48Ø IF C3 = 139 THEN Y1 = Y1 + 1: REM DOWN ARROW
B2 1Ø49Ø IF C3 = 138 THEN Y1 = Y1 - 1: REM UP ARROW
BF 1Ø5ØØ IF C3 = 149 THEN X1 = X1 + 1: REM RIGHT ARROW
71 1Ø51Ø IF C3 = 136 THEN X1 = X1 - 1: REM LEFT ARROW
56 1Ø52Ø IF Y1 > 24 THEN Y1 = 24
DC 1Ø53Ø IF Y1 < 1 THEN Y1 = 1
9C 1Ø54Ø IF X1 > 4Ø THEN X1 = 4Ø
EB 1Ø55Ø IF X1 < 1 THEN X1 = 1
5D 1Ø56Ø REM
Ø6 1Ø57Ø REM POSITION CURSOR
6D 1Ø58Ø REM
A4 1Ø59Ø POKE VØ,CØ
CA 1Ø6ØØ GOSUB 1Ø8ØØ
42 1Ø61Ø CØ = PEEK (VØ)
9E 1Ø62Ø IF VØ = V1 THEN CØ = C2
44 1Ø63Ø REM CHANGE TO FLASHING CHARACTER
87 1Ø64Ø C1 = CØ
23 1Ø65Ø IF C1 > 127 THEN C1 = C1 - 64
7F 1Ø66Ø IF C1 > 64 THEN C1 = C1 - 64
D9 1Ø67Ø IF C1 > 95 THEN C1 = C1 - 32
4B 1Ø68Ø IF C1 < 64 THEN C1 = C1 + 64
C8 1Ø69Ø POKE VØ,C1
CE 1Ø7ØØ GOTO 1Ø15Ø
6B 1Ø71Ø IF X1 < 15 OR Y1 < 4 OR Y1 > 1Ø THEN 1Ø15Ø
DE 1Ø72Ø GOSUB 1Ø8ØØ
DC 1Ø73Ø POKE VØ,C3
51 1Ø74Ø CØ = C3
CE 1Ø75Ø IF VØ = V1 THEN C2 = C3
ØC 1Ø76Ø X1 = X1 + 1: IF X1 > 39 THEN X1 = 39
67 1Ø77Ø GOTO 1Ø59Ø
16 1Ø78Ø REM CALCULATE VØ
6E 1Ø79Ø REM (VIDEO BUFFER ADDRESS)
6Ø 1Ø8ØØ VØ = 1Ø23 + 128 * (Y1 - 1) + X1
2B 1Ø81Ø IF Y1 > 8 THEN VØ = VØ - 984
7F 1Ø82Ø IF Y1 > 16 THEN VØ = VØ - 984
8B 1Ø83Ø RETURN
9A 19999 REM #2ØØØØ
1A 2ØØØØ REM
AE 2ØØ1Ø REM STRIP MENU
2A 2ØØ2Ø REM
C2 2ØØ3Ø PRINT D$"IN#Ø"
CB 2ØØ4Ø IF XØ > 8 AND XØ < 14 THEN NF$ = "":NL$ = "":AD$ = "":CI$ = "":ST$ = "":ZI$ = "":TE$ = "": GOTO 25Ø
1F 2ØØ5Ø IF XØ > 15 AND XØ < 2Ø THEN HOME : END
73 2ØØ6Ø IF XØ > 21 AND XØ < 26 THEN 3ØØ3Ø
7A 2ØØ7Ø IF XØ > 27 AND XØ < 32 THEN 2Ø1ØØ
71 2ØØ8Ø VTAB 1: HTAB 4Ø: PRINT D$"IN#"SØ: GOTO 1Ø15Ø
17 2ØØ9Ø REM HELP TEXT
CD 2Ø1ØØ VTAB 12: HTAB 1
8A 2Ø11Ø PRINT "THE FLASHING REFLEX (^) IS THE MOUSE"
75 2Ø12Ø PRINT "POINTER AND THE FLASHING RECTANGLE IS"
4B 2Ø13Ø PRINT "THE CURSOR.  TO MOVE THE CURSOR TO THE"
36 2Ø14Ø PRINT "ENTRY YOU WANT TO CHANGE, USE THE ARROW "
4E 2Ø15Ø PRINT "KEYS OR USE THE MOUSE TO MOVE THE MOUSE "
47 2Ø16Ø PRINT "POINTER, THEN PRESS THE MOUSE BUTTON TO "
E6 2Ø17Ø PRINT "MOVE THE CURSOR TO THAT POINT.  TYPE"
EA 2Ø18Ø PRINT "NEW OR CORRECTED DATA, THEN MOVE THE"
31 2Ø19Ø PRINT "MOUSE CURSOR TO 'DONE' IN THE MENU"
4A 2Ø2ØØ PRINT "BELOW AND PRESS THE MOUSE BUTTON TO"
D4 2Ø21Ø PRINT "ACCEPT THE ENTRIES ABOVE."
D9 2Ø22Ø PRINT D$"IN#"SØ
D3 2Ø23Ø GOTO 1Ø15Ø
9D 29999 REM #3ØØØØ
1B 3ØØØØ REM
28 3ØØ1Ø REM EXAMPLE
2B 3ØØ2Ø REM
A1 3ØØ3Ø Y1 = 4: GOSUB 63Ø5Ø:NF$ = A$
2C 3ØØ4Ø Y1 = 5: GOSUB 63Ø5Ø:NL$ = A$
91 3ØØ5Ø Y1 = 6: GOSUB 63Ø5Ø:AD$ = A$
1C 3ØØ6Ø Y1 = 7: GOSUB 63Ø5Ø:CI$ = A$
E9 3ØØ7Ø Y1 = 8: GOSUB 63Ø5Ø:ST$ = A$
11 3ØØ8Ø Y1 = 9: GOSUB 63Ø5Ø:ZI$ = A$
17 3ØØ9Ø Y1 = 1Ø: GOSUB 63Ø5Ø:TE$ = A$
2E 3Ø1ØØ REM GO TO REMAINDER OF YOUR PROGRAM
1C 3Ø11Ø REM FOR EXAMPLE ...
36 3Ø12Ø HOME
5E 3Ø13Ø VTAB 1Ø
EE 3Ø14Ø PRINT NF$" "NL$
38 3Ø15Ø PRINT AD$
BØ 3Ø16Ø PRINT CI$", "ST$" "ZI$
9C 3Ø17Ø PRINT TE$
CA 3Ø18Ø CALL - 198: CALL - 198
89 3Ø19Ø END : REM END OF EXAMPLE
A5 62999 REM #63ØØØ
24 63ØØØ REM
2C 63Ø1Ø REM SUBROUTINE TO "READ"
1F 63Ø2Ø REM STRINGS FROM THE
B3 63Ø3Ø REM VIDEO BUFFER
44 63Ø4Ø REM
ØF 63Ø5Ø VTAB 24: FLASH : PRINT " WORKING ...                    ";: NORMAL : VTAB 1: HTAB 1
C9 63Ø6Ø A$ = ""
FC 63Ø7Ø REM CALCULATE VØ
55 63Ø8Ø REM (VIDEO BUFFER ADDRESS)
A5 63Ø9Ø VØ = 1Ø37 + 128 * (Y1 - 1)
12 631ØØ IF Y1 > 8 THEN VØ = VØ - 984
66 6311Ø IF Y1 > 16 THEN VØ = VØ - 984
2F 6312Ø FOR I = 1 TO 25
67 6313Ø CØ = PEEK (VØ + I)
DD 6314Ø IF CØ = 16Ø AND PEEK (VØ + I + 1) = 16Ø THEN 6319Ø: REM END IF TWO BLANKS
F9 6316Ø IF CØ > 128 THEN CØ = CØ - 128
F5 6317Ø A$ = A$ + CHR$ (CØ)
D5 6318Ø NEXT I
C2 6319Ø IF RIGHT$ (A$,1) = CHR$ (32) THEN A$ = LEFT$ (A$, LEN (A$) - 1): GOTO 6319Ø: REM REMOVE TRAILING BLANKS
66 632ØØ RETURN
```

Program 2. Sample Screen Maker

```
51 1Ø D$ = CHR$ (4)
Ø7 2Ø PRINT D$"OPEN TEXT"
CF 3Ø PRINT D$"WRITE TEXT"
EA 4Ø PRINT "COMPUTE!"
8E 5Ø PRINT "REEDER SERVICE"
F1 6Ø PRINT "P.O. BOX 1Ø958"
E3 7Ø PRINT "DES MOINES"
C8 8Ø PRINT "IA"
FC 9Ø PRINT "5Ø95Ø"
E9 1ØØ PRINT "1-8ØØ-346-6767"
CC 11Ø PRINT D$"CLOSE TEXT"
```

©

Atari BootStuffer

Randy Boyd

This short, handy program for all eight-bit Atari computers lets you store as many as ten boot programs on a single disk and execute any of the programs just by pressing one key. A disk drive is required.

If you're like many Atari computer users, you probably have a number of disks that contain nothing but a single boot program. Even if you don't mind the expense of storing only one program per disk, that's not a very efficient arrangement. "Atari BootStuffer" allows you to put as many as ten boot programs on a single disk (depending on how long each program is), and still use each program as if it were alone on the disk.

Type in Atari Bootstuffer from the listing below, and save it. As listed here, the program works on an Atari 800 with an 810 disk drive. If you have an XL or XE model, change the numbers in line 750 as shown in the REM in line 740. If you have a 1050 disk drive with DOS 2.5 or 3.0 and wish to use enhanced-density, change lines 1140, 1170, 1300 and 1340 as indicated in the REM lines in the program listing. Changing those lines gives you 1040 sectors per disk (of course, this is not possible on an 810 disk drive, which doesn't support enhanced-density).

Creating A BootStuffer Disk

Before running Atari BootStuffer, format a disk to be used as the special BootStuffer disk. Now run the program and insert the freshly formatted disk in the drive. When you press the space bar, the screen turns green and the drive spins for about one minute. When the screen turns red, the special disk is ready to use. Reboot the system: The computer loads and executes a machine language program which lets you use the BootStuffer disk. When the prompt appears, you can press S to save a program on the disk or press L to load and run a program.

Since you just formatted the disk, it doesn't yet contain any programs you can load. Press S to choose the save option. The program indicates how many sectors are free in the current block and asks whether you want to load the target program from disk (press D) or cassette (press C). From that point onward, simply follow the prompts on the screen: The target program is loaded into memory and saved on the boot disk. If a load error occurs, the screen flashes red and the program starts over again. By repeating this process, you can save as many as ten boot programs on one disk (of course, the number of programs you can fit on one disk depends on how long they are).

BootStuffer prepares the disk by dividing it into ten blocks numbered 0–9, each containing 255 sectors. Since it uses the operating system boot routines, this program is not able to read sectors 256, 512, 768 or 1024. The BootStuffer code occupies the 13 lowest-numbered sectors on the disk, so a single-density disk can store programs only in sectors 13–255, 257–511 and 513–720. An enhanced-density disk with 1040 sectors can use all of the single-density sectors plus sectors 513–767, 769–1023 and 1025–1040.

It's important that you arrange the boot programs to fit into the Bootstuffer disk without wasting too many sectors. The program tells you how many sectors are left in the current block, and how many sectors are in the program you're trying to save. If a program is too large to fit in the current block, BootStuffer prompts you to save a smaller program in that block. If you don't have a smaller program, you can press N to advance to the next block. However, skipping to the next block wastes the free sectors remaining in the last block. If you try to save a program that requires more space than is left on the disk, BootStuffer generates a DISK FULL message and permits you to save a smaller program in the same space.

When you name a program to be saved on the disk, make sure the name is ten characters or less. Once you have saved as many programs as you want, put the BootStuffer disk in the drive and reboot the system, then press L to choose the load option. The contents of all ten blocks are displayed, and you're prompted to choose which program you want to execute. Press a number key from 0–9: The program in that block automatically loads from disk and executes. Blocks that don't contain a program are marked as empty. Don't select an empty block from the load menu: You may cause the system to crash.

Atari BootStuffer

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

```
KL 500 FOR X=16384 TO 17920:
       POKE X,0:NEXT X
BA 510 ? "PLACE FORMATTED DI
       SK IN DRIVE"
EN 520 ? "PRESS SPACE BAR":?
       :? "█PLEASE WAIT█"
```

```
BH 530 IF PEEK(764)=255 THEN GOTO 530
OA 540 ST=1536:POKE 710,192:POKE 712,192
LD 550 READ J:IF J=-1 THEN GOTO 580
AI 560 POKE ST,J:ST=ST+1
BP 570 GOTO 550
IH 580 ST=16384
LD 590 READ J:IF J=-2 THEN GOTO 620
AD 600 POKE ST,J:ST=ST+1
BO 610 GOTO 590
EH 620 X=USR(1536)
LC 630 ? :? :PRINT "DONE":POKE 712,64:POKE 710,64
CB 634 ? "PRESS SPACE BAR FOR ANOTHER COPY"
LO 636 POKE 764,255:IF PEEK(764)=255 THEN 636
OK 638 IF PEEK(764)=33 THEN GOTO 620
HB 640 END
CD 650 REM *** DISK SAVER *********
JB 660 DATA 104,169,0,141,11,3,133,240,141,4,3,169,1,141,1,3,141
PA 670 DATA 10,3,169,49,141,0,3,169,87,141,2,3,169,12,133,245,169
JO 680 DATA 64,133,241,165,240,141,4,3,165,241,141,5,3,24,165,240,105
EC 690 DATA 128,133,240,165,241,105,0,133,241,32,83,228,238,10,3,198,245
DD 700 DATA 208,223,96,-1
JN 710 REM *** BOOTSTUFFER ********
LB 720 DATA 67,12,0,6,6,6,76,34,6,162,0,160,0,169,0,145,251
IE 730 DATA 200,192,0,208,247,230,252,165,252,201,15,208,237,76
IE 740 REM FOR XL/XE VERS. {4 SPACES}THIS IS 177,197
BH 750 DATA 247,242
KN 760 DATA 0,169,35,133,251,169,6,133,252,169,196,141,200,2,141,198,2,162
DJ 770 DATA 3,32,210,8,162,4,32,210,8,32,117,8,162,20,32,210,8
PE 780 DATA 32,117,8,32,5,9,32,245,8,201,76,208,14,169,155,32,245
DL 790 DATA 8,32,143,9,32,7,7,76,9,6,201,83,208,48,169,155,32
EN 800 DATA 245,8,32,117,8,32,76,9,32,117,8,162,21,32,210,8,32
CB 810 DATA 117,8,32,5,9,201,67,208,6,32,64,10,76,60,6,201,68
CH 820 DATA 208,9,32,189,6,32,143,9,32,80,7,76,60,6,162,11,32
KD 830 DATA 210,8,32,131,9,76,60,6,162,15,32,210,8,162,16,32,210
LN 840 DATA 8,162,18,32,210,8,32,5,9,32,117,8,32,38,8,32,52
IG 850 DATA 8,96,32,252,6,32,161,6,173,1,12,133,253,32,149,9,32
AG 860 DATA 190,9,24,173,241,11,109,1,12,144,27,162,12,32,210,8,32
OG 870 DATA 5,9,201,89,240,12,169,1,141,241,11,238,242,11,32,76,9
ND 880 DATA 96,76,60,6,24,162,19,32,210,8,32,117,8,96,169,49,141
LA 890 DATA 0,3,169,1,141,1,3,96,32,165,8,162,1,32,210,8,32
LJ 900 DATA 5,9,72,32,245,8,32,117,8,104,24,233,47,133,247,32,64
OB 910 DATA 9,169,0,105,12,166,247,240,5,105,15,202,208,251,170,189,91
AL 920 DATA 11,141,10,3,232,189,91,11,141,11,3,32,117,8,96,162,255
FK 930 DATA 232,189,91,11,201,197,208,248,134,248,134,249,96,32,213,7,32
OJ 940 DATA 143,9,32,171,6,32,77,8,174,1,12,134,255,202,138,72,32
DI 950 DATA 65,8,104,170,224,1,208,244,162,15,32,210,8,162,3,32,210
HH 960 DATA 8,162,18,32,210,8,32,5,9,32,117,8,173,241,11,141,10
OF 970 DATA 3,206,10,3,173,242,11,141,11,3,169,87,32,52,8,166,255
FE 980 DATA 202,138,72,32,65,8,104,170,224,1,208,244,32,95,8,32,176
BJ 990 DATA 7,162,10,32,210,8,96,169,0,141,4,3,169,11,141,5,3
JM 1000 DATA 169,11,141,10,3,169,0,141,11,3,169,87,141,2,3,32,83
FO 1010 DATA 228,48,4,32,65,8,96,76,150,6,32,65,7,32,117,8,162
AK 1020 DATA 6,32,210,8,169,11,133,245,32,5,9,32,245,8,201,155,208
BA 1030 DATA 11,230,248,198,245,208,250,166,248,202,208,11,166,248,157,91,11
AM 1040 DATA 230,248,198,245,208,224,134,250,32,117,8,162,8,32,210,8,32
OJ 1050 DATA 137,9,32,117,8,32,5,9,201,89,240,10,32,117,8,165,249
NL 1060 DATA 133,248,76,216,7,96,32,252,6,169,0,141,10,3,141,11,3
LD 1070 DATA 169,82,96,141,2,3,169,11,141,5,3,169,128,141,4,3,32
IF 1080 DATA 123,8,32,83,228,48,1,96,76,150,6,166,250,232,24,173,241
AH 1090 DATA 11,157,91,11,232,173,242,11,157,91,11,96,24,173,241,11,109
FA 1100 DATA 1,12,141,241,11,144,8,173,242,11,105,0,141,242,11,24,96
OB 1110 DATA 169,155,32,245,8,96,24,173,4,3,105,128,141,4,3,144,3
EK 1120 DATA 238,5,3,24,238,10,3,32,145,8,96,173,10,3,201
JO 1130 REM FOR ENHANCED DENSITY CHANGE LINE 1140 TO DATA 16
HK 1140 DATA 208
HN 1150 DATA 208,12,173,11,3,201
HB 1160 REM FOR ENHANCED DENSITY CHANGE LINE 1170 TO DATA 4
BF 1170 DATA 2
DG 1180 DATA 208,5,162,2,32,210,8,96,162,0,134
EH 1190 DATA 246,169,0,133,245,138,72,189,91,11,32,245,8,104,170,232,230
MH 1200 DATA 245,165,245,201,13,208,237,138,72,32,117,8,104,170,232,232,230
MF 1210 DATA 246,165,246,201,10,208,216,96,232,134,240,162,0,189,122,10,232
AL 1220 DATA 201,155,208,248,198,240,208,244,189,122,10,232,134,254,32,245,8
AI 1230 DATA 201,155,240,4,166,254,208,239,96,162,11,142,66,3,162,0,142
AD 1240 DATA 72,3,142,73,3,76,86,228,162,80,169,3,157,66,3,169,38
NK 1250 DATA 157,68,3,169,3,157,69,3,169,4,157,74,3,169,0,157,75
JL 1260 DATA 3,32,86,228,169,7,157,66,3,169,0,157,72,3,157,73,3
CE 1270 DATA 32,86,228,133,241,169,12,157,66,3,32,86,228,165,241,96,201
KO 1280 DATA 10,176,5,201,0,144,1,96,76,60,6,173,242,11,201
HA 1290 REM FOR ENHANCED DENSITY CHANGE LINE 1300 TO DATA 4
BA 1300 DATA 2
AM 1310 DATA 240,24,56,169,0,237,241,11,133,253,32,149,9,32,190,9,162,7
KI 1320 DATA 32,210,8,162,0,76,210,8,56,169
KC 1330 REM FOR ENHANCED DENSITY CHANGE LINE 1340 TO DATA 16
HM 1340 DATA 208
OK 1350 DATA 237,241,11,133,253,32
MH 1360 DATA 149,9,32,190,9,162,7,32,210,8,162,9,76,210,8,169,64
EK 1370 DATA 141,198,2,96,169,244,141,198,2,96,169,196,141,198,2,96,169
AE 1380 DATA 48,133,225,133,226,133,227,56,165,253,233,100,144,6,133,253,230
KB 1390 DATA 225,16,246,56,165,253,233,10,144,6,133,253,230,226,16,243,230
```

```
IN 1400 DATA 227,198,253,208,250,96,165,225,32,245,8,165,226,32,245,8,165
BP 1410 DATA 227,32,245,8,162,14,32,210,8,96,162,16,169,3,157,66,3
AF 1420 DATA 169,4,157,74,3,169,128,157,75,3,169,0,157,68,3,169,0
FB 1430 DATA 133,234,133,236,169,6,157,69,3,169,12,133,235,32,86,228,230
NJ 1440 DATA 236,169,7,157,66,3,165,234,157,68,3,165,235,157,69,3,169
BK 1450 DATA 128,157,72,3,169,0,157,73,3,24,165,234,105,128,133,234,165
NF 1460 DATA 235,105,0,133,235,32,86,228,48,21,206,1,12,208,206,169,12
PM 1470 DATA 157,66,3,32,86,228,48,6,165,236,141,1,12,96,76,150,6
AD 1480 DATA 162,15,32,210,8,162,13,32,210,8,162,5,32,210,8,162,18
HH 1490 DATA 32,210,8,32,117,8,32,5,9,169,12,141,252,2,32,211,9
EO 1500 DATA 32,195,6,32,213,7,32,143,9,32,252,6,32,77,8,174,1
DN 1510 DATA 12,134,255,32,109,7,96,155,84,72,73,83,32,66,76,79,67
OJ 1520 DATA 75,155,83,69,76,69,67,84,73,79,78,63,155,198,213,204,204
CO 1530 DATA 155,160,196,201,211,203,173,203,210,193,205,160,155,98,121,32,82
DP 1540 DATA 65,78,68,89,32,66,79,89,68,155,208,210,197,211,211,160,208
KG 1550 DATA 204,193,217,155,78,65,77,69,32,63,155,32,76,69,70,84,32
ND 1560 DATA 79,78,155,83,85,82,69,63,32,89,47,78,155,212,200,201,211
NP 1570 DATA 160,196,201,211,203,155,196,207,206,197,155,194,207,207,212,160,197
GG 1580 DATA 210,210,207,210,155,211,205,193,204,204,197,210,160,191,155,66,79
CP 1590 DATA 79,84,32,84,65,80,69,155,45,83,69,67,84,79,82,83,155
FN 1600 DATA 32,32,73,78,83,69,82,84,155,194,207,207,212,173,173,196,201
LA 1610 DATA 211,203,155,155,200,201,212,173,173,210,197,212,213,210,206,155,208
LK 1620 DATA 210,207,199,210,193,205,173,207,235,161,155,76,41,79,65,68,32
NM 1630 DATA 79,82,32,83,41,65,86,69,32,63,155,67,41,65,83,83,32
BK 1640 DATA 79,82,32,68,41,73,83,75,32,63,155,48,46,197,77,80,84
NG 1650 DATA 89,32,32,32,32,32,32,48,32,49,46,197,77,80,84,89,32
ME 1660 DATA 32,32,32,32,32,49,32,50,46,197,77,80,84,89,32,32,32
LO 1670 DATA 32,32,32,50,32,51,46,197,77,80,84,89,32,32,32,32,32
MD 1680 DATA 32,51,32,52,46,197,77,80,84,89,32,32,32,32,32,32,52
MJ 1690 DATA 32,53,46,197,77,80,84,89,32,32,32,32,32,32,53,32,54
AB 1700 DATA 46,197,77,80,84,89,32,32,32,32,32,32,54,32,55,46,197
MP 1710 DATA 77,80,84,89,32,32,32,32,32,32,55,32,56,46,197,77,80
NJ 1720 DATA 84,89,32,32,32,32,32,32,56,32,57,46,197,77,80,84,89
IL 1730 DATA 32,32,32,32,32,32,57,32,13,0,0,0,0,0,0,-2
```

©

Requester Windows In Amiga BASIC

Tom R. Halfhill, Editor

Here's how to add your own custom requester windows to any Amiga BASIC program. Like dialog boxes on the Macintosh, requester windows allow your programs to flag errors or request confirmation before carrying out important functions. The routine is written for Microsoft Amiga BASIC, which is now being shipped in place of MetaComCo ABasiC and is available as an upgrade to early Amiga owners.

Amiga BASIC is the most powerful BASIC interpreter supplied with any personal computer on the market. Written by Microsoft, it combines in a single language almost every feature found in IBM PC Advanced BASIC plus Microsoft BASIC for the Macintosh. In fact, many IBM BASICA and Macintosh BASIC programs will run on the Amiga with minor modifications.

However, Amiga BASIC does lack two key statements found in Macintosh BASIC: DIALOG and BUTTON. Both are important for writing BASIC programs which retain the mouse-and-window user interface common to the Macintosh and Amiga Workbench. Fortunately, both commands can be simulated fairly easily with Amiga BASIC's WINDOW and MOUSE statements.

In Macintosh BASIC, the DIALOG command lets a program open a *dialog box* (a small window) like those displayed by the Macintosh's operating system whenever the user must choose between two or more options. Dialog boxes also flag errors and alert users when they're about to activate a function that has irreversible consequences—such as quitting a program without saving the data on disk. For example, if the user pulls down a menu and selects Quit, a dialog box might open up and ask, "Quit program? (Data file not saved.)" Below this message is usually a pair of small boxes or circles called *buttons* which might be labeled OK and CANCEL. Pointing and clicking the mouse on the OK button exits the program; pointing and clicking on the CANCEL button cancels the Quit function and returns to the main program so the user can save his data if desired.

In Amiga BASIC, the DIALOG and BUTTON commands must be simulated by a routine that uses the WINDOW and MOUSE statements. For greater convenience, the routine can be written as a *subprogram*, another advanced feature included in Amiga BASIC. Subprograms are similar to subroutines, except they can have *local variables*. These are variables which are independent of the main program. For instance, if your main program uses a variable X for some purpose, a subprogram can also use a variable named X and it is treated as a separate variable. If the subprogram changes the value of its variable X, the main program's variable X is unaffected, and vice versa. On the other hand, a subprogram can also specify *shared variables*, sometimes known as *global* variables—those which are common to both the subprogram and the main program.

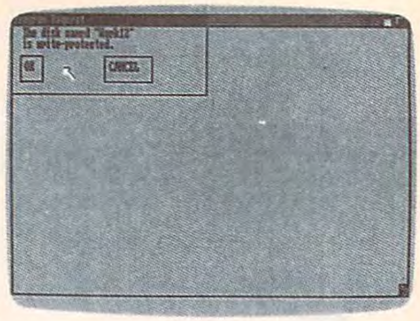
A major advantage of subprograms is that you can build up a library of useful routines on disk and add them to any new programs you write. This saves you the trouble of writing the same subprograms again and again. Although you can do the same thing with ordinary BASIC subroutines, there's always the chance that a subroutine variable might conflict with an identically named variable in your main program. Since subprogram variables are local, you're freed from this worry. Subprograms are truly programs within a program.

The Requester Subprogram

On the Amiga, dialog boxes are called *requesters*. Probably the most frequently encountered requester is the one that pops up when the Amiga asks you to insert a different disk. For the sake of consistency, an Amiga requester generally appears as a small window in the upper-left corner of the screen, has a title bar labeled System Request, has two or three buttons, does not have a resizing gadget or close gadget, and cannot be moved elsewhere on the screen.

The "Requester Window Subprogram" listed below duplicates most of these features. It creates a window that appears in the upper-left corner of the screen (or up to the full width of the screen in low-resolution modes); the window has a title bar labeled Program Request (to distinguish it from System Request windows); there is no resizing gadget or close gadget; and the window cannot be moved elsewhere on the screen. Unlike system requesters, this requester always displays two buttons, and they're always labeled OK and CANCEL.

The subprogram lets you display one or two lines of your own text in the Program Request window. The maximum number of characters allowed in each line depends on whether the Amiga has been set for 60- or 80-column text with the Preferences tool. If Preferences is set for 60 columns, each requester line can be up to 31 characters long. If Preferences is set for 80 columns, each line can be up to 39 characters. (You can adjust the subprogram for either mode by changing a single program statement; see the remarks in the listing.) If you try to display a line of text which exceeds these limits, the subprogram leaves off the extra characters. Since you won't know how Preferences is set if you're writing programs that might be used by other people, it's safest to assume 60 columns and restrict each line of your message to 31 characters.



A short subprogram lets you quickly and easily add custom requester windows to your own Amiga BASIC programs.

Opening a Program Request window is this simple:

```
request1$="This is the first line."
request2$="This is the second line."
CALL Requester
```

The two lines of your message are defined in the string variables *request1$* and *request2$*, and the CALL statement runs the subprogram (similar to GOSUB). The subprogram opens the requester window and waits for the user to click on the OK or CANCEL button. Clicks outside the buttons are ignored, although a click outside the requester window itself deselects it as the active window. It can be reselected, of course, by clicking within the window.

If the user clicks on OK, the subprogram returns a value of 1 in the variable *answer*. If the user clicks on CANCEL, *answer* equals 0. In either case, the subprogram closes the requester window after the button click and passes control back to the line following the CALL Requester statement. By testing *answer*, your program can branch to different routines to handle the user's response as required.

Requester Window Subprogram

```
RequesterSub:
SUB Requester STATIC
 SHARED request1$,request2$,answer:' Global variables.
 ' Add screen parameter if needed to next line.
 WINDOW 2,"Program Request",(0,0)-(311,45),16
 ' Following lines truncate prompts if too long.
 ' If Preferences is set for 60 columns,
 ' use maxwidth=INT(WINDOW(2)/10) for next line;
 ' otherwise use maxwidth=INT(WINDOW(2)/8).
 maxwidth=INT(WINDOW(2)/10)
 request1$=LEFT$(request1$,maxwidth)
 request2$=LEFT$(request2$,maxwidth)
 PRINT request1$:PRINT request2$
 ' This section draws buttons.
 LINE (12,20)-(50,38),1,b
 LINE (152,20)-(228,38),1,b
 LOCATE 4,1:PRINT PTAB(20);"OK";
 PRINT PTAB(160);"CANCEL"
 ' This section gets input.
 reqloop:
  WHILE MOUSE(0)=0:WEND:' Wait for button click.
  m1=MOUSE(1):m2=MOUSE(2)
  IF m1>12 AND m1<50 AND m2>20 AND m2<38 THEN
   answer=1:' OK was selected.
   LINE (12,20)-(50,38),1,bf:' Flash OK box.
   WHILE MOUSE(0)<>0:WEND:' Wait for button release.
   WINDOW CLOSE 2:EXIT SUB
  ELSE
   IF m1>152 AND m1<228 AND m2>20 AND m2<38 THEN
    answer=0:' CANCEL was selected.
    LINE (152,20)-(228,38),1,bf:' Flash CANCEL box.
    WHILE MOUSE(0)<>0:WEND:' Wait for button release.
    WINDOW CLOSE 2:EXIT SUB
   ELSE
    GOTO reqloop
   END IF
  END IF
 GOTO reqloop
 END SUB
```

Hints For Use

Here's an example. Suppose your BASIC program sets up a Project menu with a Quit selection (a consistent feature in Amiga software). When your MENU statement detects that Quit has been selected, it can GOSUB Quit:

```
Quit:
 MENU OFF:CLS
 request1$="Quit program?"
 request2$="(OK exits to Workbench or
   CLI.)"
 CALL Requester
 IF answer=0 THEN RETURN
 SYSTEM
```

If the user selects Quit by accident or changes his mind, he can click on CANCEL and no harm is done—the Quit routine merely RETURNs. Otherwise, a click on OK stops the program and exits BASIC with the SYSTEM command. Of course, you could also include a check to see if any data created with the program has been saved, and if necessary prompt the user to save it before quitting.

There are only two more details to keep in mind when using the requester routine. First, the WINDOW statement near the beginning of the subprogram opens WINDOW 2. If there's a chance that your program might already have two or more windows open when the requester is called, change this statement to WINDOW 3, or WINDOW 4, or whatever is necessary to avoid a conflict.

Second, the WINDOW statement defaults to the primary (Workbench) screen. That means the requester window always pops up on the primary screen. If your main program creates a secondary screen with the SCREEN statement, you'll want the requester window to appear on that screen instead of the primary screen. Otherwise, the requester will be invisible. To make the requester window appear on your program's secondary screen, append the screen's number to the WINDOW statement.

For instance, if your program creates a secondary screen with a statement such as this:

```
SCREEN 1,320,200,1,1
```

change the WINDOW statement in the requester subprogram as follows:

```
WINDOW 2,"Program Request",(0,0)-
  (311,45),16,1
```

This makes sure the requester will be visible. ©

Softkeys For Atari BASIC

Raymond Citak

This labor-saving utility adds automatic line numbering and 19 preprogrammed "soft" keys to your Atari computer. Even better, the soft key assignments are compatible with COMPUTE!'s *"Automatic Proofreader." For any Atari 400/800, XL, or XE computer with at least 48K RAM.*

If you write your own BASIC programs or enter the programs listed in COMPUTE!, you'll welcome any utility that cuts down on your typing time. "Softkeys For Atari BASIC" does exactly that—it gives you automatic line numbering and 19 preprogrammed soft keys that enter an entire BASIC word with just one keystroke. And there are two extra soft keys you can program for your own use.

Type in the program below and save it on disk or tape before running it for the first time. If you plan to use it along with the "Automatic Proofreader" to type in a COMPUTE! program, you should load and run Automatic Proofreader at this point. (Of course, this program works on its own, even if you're not using the Proofreader; but when the two are used together, you must install the Proofreader first.)

Now load and run the Softkeys program. It begins by asking you for the starting line number of the program you'll be typing in. Enter that number and press RETURN. Now you're asked to enter the increment (how much the line number increases between one line and the next). Most programs published in COMPUTE! are numbered in increments of ten, but you should always check the program listing to make sure. This number can be changed if the listing later changes to a different increment or skips some line numbers.

Automatic Line Numbering

After you enter this information, Softkeys installs its machine language portion in memory, deletes its BASIC portion, and leaves the computer ready for you to use. On the line below the READY prompt you'll see the first line number followed by a space. Type in the first line from the program listing, then press RETURN to enter it in memory. The computer automatically prints the next line number and waits for you to enter the next line.

If the computer detects an error in the line, it prints the line again and shows where the error occurred. To retype the line, simply press SHIFT–DELETE, type in the correct line number and reenter the line. If you prefer, you can move the cursor back to the old line as usual, correct it, and press RETURN again. Just as in normal screen editing, the cursor can be anywhere on the line when you press RETURN.

The SHIFT–DELETE key combination also lets you perform several other tasks. If the program listing skips line numbers, press SHIFT–DELETE, then enter the new line number and continue typing as before. You can also use SHIFT–DELETE to enter any BASIC command from direct mode. For example, you may want to continue typing a program that you've partially entered and saved. Run Softkeys and answer the prompts as you did when you began typing the program. When the READY prompt comes back, press SHIFT–DELETE, then enter the command you would ordinarily use to load the program. After the program loads, the computer finds the last line number used in the program and automatically continues numbering from that point.

If you press SHIFT–DELETE and then change your mind, press RETURN without typing anything else: The correct line number reappears.

At times you'll need to change the line number increment midway through the program. To do this, press BREAK to disable Softkeys, then enter a USR statement in this format:

U=USR(39300,*line number,increment*)

The *increment* parameter specifies the desired new increment value, which takes effect *after* the next line is entered. The *line number* parameter must be included, but it has no effect. The program continues with the line number in use before the USR. For example, if you've been numbering lines by tens until you reach line 500 and wish to switch to increments of five, get a blank line by pressing SHIFT–DELETE when the computer prints 500, then enter the statement U=USR(39300,100,5). After this, the prompt for line 500 returns, but the next line number is 505.

Softkey Assignments

A *softkey* is a preprogrammed key combination that lets you print a complete BASIC command with a single keystroke. This program creates a number of softkeys that let you enter commonly used commands quickly and easily. The softkeys are all entered by pressing CTRL along with another key. The accompanying table lists all of the built-in softkeys.

When Softkeys is active, you can enter any of the 19 keywords shown in the table by pressing CTRL along with the indicated key. If you press CTRL-F, the computer prints FOR, and so on. This saves typing time and helps eliminate errors (the computer never types PRIMT instead of PRINT, for example). Note that STRIG and STICK both include a left parenthesis.

Atari Softkeys

Softkey	Command
CTRL-A	GRAPHICS
CTRL-C	COLOR
CTRL-D	DATA
CTRL-E	PEEK
CTRL-F	FOR
CTRL-G	GOTO
CTRL-I	INPUT
CTRL-K	STICK(
CTRL-L	LOCATE
CTRL-N	NEXT
CTRL-O	POKE
CTRL-P	POSITION
CTRL-R	READ
CTRL-S	SOUND
CTRL-T	STRIG(
CTRL-U	GOSUB
CTRL-W	DRAWTO
CTRL-?	PRINT
CTRL-RETURN	RETURN

Though 19 softkeys are built into the program, you can add two more of your own. To do this, you'll need to supply new values in the DATA statements in lines 1100 and 1180. Each line contains 10 values. The first value is the keyscan code generated when you press a key. Before you can program your own softkey, you need to know the keyscan code for the key combination you want to use.

For example, let's say you want to program the CTRL-V key combination to print the keyword SAVE followed by a quotation mark (SAVE"). To find the keyscan code for the CTRL-V combination (or any key combination), type the following statements in direct mode (without a line number) and press RETURN:

```
FOR J=1 TO 1E9:PRINT PEEK(764)
  :NEXT J
```

The computer prints the keyscan code for whatever key or key combination is currently pressed. Try pressing different keys to see the numbers change. The number that appears when you press the desired combination is the keyscan code you need to use. In this case CTRL-V generates the keyscan value 144, so you should replace the first value in line 1100 with 144.

Encoding The Softkey

The next nine values in line 1100 represent the ATASCII values of the characters the computer should print when you press the designated softkey. The ATASCII values for the SAVE" character sequence are 83, 65, 86, 45, 34. Including the keyscan code, that comes to 6 values: Since you don't need the last four values in that line, make them all zeros (this DATA statement *must* have exactly ten values, even if you don't need to use all ten). When you're finished, line 1100 should look like this:

```
1100 DATA 144,83,65,86,45,34,0,0,0,0
```

To use your new softkey, simply rerun the program and try it out. By repeating the process, you can change line 1180 to add another, giving you a total of 21 softkeys. When programming a new softkey, note that you must include a space (character 32) if you want the cursor to move right one space after printing a keyword.

Occasionally, a program requires you to type in a character that requires a CTRL-key combination already used by Softkeys. Disable Softkeys by pressing BREAK, then enter the line. After that's done, you can reactivate the utility with a USR command as described above.

The machine language portion of this program resides in high memory just below the display list in GRAPHICS 0. Use caution if you run a program and later issue the USR command to activate this utility. If the previous program used high memory for any purpose, the computer may crash.

Once you're satisfied with all the softkey assignments, you may want to convert the machine language portion of this program to a binary object file on disk. To do this, first run the BASIC portion, exit to DOS, select the Binary Save option, and save memory from $9984–$9BF1.

Softkeys For Atari BASIC

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

```
BI 10 DIM A$(3):? CHR$(125):? :? "POKING DATA... PLEASE WAIT"
PE 20 FOR J=39300 TO 39921:READ A:POKE J,A:NEXT J
IH 30 ? CHR$(125)
CM 40 TRAP 40:POSITION 2,2:? "WHAT LINE NUMBER TO START WITH";:INPUT LN
LH 50 TRAP 50:POSITION 2,4:? "WHAT INCREMENT";:INPUT INC
KE 60 IF LN>=32767 OR INC>=32767 THEN 30
LC 70 IF INC<=0 OR LN<0 THEN 30
GP 80 TRAP 40000:? CHR$(125):?
FO 90 IF PEEK(1614)=93 AND PEEK(1615)=6 THEN A$="ARE":GOTO 110
BJ 100 A$="IS":GOTO 120
MB 110 ? "THE AUTOMATIC PROOFREADER PROGRAM AND"
PA 120 ? "THE AUTONUMBER PROGRAM WITH ";CHR$(34);"SOFT";CHR$(34)
MP 130 ? "KEYS ";A$;" NOW READY FOR YOUR INPUT."
DI 140 ? "USE BREAK TO DISABLE THE PROGRAM."
MG 150 ? "USE U=USR(39300,ln,inc) TO ENABLE."
GD 160 U=USR(39300,LN,INC):NEW
BD 170 DATA 104,104,141,223,153,104
BE 180 DATA 141,222,153,104,141,221
BM 190 DATA 153,104,141,220,153,173
FJ 200 DATA 36,2,133,208,173,37
FJ 210 DATA 2,133,209,169,5,133
FM 220 DATA 194,133,206,173,8,2
FJ 230 DATA 141,233,154,173,9,2
CI 240 DATA 141,234,154,169,193,141
AA 250 DATA 8,2,169,154,141,9
FM 260 DATA 2,174,6,228,232,142
NA 270 DATA 188,154,174,7,228,142
AD 280 DATA 189,154,169,224,141,54
CO 290 DATA 2,169,153,141,55,2
FN 300 DATA 160,3,162,154,169,7
MF 310 DATA 32,92,228,96,0,0
FI 320 DATA 0,0,164,208,166,209
GK 330 DATA 169,7,32,92,228,173
FK 340 DATA 233,154,141,8,2,173
GC 350 DATA 234,154,141,9,2,169
CC 360 DATA 0,133,17,141,255,2
II 370 DATA 141,240,2,133,77,104
DJ 380 DATA 64,8,72,152,72,138
GD 390 DATA 72,165,85,201,2,208
EM 400 DATA 25,173,242,2,201,12
JM 410 DATA 208,18,169,23,229,84
JE 420 DATA 48,12,165,194,201,93
IJ 430 DATA 208,6,165,206,240,11
CJ 440 DATA 198,206,104,170,104,168
JJ 450 DATA 104,40,76,98,228,160
GA 460 DATA 1,177,136,16,13,173
BO 470 DATA 222,153,133,212,173,223
MB 480 DATA 153,133,213,24,144,55
CN 490 DATA 165,136,133,204,165,137
LG 500 DATA 133,205,160,1,177,204
MG 510 DATA 48,26,136,177,204,133
BG 520 DATA 212,200,177,204,133,213
OD 530 DATA 200,177,204,24,101,204
LF 540 DATA 133,204,165,205,105,0
PC 550 DATA 133,205,208,224,24,165
BJ 560 DATA 212,109,220,153,133,212
CD 570 DATA 165,213,109,221,153,133
PB 580 DATA 213,32,170,217,165,212
LL 590 DATA 41,15,133,206,230,206
IA 600 DATA 162,0,181,213,41,240
PC 610 DATA 208,4,224,0,240,9
NP 620 DATA 74,74,74,74,9,48
LP 630 DATA 32,183,154,181,213,41
DG 640 DATA 15,9,48,32,183,154
CN 650 DATA 232,228,206,208,223,169
GH 660 DATA 32,32,183,154,169,5
MI 670 DATA 133,194,133,206,76,40
MP 680 DATA 154,168,138,72,152,32
MJ 690 DATA 0,0,104,170,96,8
GE 700 DATA 72,138,72,152,72,44
OO 710 DATA 9,210,16,22,162,0
JG 720 DATA 189,16,155,205,9,210
HJ 730 DATA 240,21,160,0,232,200
OO 740 DATA 192,11,208,250,224,231
CI 750 DATA 208,236,104,168,104,170
PA 760 DATA 104,40,76,0,0,232
GJ 770 DATA 189,16,155,240,6,32
PN 780 DATA 183,154,24,144,244,162
MB 790 DATA 126,142,31,208,173,11
NP 800 DATA 212,205,11,212,240,251
OJ 810 DATA 202,202,16,241,104,168
LF 820 DATA 104,170,104,40,104,64
FA 830 DATA 146,67,79,76,79,82
FG 840 DATA 32,0,0,0,0,186
NI 850 DATA 68,65,84,65,32,0
FO 860 DATA 0,0,0,0,189,71
KG 870 DATA 79,84,79,32,0,0
JO 880 DATA 0,0,0,128,76,79
NP 890 DATA 67,65,84,69,32,0
MO 900 DATA 0,0,138,80,79,83
BE 910 DATA 73,84,73,79,78,32
BC 920 DATA 0,168,82,69,65,68
OH 930 DATA 32,0,0,0,0,0
FC 940 DATA 190,83,79,85,78,68
FE 950 DATA 32,0,0,0,0,173
AM 960 DATA 83,84,82,73,71,40
FG 970 DATA 0,0,0,0,141,73
NM 980 DATA 78,80,85,84,32,0
JK 990 DATA 0,0,0,136,80,79
IH 1000 DATA 75,69,32,0,0,0
PG 1010 DATA 0,0,170,80,69,69
EJ 1020 DATA 75,40,0,0,0,0
DO 1030 DATA 0,163,78,69,88,84
BA 1040 DATA 32,0,0,0,0,0
HA 1050 DATA 166,80,82,73,78,84
HJ 1060 DATA 32,0,0,0,0,133
DM 1070 DATA 83,84,73,67,75,40
CH 1080 DATA 0,0,0,0
MN 1090 REM CHANGE NEXT 10 BYTES TO INSERT YOUR OWN "SOFT" KEY.
AK 1100 DATA 173,83,84,82,73,71,40,0,0,0
AG 1110 DATA 0,139,71,79
MD 1120 DATA 83,85,66,32,0,0
PK 1130 DATA 0,0,174,68,82,65
MM 1140 DATA 87,84,79,32,0,0
DE 1150 DATA 0,140,82,69,84,85
IM 1160 DATA 82,78,32,0,0,0
MM 1170 REM CHANGE NEXT 10 BYTES TO INSERT YOUR OWN "SOFT" KEY.
OG 1180 DATA 168,82,69,65,68,32,0,0,0,0
NM 1190 DATA 0,191
DB 1200 DATA 71,82,65,80,72,73
CL 1210 DATA 67,83,32,0,184,70
NC 1220 DATA 79,82,32,0
```

©

BASIC Sound On The Atari ST

Almost any music or sound effect can be created with the WAVE and SOUND commands in Atari ST BASIC. This article shows how to get started with ST sound and includes sample sound effects and a simulated piano program. The article is an excerpt from the newly released COMPUTE!'s ST Programmers Guide *(by the editors of* COMPUTE!, *$16.95).*

The Atari 520ST contains a General Instruments sound chip that has three voices (sound channels) and a range of eight octaves. In fact, it's the same sound chip found in the MSX-standard computers sold in Japan and Europe. The chip's best feature is that it supports *envelope-oriented* sound—you can create a sound by defining the shape of its envelope. This allows considerable flexibility when designing sound effects and musical instrument tones. However, for programmers, it also requires more work than the SOUND command found in Atari BASIC for the eight-bit computers.

There are two sound commands in ST BASIC: WAVE and SOUND. WAVE controls the make-up of the sound:

WAVE *sound type, envelope, shape, period, delay*

Some of these parameters require values that toggle certain bits to activate certain functions. If you're not familiar with bit manipulation, refer to Figure 1. The first step is to decide which function(s) you want to select. Then add up the bit values—*not the bit numbers*—corresponding to those functions. The resulting number is what you use for that particular parameter in the WAVE statement.

For instance, the first parameter, *sound type,* controls whether a voice is set to noise, tone, or both. Bits 0–2, when set, turn on tone output for voices 1–3. Bits 3–5, when set, select noise output for the three voices. Both tone and noise may be turned on at the same time. Here are some example bit values:

WAVE 1—turns on tone for voice 1
WAVE 3—turns on tone for voice 1 and voice 2
WAVE 8—turns on noise for voice 1
WAVE 15—turns on tone and noise for all three voices

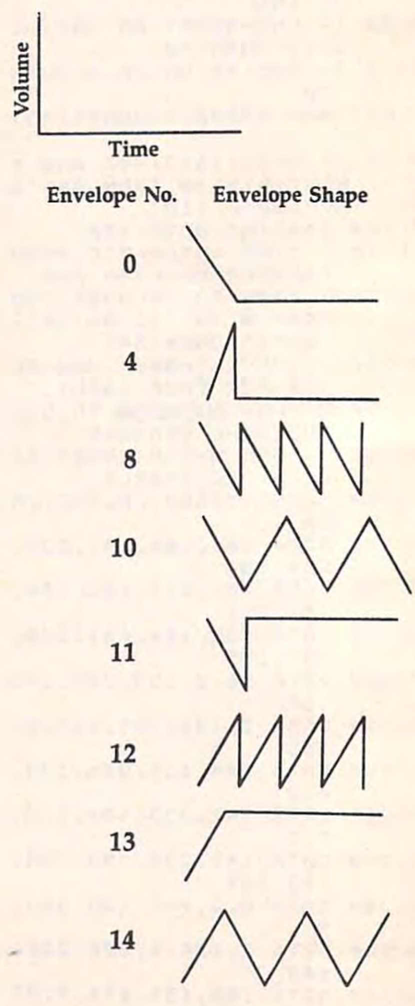
Bits 0–2 of *envelope* determine which of the three voices is controlled by the envelope generator. If a bit is set, its corresponding voice is controlled by the envelope generator.

The third parameter, *shape,*

Figure 1: Bit Values

Bit numbers →	7	6	5	4	3	2	1	0
Bit values →	128	64	32	16	8	4	2	1

Figure 2: Available Envelopes



controls the way the sound's volume rises and falls. Figure 2 gives the possible values for this parameter and shows the shape of the subsequent sound.

Each of the envelope-shape drawings is a graph of volume over time. Take a close look at envelope zero and imagine what kind of sound it would make. The first thing to notice is that as soon as the sound begins, volume is at maximum. As time passes, the volume slowly fades away until it reaches zero. This type of sound is made by a piano. The hammer hits the string and almost immediately the volume reaches its maximum. The vibration of the string continues and slowly decays.

As you can see from Figure 2 envelopes 8, 10, 12, and 14 are repetitive. The sound continues to surge and fall long after the WAVE command is given.

The *period* parameter sets the period of the envelope, which is how fast the sound cycles. The larger the period, the longer the note takes to repeat. The last parameter, *delay*, controls the amount of time the program waits before executing the next BASIC command. Period is measured in one-fiftieth of a second increments. To hear a couple of interesting sound effects produced by WAVE, type in Program 1, "Helicopter," and Program 2, "Ding."

SOUNDing Off And On

The other music command, SOUND, turns on one of the voices for a specified duration. Its syntax is:

SOUND *voice, volume, note, octave, duration*

Voice selects which voice you want to turn on (1–3). *Volume* can be any number from 0 (off) to 15 (loudest). *Note* is a number from 1 to 12 and corresponds to the 12 notes in a scale (C, C#, D, D#, E, F, F#, G, G#, A, A#, B). The *octave* ranges from 1 (lowest) to 8 (highest). *Duration* can be any number from 0–65535. Each increment corresponds to one-fiftieth of a second.

Program 3, "Piano," uses the SOUND and WAVE commands to simulate a piano. Although it's intended as a sound demonstration, it also shows how to use other techniques, such as graphics and reading the mouse from BASIC.

You can run Program 3 in any graphics mode. When the piano keyboard appears on the screen, point to any key with the mouse and press the mouse button. The corresponding note is played.

Before typing in and running Program 3, you must make sure there's enough free memory available in BASIC. At this writing (mid-December), all 520STs were being shipped with the operating system (TOS) on disk. Later versions of the 520ST may be shipped with TOS in Read Only Memory (ROM). Until then, however, TOS must be loaded from disk into Random Access Memory (RAM). Because of the large amount of memory this requires, only a small area of storage remains for BASIC programs. When TOS and BASIC are loaded into a 520ST with 512K RAM, only about 5K is free for BASIC—enough for a program about 20 lines long. To check how much memory is available, load BASIC and type PRINT FRE(0).

Fortunately, there is a way to increase the amount of free memory by 32K. Normally, when windows are manipulated, the previous screen is saved in memory because part of it may be covered by a window and have to be restored later. The technique of saving the screen to memory is called *buffered graphics*. Although it can be quick and convenient, it requires 32K of memory to hold the screen.

If the buffered graphics option is turned off, 32K of memory is freed for BASIC. Click on Buf Graphics in the Run menu to toggle the buffered graphics on and off. This should increase free memory to 37986 bytes for BASIC programs.

Building The Piano

Let's trace through Program 3 to see how it works. Line 10 dimensions two arrays, B% and W%. These hold the note values of the black and white keys, respectively.

Next, the subroutine DRAWSCREEN is called. ST BASIC allows the use of labels instead of line numbers for many of its commands that need to make a reference to a line, like GOTO and GOSUB. Whenever you use a label in a line, make sure it is separated from the rest of the line with a colon.

The DRAWSCREEN subroutine (beginning at line 150) draws the piano keyboard. The first command of DRAWSCREEN sets the color of all screen output to black. Using only a single color for drawing ensures that the program will work in all graphics modes. The COLOR command also sets the fill pattern to solid. FULLW expands the window to full size, and CLEARW clears it.

The remaining commands of the DRAWSCREEN subroutine create the piano keyboard. Since there is no box drawing command in ST BASIC, we will simulate one using the LINEF command and FILL commands. LINEF draws a line between any two pairs of coordinates. The syntax is:

LINEF *xcoord1, ycoord1, xcoord2, ycoord2*

Next, line 20 calls the subroutine SETARRAY, which reads the note values of the black and white keys into the integer arrays B% and W%.

Reading The Mouse From BASIC

Now that the screen is set up and the arrays have been initialized, it's time to read the position of the mouse and check if the mouse button is pressed. This is done in the subroutine labeled READMOUSE at line 90. There is no BASIC command to read the mouse, so we must use one of the computer's Virtual Device Interface (VDI) routines. VDI routines are part of the computer's operating system.

The procedures necessary to call VDI routines are beyond the scope of this article, but basically involve POKEing various parameters into certain memory locations. These memory locations are not absolute addresses—instead, they're accessed via a reserved variable in ST BASIC named CONTRL. The ST automatically assigns an address to this variable which corresponds to the entry point into the VDI. By POKEing values into offsets from this address, various VDI routines can be executed.

The VDI routine for reading the position of the mouse and determining whether the mouse button is pressed has an opcode of 124, so we POKE CONTRL,124. We must tell the VDI routine that no other parameters are being passed, so two more POKEs are necessary: POKE CONTRL+2,0 and POKE CONTRL+6,0. Now we can call the VDI routine to read the mouse.

To read the horizontal and vertical position of the mouse, PEEK PTSOUT and PTSOUT+2, respectively. If the mouse button is pressed, PEEKing INTOUT will give a value of 1; otherwise, a zero is returned. (PTSOUT and INTOUT, like CONTRL, are also reserved variables for accessing VDI routines.)

The main loop of the piano program (line 30) simply waits until a mouse button is pressed. Once the button has been pressed, the vertical coordinates are checked to see if they are in the range of the piano keyboard (line 50). Then the vertical position is used to determine whether the key pressed is black or white (lines 50 and 60). If a black key is pressed, the note is calculated using the array B%; otherwise, the array W% is used.

Line 70 breaks the note value down into note and octave and then, using the SOUND command, plays the note.

Line 80 sets the envelope shape to zero. This creates a note with a similar shape to a piano's envelope. Program execution is then sent back to the main loop to check the mouse button again and SOUND another note when it is pressed.

Program 1: Helicopter

```
10 for a=1000 to 643 step -2
20 wave 8,3,14,a
30 for td=1 to 100:next:next
40 for a=643 to 1000 step 2
50 wave 8,3,14,a
60 for td=1 to 100:next:next
70 sound 1,0:sound 2,0
```

Program 2: Ding

```
10 for a= 1 to 12
15 sound 1,15,a,7
20 wave 1,1,14,5,1
30 for td=1 to 100:next:next
40 goto 10
```

Program 3: Piano

```
10 dim b%(16),w%(16)
20 gosub DRAWSCREEN:gosub
   SETARRAY
30 gosub READMOUSE:if button=0
   then 30
40 if y<70 or y>120 then 30
50 if y<100 then n=b%((x-16)/16.25)
60 if y>99 then n=w%((x-4)/16.25)
70 sound 1,15+15*(n=0),n-12*int((n-1)
   /12),3+int((n-1)/12)
80 wave 1,1,0,10000:goto 30
90 READMOUSE: poke contrl,124
100 poke contrl+2,0:poke contrl+6,0
110 vdisys(0)
120 x=peek(ptsout):y=peek(ptsout+2)
130 button=peek(intout)
140 return
150 DRAWSCREEN: color 1,1,1,1,1:fullw
   2:clearw 2
160 for a=50 to 100 step 50
170 linef 20,a,280,a:next
180 for a=20 to 280 step 16.25
190 linef a,50,a,100:next
200 for a=1 to 11:read s
210 gosub 250:next:return
220 data 32.5,48.75,81.25,97.5
230 data 113.75,146.25,162.5
240 data 195,211.25,227.5,260
250 linef s,50,s,78
260 linef s,78,s+8,78
270 linef s+8,78,s+8,50
280 fill s+1,51:fill s+5,51
290 return
300 SETARRAY: for a=1 to 16:read
   w%(a):next
310 for a=1 to 16:read b%(a):next
320 return
330 data 1,3,5,6,8,10,12,13
340 data 15,17,18,20,22,24
350 data 25,27
360 data 2,4,0,7,9,11,0,14,16
370 data 0,19,21,23,0,26,0
```

©

INSIGHT: Atari

Bill Wilkinson

Atari Character Codes

Last month's discussion about where and how to place things in memory served as a good lead-in to this month's topic: character codes. If you've read the heftier reference material, including COMPUTE! Book's *Mapping the Atari,* you may have discovered that your eight-bit Atari computer actually uses three different types of codes to represent the various characters (letters, numbers, punctuation, graphics symbols) it works with. All of these codes assign a unique number to represent each character, but the three codes are incompatible with each other because they use different numbering schemes.

The most commonly encountered code is called *ATASCII,* which stands for ATari-version American Standard Code for Information Interchange. Except for the so-called *control characters*—such as carriage return, tab, and so on—ATASCII is compatible with standard ASCII. (Why Atari chose to modify the standard is anyone's guess.) ATASCII is the character code used by PRINT, INPUT, CHR$(), ASC(), and most external devices such as printers and modems.

For example, in ATASCII (and ASCII), the code for uppercase A is 65. You can verify this in BASIC:

```
PRINT CHR$(65)
```

or

```
PRINT ASC("A")
```

Virtually every Atari BASIC book (even Atari's own) shows the character represented by each ATASCII code. You can also run Program 1 below to display each character and its code. (Press CTRL-1 to pause and continue the display.)

Screen Codes

The second character code found in your Atari is the *keyboard code.* The keyboard code for any character is actually the value read from a hardware register in memory when the key for that character on the keyboard is pressed. Program 2 below lets you find the keyboard code for any character. Just for fun, try some of the keys or key combinations which don't normally produce characters, such as CTRL-SHIFT-

CAPS). Neat, huh?

Finally: *screen codes.* This term refers to the byte value you must store in memory to display the desired character on the screen. "What?" you ask, "How do those differ from the ATASCII codes?" After all, to put the string BANANA PICKLE PUDDING on the screen, all it takes is a simple BASIC statement:

```
PRINT "BANANA PICKLE PUDDING"
```

And besides, aren't the characters in quotes supposed to be ATASCII codes? Good questions. Now for some complicated answers.

Actually, if the original Atari designers had thought just a little harder and added just a few more logic gates to the thousands already in the ANTIC and GTIA chips, ATASCII and screen codes could have been one and the same. It's similar to the mistake of making ATASCII incompatible with ASCII. Sigh. But we're stuck with what we've got, so let's figure out how it works.

For starters, consider GRAPHICS 1 and GRAPHICS 2, the large-size character modes. You may have noticed that in either of these modes you can display only 64 different characters on the screen. Now, if you recall last month's demo programs, note that we can specify the *base address* of the character set. That is, we can tell ANTIC where the character set starts by changing the contents of memory location 756 (which is actually a *shadow register* of the hardware location which does the work—see *Mapping the Atari* for more on this).

In a sense, the ANTIC chip is fairly simplistic. When it finds a byte in memory which is supposed to represent a character on the screen, it simply adds the value of that byte (multiplied times eight, because there are eight bytes in the displayable form of a character) to the character set base address. This points to the memory address for that particular character. Except...well, let's get to that in a moment.

Exception To The Rule

Because we want GRAPHICS 1 and 2 (with their limited sets of 64 different characters) to display numbers and uppercase letters (omitting lowercase letters and graphics), for these two modes it makes sense that the character set starts with the dot representation of the space character and ends with the underline—codes 32 through 95, respectively.

But why are these 64 characters the only ones available in GRAPHICS 1 or 2? Because the upper two bits of a screen byte in these modes are interpreted as *color* information, *not* as part of the character (see the modification to Program 3 below). So only the lower six bits choose a character from the character set. Six bits can represent only 64 possible combinations, which is why these modes can display only 64 characters. Bit pattern 000000 becomes a space, 100101 is an E, and 111111 becomes an underline, and so on.

When you use GRAPHICS 0 (normal text), however, there is a strange side effect. In this mode, only the single upper bit is the color bit (actually, it's the inverse video bit). This leaves 7 bits to represent a character, so we can have values from 0 to 127 decimal (0000000 to 1111111 binary, \$00 to \$7F hex). Again, this value—after being multiplied by eight—is added to the value of the character set base address. But which numbers in that 0 to 127 range represent which characters?

Well, we already know what the first 64 characters are—since the Atari's hardware limitations dictate that they *must* be the same as in modes 1 and 2. So the next 64 are the other characters. Program 3 illustrates how the ATASCII character set is linked to the screen set. Note how all the characters are presented twice, once in screen code (i.e., character ROM) order and once in ATASCII order. For some additional fun and info on modes 1 and 2, change line 10 to GRAPHICS 1. (Do not change it to GRAPHICS 2 unless you put a STOP in line 65 after the first FOR-NEXT loop.) Do you see what I mean about the upper two bits being color information?

Now you know why there are three different character codes used in your computer. How can you take advantage of this information? Well, if you combine this knowledge with the programs I presented last month, you could invent your own character set and design a word processor for some foreign language. (If you come up with a good Cyrillic character set, let me know.)

Actually, if you own an XL or XE machine, you have a second character set already built in. Just add this line to Program 3:

```
20 POKE 756,204
```

This tells the operating system and ANTIC that the base of the character set is at \$CC00, which is where the international character set resides. Someday you might find some use for these characters. How will you know until you try?

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

Program 1: ATASCII Codes

```
OC 1Ø GRAPHICS Ø
HK 2Ø FOR I=Ø TO 255:PRINT I
      ,
ID 3Ø IF I=155 THEN PRINT "[
      RETURN]":GOTO 5Ø
IC 4Ø PRINT CHR$(27);CHR$(I)
ON 5Ø NEXT I
NH 6Ø REM USE CONTROL-1 TO P
      AUSE
```

Program 2: Keyboard Codes

```
BC 1Ø DIM HEX$(16):HEX$="Ø12
      3456789ABCDEF"
PK 2Ø POKE 764,255
LL 3Ø KEYCODE=PEEK(764)
BC 4Ø IF KEYCODE=255 THEN 3Ø
MF 5Ø HI=INT(KEYCODE/16):LOW
      =KEYCODE-16*HI
LJ 6Ø PRINT "KEYCODE: HEX $";
ON 7Ø PRINT HEX$(HI+1,HI+1);
      HEX$(LOW+1,LOW+1);
JD 8Ø PRINT ", DECIMAL ";KEY
      CODE
AE 9Ø GOTO 2Ø
```

Program 3: Screen Codes

```
OC 1Ø GRAPHICS Ø
BP 3Ø SCREEN=PEEK(88)+256*PE
      EK(89)
BB 4Ø REM FIRST:  SCREEN COD
      E ORDER
EB 5Ø FOR C=Ø TO 255:POKE SC
      REEN+C,C
OI 6Ø NEXT C
CO 7Ø REM THEN:{3 SPACES}ATA
      SCII ORDER
HA 8Ø SCR2=SCREEN+4Ø*8
BH 9Ø FOR C=Ø TO 255:CHAR=C
MP 1ØØ IF C>127 THEN CHAR=C-
       127
EK 11Ø IF CHAR<32 THEN CHAR=
       C+64:GOTO 14Ø
MB 12Ø IF CHAR>95 THEN CHAR=
       C:GOTO 14Ø
ME 13Ø CHAR=C-32
JB 14Ø POKE SCR2+C,CHAR
BI 15Ø NEXT C
BH 999 GOTO 999:REM WAIT FOR
        BREAK KEY
```

©

The Beginners Page

Tom R. Halfhill, Editor

Cutting Strings Without Scissors

Now that we've covered the fundamentals of creating string variables over the past few columns, we can start exploring some of the more powerful string manipulations available in BASIC. Practically all BASIC languages have commands and functions for slicing strings of characters into smaller pieces, pasting two or more strings together to make longer strings, extracting certain sections from within strings, and inserting or replacing portions of strings. Some BASICs even have commands for rapidly searching through strings to find certain sequences of characters.

Since it may not be apparent why you'd want to do any of these things in your own programs, we'll show some common examples for each technique as we go along. In general, these functions give your programs the power to manipulate strings of characters for sorting, screen formatting, printing, storing and retrieving information, and other text-oriented operations.

Slicing Up Strings

Microsoft-style BASICs—such as those included with Commodore, Apple, IBM, Atari ST, and Amiga computers—generally have three functions for extracting shorter strings from longer strings: LEFT$, RIGHT$, and MID$ (pronounced "left-string," "right-string," and "mid-string"). TI BASIC has only one string function, SEG$, which is very similar to MID$. Atari BASIC, found on the 400/800, XL, and XE computers, handles string manipulations quite differently, as we'll see next month.

LEFT$ and RIGHT$ are easy to visualize: They extract characters from the leftmost and rightmost sections of a character string, respectively. You simply follow the keyword with the string variable you're extracting from and the number of characters you want to extract. For example:

```
10 A$="GEORGE WASHINGTON
   CARVER"
20 PRINT A$
30 B$=LEFT$(A$,6)
40 PRINT B$
50 B$=RIGHT$(A$,6)
60 PRINT B$
70 PRINT A$
```

When you type RUN, you should see this on the screen:

```
GEORGE WASHINGTON CARVER
GEORGE
CARVER
GEORGE WASHINGTON CARVER
```

To see how LEFT$ works, look at the statement B$=LEFT$(A$,6) in line 30. It grabs the leftmost six characters of A$—GEORGE—and stores them in B$. Line 40 confirms this by printing B$. To extract the phrase GEORGE WASHINGTON from A$, we could change line 30 to read B$=LEFT$(A$,17)—keeping in mind that spaces count as characters, just like letters, numbers, and symbols. (Of course, you can use your own variable names for A$ and B$ as long as you stick to this basic format.)

RIGHT$ is the opposite of LEFT$: It extracts the rightmost number of characters in A$ that you specify in the RIGHT$ statement. If you change line 50 to read B$= RIGHT$(A$,17), the result is WASHINGTON CARVER.

Line 70 shows that A$ remains intact after sections of it have been extracted with the LEFT$ and RIGHT$ functions. LEFT$ and RIGHT$ actually *copy* sections of the string into B$, rather than cutting the sections out.

Putting Lefty To Work

If you specify a value in a LEFT$ or RIGHT$ statement that is greater than the length of the string—in this case, say, B$=LEFT$(A$,35)—most BASICs return all of A$ in B$, the equivalent of B$=A$. This can happen in a program when you're unsure about the current length of A$, or if you're using a variable for the number parameter in a LEFT$ or RIGHT$ statement and the variable somehow is increased beyond the length of A$. If you specify a zero for this number—as in B$= RIGHT$(A$,0)—most BASICs return a null (empty) string.

If the number you specify in the LEFT$ or RIGHT$ statement is greater than 255, you'll probably get an error. Most Microsoft BASICs don't allow strings longer than 255 characters, so any reference to numbers greater than 255 in string-manipulation statements is invalid. Exceptions are the latest and most advanced Microsoft BASICs, such as Macintosh Microsoft BASIC and Amiga BASIC. They allow strings up to 32,767 characters long.

Of the two functions, LEFT$ is probably used more often than RIGHT$. One practical application of LEFT$ is to truncate user input to a predetermined length. For instance, let's say you're writing a program that asks for the user's name. At some point your program prints the name on the screen, but you want to limit the name to ten characters to keep from messing up your screen formatting. The solution is a line such as INPUT MYNAME$:MYNAME$=LEFT$(MYNAME$,10). Note that in this case, the original content of MYNAME$ *is* lost, because the LEFT$ function stores the leftmost ten characters back into MYNAME$.

Here's another application for LEFT$: Suppose your program asks the user a yes or no question. You can evaluate the answer with a line such as INPUT ANSWER$:IF LEFT$ (ANSWER$,1)="Y" THEN GOTO 1000 (assuming that line 1000 is the beginning of your "Yes" routine). That way, your program responds correctly whether the user types Y, YES, YEAH, YEP, YES SIR, or even YOU BET. ©

Computers and Society

David D. Thornburg, Associate Editor

Humanizing The User Interface, Part 1

Computers should be easy to use. Somehow this seems an obvious requirement for a product, yet many computer users are frustrated at the cumbersome nature of the programs they use day in and day out.

In previous columns, I've argued the case that computers should be transparent to their users—that the computer should disappear into the background, freeing the user to interact directly with the application. A key to transparent computing is the user interface—the vehicle through which the user interacts with the computer. The user interface has three components—input, output, and content.

Input generally involves the communication of physical motion from the user to the computer, signaling the computer to perform various activities. Typing on a keyboard, speaking into a microphone, or drawing a line with a finger on a touch tablet are all ways of using physical movement to convey information to a computer.

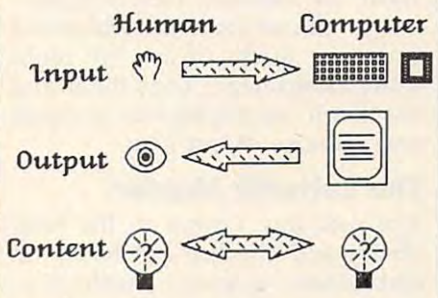
Output consists of messages communicated from the computer to the user's senses. The most often-used sense is vision—usually the screen display.

Content is the purpose of the computer activity—the management of text, the computation of spreadsheets, or the creation of graphic images, to name just a few. Although input flows from the user to the computer, and output goes from the computer to the user, the communication of content is purely inferential. In other words, the user has an internal model of what the computer program is doing, or how it is doing its task. To use a program successfully, it's not important if the user's model of what is happening is accurate. All that's important is if the model is consistent with the program's behavior.

Joy Or Pain

When we're working with a program that has a well-balanced user interface, computing is a joy. When the user interface is bad, we may think that computing just isn't worth the effort.

Fortunately there are a few good programs available that show how easy computers can be to use. Most users of *The Print Shop* (from Brøderbund) would agree that this product is wonderfully easy to use. Many people probably haven't read the instruction manual. This product also has good input and output interfaces that step the user through the creation of customized greeting cards, posters, banners, calendars, etc. This product is one of the top sellers of all time, so the role of a good user interface cannot be underestimated.



Quite often, software designers try to make their products easy to use by designing them to work with a modern input device like a mouse or touch tablet. Unfortunately, this isn't enough. For a computer application to appear transparent, the input, output, and content of the system must be meshed to create a combined ambience that is both natural to the user and appropriate to the task at hand. For example, any attempt to design input devices independently of the applications that use them is risky at best. A program that lets numbers be entered with a joystick may be appropriate for a game in which the joystick is used to select the number of players, but it is clearly the wrong approach for a financial analysis package that requires almost constant entry and update of numbers.

One reason I invented the KoalaPad was to make computers easier to use. Yet input devices like the KoalaPad are not enough by themselves. They can play an important role only when their use is a complementary part of the design of the whole product. This is why some people are frustrated by the Macintosh—not all Mac software is easy to use. It's true that this computer (and the Amiga) is capable of supporting tremendously powerful and easy-to-use software; but it's also true that many programs fall short in this important area.

It's hard to design a good user interface. Millions of dollars went into the research at Xerox that led to the desktop metaphor—the use of windows and pop-up menus that are now becoming commonplace. It took a heavy investment to bring the KoalaPad and Muppet Learning Keys to market. The cost of developing a good program for a personal computer can easily exceed $100,000. (Remember this the next time you think software costs too much!)

As difficult as this task may seem, those of us involved with computer software development owe it to our customers to make ease-of-use our top priority. The market slump of 1984 and 1985 showed that the public is unwilling to blindly accept everything thrown its way.

Next month we'll explore one model of human behavior that provides valuable clues in the search for the best user interfaces. ©

The World Inside the Computer

Fred D'Ignazio, Associate Editor

Snowflakes, Quilts, And Stained Glass Windows

Recently I reviewed the new Amiga from Commodore on public TV's *Educational Computing.* Afterward, I hoped to have a few days to play with the machine before returning it. But I hadn't reckoned with my kids.

They were hooked on the Amiga's mouse, windows, and brilliant colors the first time I turned on the computer. They played with it constantly. The only time I got on the machine was after their bedtime.

My children's favorite program was Electronic Arts' *Deluxe Paint.* It is the most spectacular microcomputer paint program I have ever seen. With its animated, cycling colors and its dozens of drawing and painting tools, it even surpasses the *MacPaint* program on the Macintosh. It is so seductive and so much fun to use that it qualifies as "computer popcorn" (see my column on "Computer Popcorn," COMPUTE!, January 1984). Once you start using it, it's almost impossible to stop.

Like my children, I quickly fell in love with the program. But I still had a nagging doubt. Computer popcorn is scrumptious. But is it also nutritious? How could my children and I use the program to feed our minds and imaginations? Could the program teach us to be artists?

Just A Doodler

So many computer art programs are of the easy-draw variety, like easy-cook microwave ovens and easy-play organs. They get you started drawing, playing, and having fun in no time at all. Then, *bonk!*, you bump into the limitations of your own skills, abilities, and imagination. You've become a super doodler, but you aren't any closer to making professional drawings, pictures, or art. That's because creativity programs, in general, are tools, not teachers. They are enormously enticing tools, but they can never replace a certain amount of training or skill.

Like many people whose artistic aspirations far exceed their abilities, I found this situation extremely frustrating. And I wondered how my children could acquire the skills to use this program properly. I couldn't teach them the skills, and neither could the program.

Then, suddenly, a solution appeared. One night my six-year-old son Eric was scribbling away on the Amiga with *Deluxe Paint.* "Do you like my picture?" he said, turning toward us. My wife and I looked up. We were astounded. From across the room, Eric's glowing picture resembled a stained-glass window. It could have adorned a medieval cathedral. It was beautiful!

Later, as I was falling asleep, I realized Eric had helped me stumble onto a way out of my dilemma. What we needed were images—images drawn from the real world and from works of art. We could study these images, copy them, and use them as inspiration to build new pictures of our own.

The Butterfly Maiden

The next day I went to the local library and checked out books on embroidery, quilting, needlepoint, and nature. The books were filled with images—colorful pictures of the diverse designs and patterns that man and nature can devise. These were to be our teachers.

When I showed these images to my children, I concentrated on patterns and shapes that were symmetrical and geometric. Eric and my daughter Catie could draw these images effortlessly with the tools in *Deluxe Paint.* Catie especially liked the totem-pole faces on blankets woven by the Chilkat Indians of the Pacific Northwest; the brilliant colors and intricate geometric patterns found in nineteenth-century American pieced quilts; and pictures of the Butterfly Maiden, a Hopi kachina doll from northeastern Arizona.

I liked a tapestry, Nightsun, by the German artist Dirk Holger. Eric liked the Resurrection angels, saints, and serpents he found on stained-glass windows from South Africa, the French Loire, and Dublin, Ireland.

As we tried to copy these pictures, and those of Persian lions, helix-shelled snails, and the swirling atmosphere of Jupiter, we found that some images were easier to work with than others. Anything made with needlework, stitching, or embroidery was especially nice because the graph-paper patterns resembled pixels on the computer screen. Pure colors were easier than complex shadings and color blends. The blocky nature of many images was easy to reproduce on the computer, and big patterns made by endlessly repeating little patterns were easy to build using copy and paintbrush commands.

The next day, we went outdoors to look for images on our own. Our field trip turned up all sorts of new shapes: water spurting from the garden hose, wedding cakes at a local bakery, pine cones, and wildflowers. We carried many of these objects to the computer and tried copying their basic patterns. And at night we went back outdoors and looked up at the stars. When we grew cold, we came inside and drew dot-to-dot constellations.

We had found a solution to our problem. We had taken a first step toward becoming computer artists. And we did it by feeding our imagination fresh images, and by studying and copying these images to uncover their underlying patterns and designs. ©

Telecomputing Today

Arlan R. Levitan

Games Modem People Play

When most people think about telecomputing, the first things that probably come to mind will be downloading public domain programs from electronic bulletin boards, retrieving stock quotes and financial information from commercial information services, or communicating with other hobbyists via Special Interest Groups (SIGs). Modems are often viewed as strictly utilitarian pieces of computer gear.

But there is a lighter side to telecomputing—multiple-player telegaming.

The first multiplayer telecomputing game I can recall involved a group of five or six people who were logged onto an online conferencing service playing *Dungeons and Dragons*. Players in California, Illinois, and New York were exploring the stygian depths of underground catacombs created by a Dungeon Master running the whole show from the keyboard of his Apple II in Austin, Texas.

The CompuServe Information Service was one of the pioneers in developing multiplayer online games. CompuServe currently offers a half-dozen or so such diversions to its subscribers. The blast-and-burn crowd can choose among multiple flavors of interstellar conflict: *SpaceWars*, *MegaWars I*, and *MegaWars III*. These games vary in both depth of play and the number of players who may simultaneously participate. *MegaWars III* is the clear heavyweight of the group. It has multiple game phases, including violent battle and economic warfare, and up to a hundred players can be pounding away at their keyboards at once.

Those with more pedestrian tastes may opt for a game of multiplayer blackjack, trading quips with the dealer and other players as electronic gambling chips trade hands.

Wheel Of Misfortune

Not all attempts at multiuser games are smash hits. CompuServe's latest creation is *You Guessed It!*, a TV-style quiz game in which players form teams and take turns attempting to answer questions while ignoring incredibly bad jokes delivered by an eerily obnoxious electronic master of ceremonies. The winners garner points that may be used to purchase gifts offered by sponsors, whose commercials regularly interrupt the game.

I tried *You Guessed It!* for about two hours, racking up what I thought was a respectable number of points. Then I eagerly issued the command that would transfer me to the "prize room" where players can trade points for their heart's desire. But the only prize I qualified for was a bumper sticker that advertised one of the *You Guessed It!* sponsors. To be fair, it did appear that if I played for another hour or two I could lay my hands on a baseball cap which sported (you guessed it) another advertiser's logo.

One of the more interesting experiments in telegaming that I've seen is a moderately obscure program called *COMM-BAT*, marketed by Adventure International. Some friends and I purchased copies of *COMM-BAT* for our Atari 800s back in early 1981 when 300 bits-per-second (bps) modems were still hot stuff for home use.

COMM-BAT lets two computers hook up over phone lines and presents each player with a battlefield map. The adversaries send tanks armed with rockets and lasers scurrying about in search of the enemy's base. When a player's base is destroyed, the game ends. The programs on both ends of the telecomputing link communicate with each other, updating the current battle information displayed on the screens. Players can also send insults and ultimatums to each other during the game.

A Reunion Battle

COMM-BAT does have its limitations. The character graphics are crude, but intentionally so. Versions of *COMM-BAT* were written for TRS-80, Apple, and Atari computers, and owners of these different systems could play *COMM-BAT* with each other and see identical displays on their screens. The biggest drawback was that the game progressed rather slowly due to the 300 bps modems.

Just for grins I pulled out my old copy of *COMM-BAT* and called one of my ex-buddies, now a resident of Denver, Colorado and a fellow user of GTE's PC Pursuit service (see "Telecomputing Today," December 1985). We cranked up our Ataris (now equipped with 1200 bps modems), linked up via PC Pursuit, and had a jolly old transcontinental time blasting the daylights out of each other. The extra speed of the 1200 bps modems and a noise-free connection transformed a mildly interesting game into good, clean Ramboesque fun. Out of curiosity, I called Adventure International and found that *COMM-BAT* is still available. The $49.95 price gets you all three versions of the program.

I'd like to hear about any other commercial or public domain telecomputing games that you may have encountered. I seem to recall some implementations of chess and *Battleship* having been done in the past. I'll compile a list and publish the results in a future issue.

Contact Levitan on The Source (TCT987), CompuServe (70675,463), or Delphi (ARLANL). ©

Donald B. Trivette

The Ultimate Entertainment Center

Picture yourself in front of a 26-inch color monitor—shoes off, feet up, remote control in hand. But this is not just any remote control. This is a special remote unit that controls all of the components in your entertainment/computing system.

You push the TV button and bring up *World News Tonight* on the monitor: Peter Jennings reports that the stock market has soared to new highs. As he fades into a commercial, you decide to call Dow Jones News/Retrieval to see how your own stocks did. But first, you push the compact disc button to fill the room with a Beethoven symphony so real that you wonder where the orchestra is hiding. Then you press the VID2 key to put the computer video on the screen. You reach for the PCjr's wireless keyboard and start the appropriate communications program; then you press TV to return to the news while the computer retrieves the quotes.

At the next break, you display the Dow Jones results onscreen with the VID2 key. After the newscast, you press the VCR STOP, REWIND, and PLAY keys to view the "M*A*S*H" rerun you've been taping from an independent station. But first, you check the progress of the cassette tape you've been recording from an FM stereo broadcast.

This isn't a pipe dream—this is RCA's Dimensia. Billed as intelligent audio/video, it integrates numerous components into a single system commanded from a single remote control. The heart of the system is a 26-inch stereo monitor/receiver. Once you acquire the monitor, you can add other components according to your needs and budget. Current Dimensia components are an AM/FM receiver/amplifier, a compact audio disc player, a cassette tape recorder, two phonographs, a graphic equalizer, and several models of stereo VHS video recorders.

Connection Options

RCA designed the Dimensia system so you can also connect non-Dimensia components, including home computers. The PCjr, with its wireless keyboard, is a particularly good choice; it can be connected in three ways. Like most home computers and videogame machines, the PCjr can be hooked up to a TV's antenna terminals with an RF modulator. Since the Dimensia system allows multiple antennas—selected by remote control—you can switch between the PCjr's screen, cable service, and a satellite dish.

The PCjr also has a composite video output that can be connected to one of the monitor's three video input jacks. The PREVIOUS CHANNEL key lets you instantly switch between a TV program and the computer screen, so you can watch *Dynasty* and play *King's Quest* at the same time.

A third connection option is the Dimensia's RGB direct-drive video input. Although the Dimensia's RGB connectors aren't compatible with the PCjr's RGB plug, the signals *are* compatible. Radio Shack sells a four-conductor, color-coded patch cable that can be modified by anyone handy with a soldering iron to make the connection.

For everything but text, the Dimensia's composite video is as clear as the RGB mode, and it has an added advantage: You can record its output with a video cassette recorder. This means you can run programs on the PCjr and record the results on the VCR, which is perfect for putting titles on your home videos. You can also dub stereo audio from a compact disc player, the AM/FM tuner, the cassette recorder, or the phonograph.

A Piqued PCjr

Since both the Dimensia and the PCjr keyboard use an infrared remote control, there is the possibility of conflict. I couldn't find any button on the Dimensia's 52-key remote controller that the PCjr would recognize, but the computer was well aware that strange infrared signals were reaching its sensor. It squealed like a perturbed pig every time I used the Dimensia remote. This is easily and permanently solved by amputating Junior's little beeper—something I had intended to do for months anyway.

There's another annoying aspect of the PCjr you may want to fix, even if you don't have the Dimensia monitor. The joystick is not a wireless device and the cable that connects it to the computer is too short to reach across the room. Once again, it's Radio Shack to the rescue with its ten-foot joystick extension cord. Of course, this cord was designed for Tandy computers and the connections are not compatible with the PCjr's unusual plugs, so it's back to the soldering iron. Simply chop the joystick cable about eight inches from where it connects to the computer and solder a sub-D nine-pin connector (also available at Radio Shack) on each end, being careful to keep the pin numbers and wire colors consistent. It works perfectly.

The complete Dimensia system with all the components can cost as much as $5,000—but don't hesitate to haggle. The more components you buy, the better deal you can get.

Besides its flexibility, the Dimensia also may be the world's most user-friendly entertainment center. Although not documented in the manuals and unknown to sales people, the monitor displays a help screen across the bottom of the picture when you press AUX 0 0. Drop by a dealer and try it. ©

Programming the TI

C. Regena

IF-THEN Statements

IF-THEN statements are *conditional transfer* commands that make it seem as if computers can think. IF a specified condition is true, THEN the program skips to a certain line number elsewhere in the program; otherwise, the program simply continues to the next line as usual. TI BASIC also allows an ELSE statement as part of IF-THEN. It takes this form:

IF *condition* THEN *line1* ELSE *line2*

IF the condition is true, THEN the computer goes to *line1*, or ELSE the computer goes to *line2*. If the optional ELSE is omitted, control merely passes to the following line. Here's a common example:

```
200 IF SCORE=10 THEN 900
210 PRINT SCORE
```

This statement says that if the value of the variable SCORE is equal to 10, then the program should continue at line 900. Otherwise, the program continues to the next line and prints the score.

You can use the other relational operators to define conditions in IF-THEN statements, too:

```
300 IF A<B THEN 700
400 IF X>Y THEN 200 ELSE 580
500 IF J<>8 THEN 800
```

In each case, the computer evaluates the condition—the expression between the words IF and THEN. If the expression is true, it has the value of −1. If the expression is false, it has the value of zero. Therefore, a statement such as this is valid:

```
320 IF A THEN 400
```

This doesn't look like the more common relational examples, but it implies that if A is equal to −1, then the program goes to line 400.

The condition may look more complex. If you keep in mind that true is −1 and false is zero, you can usually follow the logic. An example is:

```
150 IF (A=B)+C THEN 200
```

The part within the parentheses (A=B) is evaluated first. If A equals B, then the expression is −1 (true); if A does not equal B, the expression is zero (false). This value is then added to the value for C. If the result is −1, the condition is true and control passes to line 200.

Simulating AND/OR

Most other versions of BASIC allow the use of AND and OR in IF-THEN expressions. TI BASIC does not, but we can translate. Again, keep in mind that −1 indicates true.

Suppose we want to test the conditions A=B and C=D. If both are true (IF A=B AND C=D), then we want the program to continue at line 700. Here's one way to do this:

```
IF (A=B)+(C=D)=-2 THEN 700
```

If both conditions are true, each will yield −1 values, so the total will be −2.

Here's an equivalent way to make this test:

```
IF -(A=B)*(C=D) THEN 700
```

Notice that −1 times −1 is +1, so the negative sign in front converts the whole expression to −1 for true.

The word OR is used when one condition OR the other condition is true, but not both:

```
IF (X<Y) OR (X>Z) THEN 300
```

This can be translated to TI BASIC like this:

```
IF (X<Y)+(X>Z) THEN 300
```

Program control transfers to line 300 only if the expression evaluates to −1. This happens if only one of the conditions in parentheses is true (and thus −1) and the other is false (equal to zero).

Even more complex IF-THEN statements are possible by considering different combinations of + and * in evaluating conditions. Suppose after a CALL KEY statement the user may press either ENTER or any of the number keys. Here's the easiest way to set up the logic:

```
200 CALL KEY(0,K,S)
210 IF K=13 THEN 500
220 IF K<48 THEN 200
230 IF K>57 THEN 200
```

Or you can combine the IF statements like this:

```
210 IF (K<>13)+(K<48)+(K>57) THEN
  200
```

Algebra Drill

The sample program this month is a simple drill for beginning algebra students who are learning to add signed numbers. This program illustrates the use of several kinds of IF-THEN statements.

Lines 200 and 230 show two ways to check the length of the numbers to see if a randomly chosen number is negative. If necessary, a plus sign is added to the number.

Lines 280 and 300 determine the answer depending on the value of SUM.

If the answer is zero, line 360 skips the procedure for choosing the plus or minus sign in the answer. If the student needs to choose the sign, line 420 makes sure he or she presses either the plus sign or the minus sign. All other keys are ignored. Line 490 then receives the number keys pressed.

Line 530 checks the student's answer and branches appropriately. Line 590 waits for the student to press the ENTER key before continuing.

If you wish to save typing effort, you can obtain a copy of "Adding Signed Numbers" 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

Adding Signed Numbers

```
100 REM  ADDING SIGNED NU
    MBERS
110 CALL CLEAR
```

```
120 PRINT "ADDING SIGNED
    NUMBERS":::
130 SCORE=0
140 FOR PROB=1 TO 10
150 T$=""
160 RANDOMIZE
170 A=INT(19*RND)-9
180 B=INT(19*RND)-9
190 A$=STR$(A)
200 IF LEN(A$)=2 THEN 220
210 A$="+"&A$
220 B$=STR$(B)
230 IF LEN(B$)>1 THEN 250
240 B$="+"&B$
250 PRINT "ADD"
260 SUM=A+B
270 S$=STR$(SUM)
280 IF SUM<>0 THEN 300
290 S$=" "&S$
300 IF SUM<=0 THEN 320
310 S$="+"&S$
320 TA=8-LEN(S$)
330 PRINT :TAB(4);A$
340 PRINT :TAB(4);B$
350 PRINT TAB(3);"---":::
360 IF SUM=0 THEN 450
370 CALL KEY(0,K,S)
380 CALL HCHAR(23,TA,45)
390 CALL HCHAR(23,TA,32)
400 CALL HCHAR(23,TA,43)
410 CALL HCHAR(23,TA,32)
420 IF (K<>43)+(K<>45)=-2
   THEN 370
430 CALL HCHAR(23,TA,K)
440 T$=CHR$(K)
450 FOR J=1 TO LEN(S$)-1
460 CALL KEY(0,K,S)
470 CALL HCHAR(23,TA+J,63
    )
480 CALL HCHAR(23,TA+J,32
    )
490 IF (K<48)+(K>57)THEN
    460
500 CALL HCHAR(23,TA+J,K)
510 T$=T$&CHR$(K)
520 NEXT J
530 IF SUM<>VAL(T$)THEN 5
    60
540 PRINT ::"CORRECT!"
550 SCORE=SCORE+1
560 PRINT :"THE SUM IS ";
    S$
570 PRINT ::"PRESS <ENTER
    >."
580 CALL KEY(0,K,S)
590 IF K<>13 THEN 580
600 CALL CLEAR
610 NEXT PROB
620 PRINT "OUT OF 10 PROB
    LEMS,"
630 PRINT :"YOUR SCORE IS
    ";SCORE::
640 END
```
©

News & Products

Of Nordic Gods On The 64

Eurosoft International, a software publisher that specializes in introducing European software products to North America, has announced the release of *Valhalla*. Winner of the 1984 British Microcomputing Game of the Year Award, *Valhalla* is an animated, interactive game involving Nordic mythology. Thirty-six mythological characters are featured, each with a different personality. The player interacts with each of these in pursuit of the lost treasure of Valhalla. The mythological characters, shown using the "MoviSoft" animation technique, can either help or hinder your quest depending on their disposition and your actions.

Valhalla is available for the Commodore 64 at a list price of $24.95.

Eurosoft International, 114 East Ave., Norwalk, CT 06851

Circle Reader Service Number 200.

IBM PC MIDI Editor

MIDI Ensemble, a new software package from Sight & Sound for owners of musical equipment with a MIDI interface, consists of three main program modules: Recorder, Event Editor, and Phrase Editor. The Recorder module can be used for recording and overdubbing tracks; the Event Editor enables precise editing of pitch, start time, duration, and key-strike velocity; and the Phrase Editor allows copying, moving, deleting, combining and modifying musical phrases of any length. Also included is a text and graphics editor for creating diagrams or comments with a song file.

MIDI Ensemble runs on the IBM PC; list price, $495.

Sight & Sound Software, 3200 S. 166th St., New Berlin, WI 53151

Circle Reader Service Number 201.

Word Processor For Atari ST

Written by the developers of *AtariWriter* and *AtariWriter Plus*, *Regent Word* is a sophisticated, easy-to-use word processing program for the Atari ST. It features 80-column editing, function key-driven commands, local and global searches, multiple type fonts, print preview, and a communications package. It retails for $49.95.

Regent Spell is an expandable spelling checker for *Regent Word*. The program is shipped with 30,000 words; another 30,000 can be added. Misspelled words are highlighted in context. Commands can be issued via the ST's mouse or though single keystrokes. It also retails for $49.95.

Regent Software is also in the process of designing *Regent Base*, a database management program for small business use.

Regent Software, 7131 Owensmouth, #45A, Canoga Park, CA 91303

Circle Reader Service Number 202.

Home Inventory Package For The 64

What's Our Worth?, from ADITA Enterprises, is a program designed to help you do a complete inventory of your personal belongings. Screen instructions and prompts make it very easy to enter items into inventory, read all items, search for specific information, change or delete items, and make a backup data disk.

What's Our Worth? is available by mail order, and retails for $19.95.

ADITA Enterprises, 116 Bermondsey Way N.W., Calgary, Alberta, Canada T3K 1V4.

Circle Reader Service Number 203.

Educational Enchantment

Sunburst has released *The Enchanted Forest*, a mathematics learning program with a fairy tale setting for grades four and up. The game begins when the witch of the forest transforms all of the forest animals into geometric shapes of different sizes and colors and hides them in ponds. Players travel through the forest with 12 friends, using the concepts of conjunction, disjunction, and negation to break the witch's spells.

The Enchanted Forest was written by Dr. Jerzy Cwirko-Godycki, author of more than 40 children's books. It's available for the 64K Apple II+, IIe, and IIc; and the IBM PC and PCjr. The $59 list price includes a backup disk and teacher's guide.

Sunburst Communications, 39 Washington Avenue, Pleasantville, NY 10570.

Circle Reader Service Number 204.

Beach-Head Sequel For Atari

Beach-Head II, Access Software's sequel to the popular *Beach-Head* game, is now available in a version for the Atari 400/800, XL, and XE series with at least 48K of RAM. Like its predecessor, *Beach-Head II* is a World War II era arcade game that is set on the beaches of Europe. The sequel has several new features including voice synthesis, multiple play screens and play levels, sound effects, and animation techniques.

Beach-Head II for Atari lists for $39.95. It has previously been released in versions for the Commodore 64/128 and Apple II series.

Access Software Inc., 2561 South 1560 West, Woods Cross, UT 84087.

Circle Reader Service Number 205.

Munching On The Apple

Munchers and Troggles abound in the world of *Word Munchers*, an educational game for grades one through five from Minnesota Educational Computing Corporation. Players earn points by making their Munchers eat words with a particular vowel sound while avoiding the enemy Troggles. Teachers can determine which vowel sounds are used and can control the level of word difficulty. Approximately 1,700 words of varying difficulty are included.

Word Munchers runs on all Apple II computers with at least 64K RAM; joystick is optional. Suggested retail price, $49.

Minnesota Educational Computing Corporation, 3490 Lexington Avenue North, St. Paul, MI 55112.

Circle Reader Service Number 206.

Commodore Chemistry

Simon & Schuster has released a Commodore 64 version of the *Chem Lab* educational program for ages nine through twelve. The program contains 50 chemistry experiments with three levels of difficulty. All experiments are simulations of real experiments with actual results. Players can work their way up from Lab Assistant to Nobel Prize Winner.

The computerized laboratory comes equipped with on-screen simulations of: two robot arms for handling chemicals and equipment, five different pieces of lab equipment, plus three Bunsen burners and separate dispensers for gases, liquids, and solids. The chemical reactions are animated and change color, glow, melt, boil, and explode. On-screen messages tell the players what has been created.

Chem Lab for the Commodore 64, with its 96-page user's guide, sells for $39.95. Apple II and IBM PC/PCjr versions are also available.

Simon & Schuster Computer Software, 1230 Avenue of the Americas, New York, NY 10020.

Circle Reader Service Number 207.

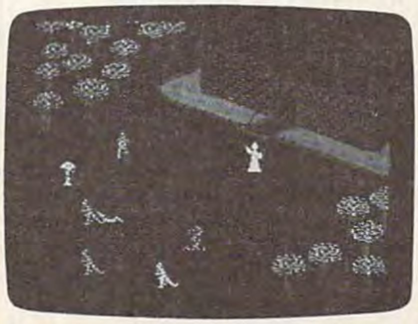
Gandalf The Sorcerer For 64

A spellbound treasure is hidden in a castle surrounded by scaly tailed lizardmen. You, Gandalf the Sorcerer, must protect the treasure by using magic powers from a shining star. Such is the scenario of *Gandalf the Sorcerer*, Tymac's new adventure game for the Commodore 64. The game is for one player and requires a joystick. Three-dimensional graphics are featured.

Suggested retail price is $39.95.

Tymac Controls Corporation, 127 Main Street, Franklin, NJ 07416.

Circle Reader Service Number 208.



The lizardmen ambush the castle in Gandalf the Sorcerer.

Paper Airplane Kit

Simon & Schuster has released *The Great International Paper Airplane Construction Kit*, a set of paper airplane templates based on the bestselling book by the same name. The program contains over a dozen full-page paper airplane designs from biplanes to space shuttles. It also comes with a library of airplane graphics to embellish the airplanes with insignias, logos, windows, engines, pilots, and stewardesses. Also included is a step-by-step manual with instructions, suggestions, and a history of paper aviation.

The Great International Paper Airplane Construction Kit runs on the Apple II series with 64K RAM ($34.95); on the IBM PC, PC-XT, PC AT, and PCjr, with DOS 2.0 or higher and color/-graphics card ($34.95); on the Macintosh with 128K RAM ($39.95); and on the Commodore 64 or 128 ($29.95).

Simon & Schuster, 1230 Avenue of the Americas, New York, NY 10020.

Circle Reader Service Number 209.

New From Mindscape

In *Dick Francis' High Stakes*, a new interactive text adventure from Mindscape, you are a wealthy English horse owner who must foil a sinister plot to cheat you. Based on the book by the popular mystery writer, Dick Francis, the game involves gambling and intrigue.

Also new from Mindscape are *The American Challenge: A Sailing Simulation*, which recreates the America's Cup sailing race, for one or two players; and *James Bond 007 Goldfinger*, an interactive text adventure involving travel, exotic weaponry, and the loves of the legendary 007.

Each game lists for $39.95 and runs on the Apple II and IBM PC computers.

Mindscape Inc., 3444 Dundee Road, Northbrook, IL 60062.

Circle Reader Service Number 210.

Educational Programs For Pre-School, High School

Grover and Ernie from "Sesame Street" enliven two new educational games from CBS Learning Systems. *Grover's Animal Adventures* takes preschoolers into four different animal environments: the African Grasslands, the Atlantic Ocean, a North American forest, and a barnyard. Children select animated animals and objects and place them in the appropriate environment on land, in water, or in the sky. In *Ernie's Big Splash*, children help Ernie find his Rubber Duckie by building a pathway that leads from the Duckie's soap dish into Ernie's bathtub. Both games are for ages four to six; each lists for $14.95.

CBS has also released *Mastering the ACT* (American College Testing Assessment), a self-paced preparation course for high school students that was developed by the National Association of Secondary School Principals. The program features full-length simulated ACT pre- and post-tests which provide self-scoring and detailed error analysis. Development exercises cover English, math, social studies, and natural sciences. For the Commodore 64/128 ($79.95), the Apple II series, and IBM PC and PCjr ($99.95 each).

CBS Learning Systems, One Fawcett Place, Greenwich, CT 06836.

Circle Reader Service Number 211.

©

MLX Machine Language Entry Program For Atari

Charles Brannon, Program Editor

MLX is a labor-saving utility that allows almost fail-safe entry of machine language programs published in COMPUTE!. *You need to know nothing about machine language to use MLX—it was designed for everyone.*

"MLX" is a new way to enter long machine language (ML) programs with a minimum of fuss. MLX lets you enter the numbers from a special list that looks similar to BASIC DATA statements. It checks your typing on a line-by-line basis. It won't let you enter illegal characters when you should be typing numbers. It won't let you enter numbers greater than 255 (forbidden in ML). It won't let you enter the wrong numbers on the wrong line. In addition, MLX creates a ready-to-use tape or disk file.

Using MLX

Type in and save MLX (you'll want to use it in the future). When you're ready to type in an ML program, run MLX. MLX asks you for three numbers: the starting address, the ending address, and the run/init address. These numbers are given in the article accompanying the ML program presented in MLX format. You must also choose one of three options for saving the file: as a boot tape, as disk binary file, or as boot disk. The article with the ML program should specify which formats may be used.

When you run MLX, you'll see a prompt corresponding to the starting address. The prompt is the current line you are entering from the listing. It increases by six each time you enter a line. That's because each line has seven numbers—six actual data numbers plus a checksum number. The checksum verifies that you typed the previous six numbers correctly. If you enter any of the six numbers wrong, or enter the checksum wrong, the computer rings a buzzer and prompts you to reenter the line. If you enter it correctly, a bell tone sounds and you continue to the next line.

MLX accepts only numbers as input. If you make a typing error, press the DEL/BACK SPACE; the entire number is deleted. You can press it as many times as necessary back to the start of the line. If you enter three-digit numbers as listed, the computer automatically prints the comma and goes on to accept the next number. If you enter fewer than three digits, you can press the comma key, the space bar, or the RETURN key to advance to the next number. The checksum automatically appears in inverse video for emphasis.

MLX Commands

When you finish typing an ML listing (assuming you type it all in one session), you can then save the completed program on tape or disk. Follow the screen instructions. If you get any errors while saving, you probably have a bad disk, or the disk is full, or you've made a typo when entering the MLX program itself.

You don't have to enter the whole ML program in one sitting. MLX lets you enter as much as you want, save it, and then reload the file from tape or disk later. MLX recognizes these commands:

CTRL-S	**Save**
CTRL-L	**Load**
CTRL-N	**New Address**
CTRL-D	**Display**

To issue a command, hold down the CTRL key (CONTROL on the XL models) and press the indicated key. When you enter a command, MLX jumps out of the line you've been typing, so we recommend you do it at a new prompt. Use the Save command (CTRL-S) to save what you've been working on. It will save on tape or disk, as if you've finished, but the tape or disk won't work, of course, until you finish the typing. Remember to make a note of what address you stop at. The next time you run MLX, answer all the prompts as you did before—regardless of where you stopped typing—then insert the disk or tape. When you get to the line number prompt, press CTRL-L to reload the partly completed file into memory. Then use the New Address command to resume typing.

To use the New Address command, press CTRL-N and enter the address where you previously stopped. The prompt will change, and you can then continue typing. Always enter a New Address that matches up with one of the line numbers in the MLX-format listing, or else the checksum won't work. The Display command lets you display a section of your typing. After you press CTRL-D, enter two addresses within the line number range of the listing. You can break out of the listing display and return to the prompt by pressing any key.

Atari MLX: Machine Language Entry

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

```
DA 100 GRAPHICS Ø:DL=PEEK(56
       Ø)+256*PEEK(561)+4:PO
       KE DL-1,71:POKE DL+2,
       6
NJ 110 POSITION 8,Ø:? "MLX":
       POSITION 23,Ø:? "fail
       safe entry":POKE 71Ø,
       Ø:?
JK 120 ? "Starting Address";
       :INPUT BEG:? "  Endin
       g Address";:INPUT FIN
       :? "Run/Init Address"
       ;:INPUT STARTADR
DD 130 DIM A(6),BUFFER$(FIN-
       BEG+127),T$(2Ø),F$(2Ø
       ),CIO$(7),SECTOR$(128
       ),DSKINV$(6)
JJ 140 OPEN #1,4,Ø,"K:":? :?
        ,"Tape or Disk:";
BM 150 BUFFER$=CHR$(Ø):BUFFE
       R$(FIN-BEG+3Ø)=BUFFER
       $:BUFFER$(2)=BUFFER$:
       SECTOR$=BUFFER$
GC 160 ADDR=BEG:CIO$="hhh":C
       IO$(4)=CHR$(17Ø):CIO$
       (5)="LV":CIO$(7)=CHR$
       (228)
EJ 170 GET #1,MEDIA:IF MEDIA
       <>84 AND MEDIA<>68 TH
       EN 17Ø
PO 180 ? CHR$(MEDIA):? :IF M
       EDIA<>ASC("T") THEN B
       UFFER$="":GOTO 25Ø
PL 190 BEG=BEG-24:BUFFER$=CH
       R$(Ø):BUFFER$(2)=CHR$
       (INT((FIN-BEG+127)/12
       8))
KF 200 H=INT(BEG/256):L=BEG-
       H*256:BUFFER$(3)=CHR$
       (L):BUFFER$(4)=CHR$(H
       )
EC 210 PINIT=BEG+8:H=INT(PIN
       IT/256):L=PINIT-H*256
       :BUFFER$(5)=CHR$(L):B
       UFFER$(6)=CHR$(H)
PB 220 FOR I=7 TO 24:READ A:
       BUFFER$(I)=CHR$(A):NE
       XT I:DATA 24,96,169,6
       Ø,141,2,211,169,Ø,133
       ,1Ø,169,Ø,133,11,76,Ø
       ,Ø
DP 230 H=INT(STARTADR/256):L
       =STARTADR-H*256:BUFFE
       R$(15)=CHR$(L):BUFFER
       $(19)=CHR$(H)
KL 240 BUFFER$(23)=CHR$(L):B
       UFFER$(24)=CHR$(H)
HI 250 IF MEDIA<>ASC("D") TH
       EN 36Ø
OO 260 ? :? "Boot Disk or Bi
       nary File:";
LI 270 GET #1,DTYPE:IF DTYPE
```

```
    <>68 AND DTYPE<>70 THEN 270
GM 280 ? CHR$(DTYPE):IF DTYPE=70 THEN 360
PJ 290 BEG=BEG-30:BUFFER$=CHR$(0):BUFFER$(2)=CHR$(INT((FIN-BEG+127)/128))
KG 300 H=INT(BEG/256):L=BEG-H*256:BUFFER$(3)=CHR$(L):BUFFER$(4)=CHR$(H)
HH 310 PINIT=STARTADR:H=INT(PINIT/256):L=PINIT-H*256:BUFFER$(5)=CHR$(L):BUFFER$(6)=CHR$(H)
AO 320 RESTORE 330:FOR I=7 TO 30:READ A:BUFFER$(I)=CHR$(A):NEXT I
GA 330 DATA 169,0,141,231,2,133,14,169,0,141,232,2,133,15,169,0,133,10,169,0,133,11,24,96
OB 340 H=INT(BEG/256):L=BEG-H*256:BUFFER$(8)=CHR$(L):BUFFER$(15)=CHR$(H)
DD 350 H=INT(STARTADR/256):L=STARTADR-H*256:BUFFER$(22)=CHR$(L):BUFFER$(26)=CHR$(H)
JP 360 GRAPHICS 0:POKE 712,10:POKE 710,10:POKE 709,2
JK 370 ? ADDR;":";:FOR J=1 TO 6
NF 380 GOSUB 570:IF N=-1 THEN J=J-1:GOTO 380
BF 390 IF N=-19 THEN 720
OI 400 IF N=-12 THEN LET READ=1:GOTO 720
AI 410 TRAP 410:IF N=-14 THEN ? :? "New Address";:INPUT ADDR:? :GOTO 370
JD 420 TRAP 32767:IF N<>-4 THEN 480
AJ 430 TRAP 430:? :? "Display:From";:INPUT F:? ," To";:INPUT T:TRAP 32767
ML 440 IF F<BEG OR F>FIN OR T<BEG OR T>FIN OR T<F THEN ? CHR$(253);"At least ";BEG;", Not More Than ";FIN:GOTO 430
MH 450 FOR I=F TO T STEP 6:? :? I;":";:FOR K=0 TO 5:N=PEEK(ADR(BUFFER$)+I+K-BEG):T$="000":T$(4-LEN(STR$(N)))=STR$(N)
MA 460 IF PEEK(764)<255 THEN GET #1,A:POP :POP :? :GOTO 370
FM 470 ? T$;",";:NEXT K:? CHR$(126);:NEXT I:? :? :GOTO 370
GA 480 IF N<0 THEN ? :GOTO 370
MH 490 A(J)=N:NEXT J
JK 500 CKSUM=ADDR-INT(ADDR/256)*256:FOR I=1 TO 6:CKSUM=CKSUM+A(I):CKSUM=CKSUM-256*(CKSUM>255):NEXT I
KK 510 RF=128:SOUND 0,200,12,8:GOSUB 570:SOUND 0,0,0,0:RF=0:? CHR$(126)
CN 520 IF N<>CKSUM THEN ? :? "Incorrect";CHR$(253);:? :GOTO 370
EK 530 FOR W=15 TO 0 STEP -1:SOUND 0,50,10,W:NEXT W
FL 540 FOR I=1 TO 6:POKE ADR(BUFFER$)+ADDR-BEG+I-1,A(I):NEXT I
HB 550 ADDR=ADDR+6:IF ADDR<=FIN THEN 370
GM 560 GOTO 710
FI 570 N=0:Z=0
PH 580 GET #1,A:IF A=155 OR A=44 OR A=32 THEN 670
FB 590 IF A<32 THEN N=-A:RETURN
EB 600 IF A<>126 THEN 630
ML 610 GOSUB 690:IF I=1 AND T=44 THEN N=-1:? CHR$(126);:GOTO 690
GN 620 GOTO 570
GJ 630 IF A<48 OR A>57 THEN 580
AN 640 ? CHR$(A+RF);:N=N*10+A-48
EB 650 IF N>255 THEN ? CHR$(253);:A=126:GOTO 600
EH 660 Z=Z+1:IF Z<3 THEN 580
JH 670 IF Z=0 THEN ? CHR$(253);:GOTO 570
KC 680 ? ",";:RETURN
ND 690 POKE 752,1:FOR I=1 TO 3:? CHR$(30);:GET #6,T:IF T<>44 AND T<>58 THEN ? CHR$(A);:NEXT I
PI 700 POKE 752,0:? " ";CHR$(126);:RETURN
KM 710 GRAPHICS 0:POKE 710,26:POKE 712,26:POKE 709,2
FF 720 IF MEDIA=ASC("T") THEN 890
OJ 730 REM DISK
OK 740 IF READ THEN ? :? "Load File":?
IG 750 IF DTYPE<>70 THEN 1040
AE 760 ? :? "Enter AUTORUN.SYS for automatic use":? :? "Enter filename":INPUT T$
GF 770 F$=T$:IF LEN(T$)>2 THEN IF T$(1,2)<>"D:" THEN F$="D:":F$(3)=T$
NJ 780 TRAP 870:CLOSE #2:OPEN #2,8-4*READ,0,F$:? :? "Working..."
JM 790 IF READ THEN FOR I=1 TO 6:GET #2,A:NEXT I:GOTO 820
PD 800 PUT #2,255:PUT #2,255
DJ 810 H=INT(BEG/256):L=BEG-H*256:PUT #2,L:PUT #2,H:H=INT(FIN/256):L=FIN-H*256:PUT #2,L:PUT #2,H
NF 820 GOSUB 970:IF PEEK(195)>1 THEN 870
IF 830 IF STARTADR=0 OR READ THEN 850
FD 840 PUT #2,224:PUT #2,2:PUT #2,225:PUT #2,2:H=INT(STARTADR/256):L=STARTADR-H*256:PUT #2,L:PUT #2,H
HH 850 TRAP 32767:CLOSE #2:? "Finished.":IF READ THEN ? :? :LET READ=0:GOTO 360
HF 860 END
FO 870 ? "Error ";PEEK(195);" trying to access":? F$:CLOSE #2:? :GOTO 760
MC 880 REM BOOT TAPE
HN 890 IF READ THEN ? :? "Read Tape"
HI 900 ? :? :? "Insert, Rewind Tape.":? "Press PLAY ";:IF NOT READ THEN ? "& RECORD"
LP 910 ? :? "Press RETURN when ready:";
JH 920 TRAP 960:CLOSE #2:OPEN #2,8-4*READ,128,"C:":? :? "Working..."
NH 930 GOSUB 970:IF PEEK(195)>1 THEN 960
HH 940 CLOSE #2:TRAP 32767:? "Finished.":? :? :IF READ THEN LET READ=0:GOTO 360
HF 950 END
CD 960 ? :? "Error ";PEEK(195);" when reading/writing boot tape":? :CLOSE #2:GOTO 890
MB 970 REM CIO Load/Save File#2 opened READ=0 for write, READ=1 for read
EA 980 X=32:REM File#2,$20
EF 990 ICCOM=834:ICBADR=836:ICBLEN=840:ICSTAT=835
MD 1000 H=INT(ADR(BUFFER$)/256):L=ADR(BUFFER$)-H*256:POKE ICBADR+X,L:POKE ICBADR+X+1,H
FH 1010 L=FIN-BEG+1:H=INT(L/256):L=L-H*256:POKE ICBLEN+X,L:POKE ICBLEN+X+1,H
MD 1020 POKE ICCOM+X,11-4*READ:A=USR(ADR(CIO$),X)
BG 1030 POKE 195,PEEK(ICSTAT):RETURN
KA 1040 REM SECTOR I/O
GC 1050 IF READ THEN 1100
HE 1060 ? :? "Format Disk In Drive 1? (Y/N):";
FC 1070 GET #1,A:IF A<>78 AND A<>89 THEN 1070
EC 1080 ? CHR$(A):IF A=78 THEN 1100
CP 1090 ? :? "Formatting...":XIO 254,#2,0,0,"D:":? "Format Complete":?
AC 1100 NR=INT((FIN-BEG+127)/128):BUFFER$(FIN-BEG+2)=CHR$(0):IF READ THEN ? "Reading...":GOTO 1120
LE 1110 ? "Writing..."
LI 1120 FOR I=1 TO NR:S=I
IO 1130 IF READ THEN GOSUB 1220:BUFFER$(I*128-127)=SECTOR$:GOTO 1160
PL 1140 SECTOR$=BUFFER$(I*128-127)
AM 1150 GOSUB 1220
DN 1160 IF PEEK(DSTATS)<>1 THEN 1200
FB 1170 NEXT I
GM 1180 IF NOT READ THEN END
DH 1190 ? :? :LET READ=0:GOTO 360
JJ 1200 ? "Error on disk access.":? "May need formatting.":GOTO 1040
KI 1210 REM
```

```
BL 1220 REM SECTOR ACCESS SUBROUTINE
IG 1230 REM Drive ONE
IH 1240 REM Pass buffer in SECTOR$
MP 1250 REM sector # in variable S
EG 1260 REM READ=1 for read,
KJ 1270 REM READ=0 for write
BN 1280 BASE=3*256
GL 1290 DUNIT=BASE+1:DCOMND=BASE+2:DSTATS=BASE+3
NL 1300 DBUFLO=BASE+4:DBUFHI=BASE+5
AI 1310 DBYTLO=BASE+8:DBYTHI=BASE+9
JA 1320 DAUX1=BASE+10:DAUX2=BASE+11
PN 1330 REM DIM DSKINV$(4)
CA 1340 DSKINV$="hLS":DSKINV$(4)=CHR$(228)
PF 1350 POKE DUNIT,1:A=ADR(SECTOR$):H=INT(A/256):L=A-256*H
BP 1360 POKE DBUFHI,H
CO 1370 POKE DBUFLO,L
PD 1380 POKE DCOMND,87-5*READ
AA 1390 POKE DAUX2,INT(S/256):POKE DAUX1,S-PEEK(DAUX2)*256
KJ 1400 A=USR(ADR(DSKINV$))
KG 1410 RETURN
```

©

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 implemented a special listing convention as well as a program to check your typing—"Automatic Proofreader.

Commodore, Apple, and Atari programs can 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. For Commodore, Apple, and Atari, a 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 S. 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 S}, 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 Atari logo key.

Any more than two spaces will be listed. For example, {6 SPACES} means press the space bar six times. Our listings never leave a space at the end of a line, instead moving it to the next printed line as {SPACE}.

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:
{CLR}	SHIFT CLR/HOME	
{HOME}	CLR/HOME	
{UP}	SHIFT ↑ CRSR ↓	
{DOWN}	↑ CRSR ↓	
{LEFT}	SHIFT ← CRSR →	
{RIGHT}	← CRSR →	
{RVS}	CTRL 9	
{OFF}	CTRL 0	
{BLK}	CTRL 1	
{WHT}	CTRL 2	
{RED}	CTRL 3	
{CYN}	CTRL 4	
{PUR}	CTRL 5	
{GRN}	CTRL 6	
{BLU}	CTRL 7	
{YEL}	CTRL 8	

When You Read:	Press:	See:
[< 1 >]	COMMODORE 1	
[< 2 >]	COMMODORE 2	
[< 3 >]	COMMODORE 3	
[< 4 >]	COMMODORE 4	
[< 5 >]	COMMODORE 5	
[< 6 >]	COMMODORE 6	
[< 7 >]	COMMODORE 7	
[< 8 >]	COMMODORE 8	
{ F1 }	f1	
{ F2 }	SHIFT f1	
{ F3 }	f3	
{ F4 }	SHIFT f3	
{ F5 }	f5	
{ F6 }	SHIFT f5	
{ F7 }	f7	
{ F8 }	SHIFT f7	
←	←	

The Automatic Proofreader

This month, we are featuring a completely new Proofreader for the Commodore 64, 128, VIC, Plus/4, and 16. Please refer to "The New Automatic Proofreader for Commodore" article elsewhere in this issue for more information. Type in the appropriate program listed below, then save it for future use. On the Atari, run the Proofreader to activate it, then enter NEW to erase the BASIC loader (the Proofreader remains active in memory as a machine language program). Pressing SYSTEM RESET deactivates the Proofreader. Use PRINT USR(1536) to reenable the Atari Proofreader. 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.

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

Compare the value provided by the Proofreader with the checksum printed in the program listing in the magazine. In Commodore listings, the checksum is set off from the rest of the line with *rem*. This prevents a syntax error if the checksum is typed in, but the REM statements and checksums need *not* be typed in.

In Atari, Apple, and IBM listings, 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) and compare the checksums. If they match, go on to the next line. If not, check your typing: You've made a mistake. 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 stil matches the listing. Because of the checksum method used, do not use abbreviations, such as ? for PRINT. The IBM Proofreader is the pickiest of all; it *will* detect errors in spacing and transposition. Be sure to leave Caps Lock on, except when typing lowercase characters.

IBM Proofreader Commands

Since the IBM Proofreader (Program 2) 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 will prompt 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 into the normal BASIC environment (this will replace 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 a 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:RE
    AD A:POKE I,A:CK=CK+A
    :NEXT I
120 IF CK<>19072 THEN ? "
    Error in DATA Stateme
    nts.  Check Typing.":
    END
130 A=USR(1536)
140 ? :? "Automatic Proof
    reader Now Activated.
    "
150 END
160 DATA 104,160,0,185,26
    ,3,201,69,240,7
170 DATA 200,200,192,34,2
    08,243,96,200,169,74
180 DATA 153,26,3,200,169
    ,6,153,26,3,162
190 DATA 0,189,0,228,157,
    74,6,232,224,16
200 DATA 208,245,169,93,1
    41,78,6,169,6,141
210 DATA 79,6,24,173,4,22
    8,105,1,141,95
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,
    124,241,76,205,238
250 DATA 0,0,0,0,0,32,62,
    246,8,201
260 DATA 155,240,13,201,3
    2,240,7,72,24,101
270 DATA 203,133,203,104,
    40,96,72,152,72,138
280 DATA 72,160,0,169,128
    ,145,88,200,192,40
290 DATA 208,249,165,203,
    74,74,74,74,24,105
300 DATA 161,160,3,145,88
    ,165,203,41,15,24
310 DATA 105,161,200,145,
    88,169,0,133,203,104
320 DATA 170,104,168,104,
    40,96
```

Program 2: IBM Proofreader

By Charles Brannon, Program Editor

```
MC 10 'Automatic Proofreader Ver
      sion 3.0 (Lines 205,206 ad
      ded/190 deleted/470,490 ch
      anged from V2.0)
LD 100 DIM L$(500),LNUM(500):COL
       OR 0,7,7:KEY OFF:CLS:MAX=
       0:LNUM(0)=65536!
PK 110 ON ERROR GOTO 120:KEY 15,
       CHR$(4)+CHR$(70):ON KEY(1
       5) GOSUB 640:KEY (15) ON:
       GOTO 130
BE 120 RESUME 130
BJ 130 DEF SEG=&H40:W=PEEK(&H4A)
IH 140 ON ERROR GOTO 650:PRINT:P
       RINT"Proofreader Ready."
KB 150 LINE INPUT L$:Y=CSRLIN-IN
       T(LEN(L$)/W)-1:LOCATE Y,1
CA 160 DEF SEG=0:POKE 1050,30:PO
       KE 1052,34:POKE 1054,0:PO
       KE 1055,79:POKE 1056,13:P
       OKE 1057,28:LINE INPUT L$
       :DEF SEG:IF L$="" THEN 15
       0
BC 170 IF LEFT$(L$,1)=" " THEN L
       $=MID$(L$,2):GOTO 170
NN 180 IF VAL(LEFT$(L$,2))=0 AND
        MID$(L$,3,1)=" " THEN L$
       =MID$(L$,4)
ND 200 IF ASC(L$)>57 THEN 260 'n
       o line number, therefore
       command
JB 205 BL=INSTR(L$," "):IF BL=0
       THEN BL$=L$:GOTO 206 ELSE
        BL$=LEFT$(L$,BL-1)
GH 206 LNUM=VAL(BL$):TEXT$=MID$(
       L$,LEN(STR$(LNUM))+1)
OG 210 IF TEXT$="" THEN GOSUB 54
       0:IF LNUM=LNUM(P) THEN GO
       SUB 560:GOTO 150 ELSE 150
MB 220 CKSUM=0:FOR I=1 TO LEN(L$
       ):CKSUM=(CKSUM+ASC(MID$(L
       $,I))*I) AND 255:NEXT:LOC
       ATE Y,1:PRINT CHR$(65+CKS
       UM/16)+CHR$(65+(CKSUM AND
        15))+" "+L$
JE 230 GOSUB 540:IF LNUM(P)=LNUM
        THEN L$(P)=TEXT$:GOTO 15
       0 'replace line
CL 240 GOSUB 580:GOTO 150 'inser
       t the line
AD 260 TEXT$="":FOR I=1 TO LEN(L
       $):A=ASC(MID$(L$,I)):TEXT
       $=TEXT$+CHR$(A+32*(A>96 A
       ND A<123)):NEXT
LP 270 DELIMITER=INSTR(TEXT$," "
       ):COMMAND$=TEXT$:ARG$="":
       IF DELIMITER THEN COMMAND
       $=LEFT$(TEXT$,DELIMITER-1
       ):ARG$=MID$(TEXT$,DELIMIT
       ER+1) ELSE DELIMITER=INST
       R(TEXT$,CHR$(34)):IF DELI
       MITER THEN COMMAND$=LEFT$
       (TEXT$,DELIMITER-1):ARG$=
       MID$(TEXT$,DELIMITER)
FC 280 IF COMMAND$<>"LIST" THEN
       410
ID 290 OPEN "scrn:" FOR OUTPUT A
       S #1
LH 300 IF ARG$="" THEN FIRST=0:P
       =MAX-1:GOTO 340
```

```
IJ 310 DELIMITER=INSTR(ARG$,"-")
       :IF DELIMITER=0 THEN LNUM
       =VAL(ARG$):GOSUB 540:FIRS
       T=P:GOTO 340
BP 320 FIRST=VAL(LEFT$(ARG$,DELI
       MITER)):LAST=VAL(MID$(ARG
       $,DELIMITER+1))
EC 330 LNUM=FIRST:GOSUB 540:FIRS
       T=P:LNUM=LAST:GOSUB 540:I
       F P=0 THEN P=MAX-1
GD 340 FOR X=FIRST TO P:N$=MID$(
       STR$(LNUM(X)),2)+" "
KA 350 IF CKFLAG=0 THEN A$="":GO
       TO 370
PF 360 CKSUM=0:A$=N$+L$(X):FOR I
       =1 TO LEN(A$):CKSUM=(CKSU
       M+ASC(MID$(A$,I))*I) AND
       255:NEXT:A$=CHR$(65+CKSUM
       /16)+CHR$(65+(CKSUM AND 1
       5))+" "
DO 370 PRINT #1,A$+N$+L$(X)
JJ 380 IF INKEY$<>"" THEN X=P
OF 390 NEXT :CLOSE #1:CKFLAG=0
CA 400 GOTO 130
PD 410 IF COMMAND$="LLIST" THEN
       OPEN "lpt1:" FOR OUTPUT A
       S #1:GOTO 300
GM 420 IF COMMAND$="CHECK" THEN
       CKFLAG=1:GOTO 290
KA 430 IF COMMAND$<>"SAVE" THEN
       450
CL 440 GOSUB 600:OPEN ARG$ FOR O
       UTPUT AS #1:ARG$="":GOTO
       300
OE 450 IF COMMAND$<>"LOAD" THEN
       490
PG 460 GOSUB 600:OPEN ARG$ FOR I
       NPUT AS #1:MAX=0:P=0
KA 470 WHILE NOT EOF(1):LINE INP
       UT #1,L$:BL=INSTR(L$," ")
       :BL$=LEFT$(L$,BL-1):LNUM(
       P)=VAL(BL$):L$(P)=MID$(L$
       ,LEN(STR$(VAL(BL$)))+1):P
       =P+1:WEND
KK 480 MAX=P:CLOSE #1:GOTO 130
GJ 490 IF COMMAND$="NEW" THEN IN
       PUT "Erase program - Are
       you sure";L$:IF LEFT$(L$,
       1)="y" OR LEFT$(L$,1)="Y"
        THEN MAX=0:LNUM(0)=65536
       !:GOTO 130:ELSE 130
CL 500 IF COMMAND$="BASIC" THEN
       COLOR 7,0,0:ON ERROR GOTO
        0:CLS:END
NC 510 IF COMMAND$<>"FILES" THEN
        520
IH 515 IF ARG$="" THEN ARG$="A:"
        ELSE SEL=1:GOSUB 600
IO 517 FILES ARG$:GOTO 130
DD 520 PRINT"Syntax error":GOTO
       130
BO 540 P=0:WHILE LNUM>LNUM(P) AN
       D P<MAX:P=P+1:WEND:RETURN
KM 560 MAX=MAX-1:FOR X=P TO MAX:
       LNUM(X)=LNUM(X+1):L$(X)=L
       $(X+1):NEXT:RETURN
GK 580 MAX=MAX+1:FOR X=MAX TO P+
       1 STEP -1:LNUM(X)=LNUM(X-
       1):L$(X)=L$(X-1):NEXT:L$(
       P)=TEXT$:LNUM(P)=LNUM:RET
       URN
GA 600 IF LEFT$(ARG$,1)<>CHR$(34
       ) THEN 520 ELSE ARG$=MID$
       (ARG$,2)
EE 610 IF RIGHT$(ARG$,1)=CHR$(34
       ) THEN ARG$=LEFT$(ARG$,LE
       N(ARG$)-1)
LA 620 IF SEL=0 AND INSTR(ARG$,"
       .")=0 THEN ARG$=ARG$+".BA
       S"
DD 630 SEL=0:RETURN
HM 640 CLOSE #1:CKFLAG=0:PRINT"S
       topped.":RETURN 150
II 650 PRINT "Error #";ERR:RESUM
       E 150
```

Program 3: Apple Proofreader

By Tim Victor, Editorial Programmer

```
10 C = 0: 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 56,0: POKE 57,3: CA
   LL 1002: GOTO 50
40 PRINT CHR$ (4);"IN#A$300"
50 POKE 34,0: HOME : POKE 34,1:
    VTAB 2: PRINT "PROOFREADER
   INSTALLED"
60 NEW
100 DATA 216,32,27,253,201,141
110 DATA 208,60,138,72,169,0
120 DATA 72,189,255,1,201,160
130 DATA 240,8,104,10,125,255
140 DATA 1,105,0,72,202,208
150 DATA 238,104,170,41,15,9
160 DATA 48,201,58,144,2,233
170 DATA 57,141,1,4,138,74
180 DATA 74,74,74,41,15,9
190 DATA 48,201,58,144,2,233
200 DATA 57,141,0,4,104,170
210 DATA 169,141,96
```

©

CAPUTE!

SpeedScript Update

There is an error in the correction to Apple *SpeedScript* from the "*SpeedScript 3.0* Revisited" article in the December 1985 issue (p. 90) which causes the page number to repeat continuously when the # formatting command is used. In line 1C88 of the listing, the 9D should be a 9C. Load *SpeedScript* back into Apple "MLX" and enter the following replacement line:

1C88: AC E5 1E D0 9C AE E6 1E EC

After making the correction, be sure to use the MLX Save option to save a new copy of *SpeedScript*.

The item in the January 1986 "Reader's Feedback" column (p. 10) that told how to make Commodore 64 *SpeedScript 3.0* default to disk for saving and loading had transposed digits in the middle POKE address. The line should have read:

POKE 4904,234: POKE 4905,169: POKE 4906,68

This modification works for all updates of version 3 (3.0, 3.1, or 3.2).

Atari Solitaire

The Atari listing for this game from the January 1986 issue (Program 2, p. 48) has a typographical error in line 910. The third character in S$, which defines the card suits, should be {.} instead of the apostrophe shown. CTRL-period is the diamond graphic character.

Formatted Printouts For Commodore

There are two errors in the DATA statements for this program from the January 1986 issue (p. 99). In line 540, the null string, "", should come before the item "BLACK" rather than after it. In line 640, the last item in the line should be " ↑ " rather than a null string.

Skyscape For IBM & Apple

Certain combinations of date and time inputs cause syntax errors in the IBM and Apple versions of this astronomy program from the November 1985 issue (p. 62). To correct this, change CC <= 0 to CC <= 1 in line 2060 of the IBM version (Program 3) and line 1770 of the Apple version (Program 4).

Memo Diary

The Commodore version of this calendar utility from the December 1985 issue (p. 65) won't work with tape. Tape users should modify the OPEN statement in line 3170 as follows:

OPEN 1,1+7*D1,8*D1+1,F$+G$:

Author Jim Butterfield also recommends that line 660 be replaced with 660 REM. With this change the calendar file will always be updated. ©

Classified

SOFTWARE

TI-99/4A NEW STATES AND CAPITALS GAME
Hi-Res map of USA. Send $12 for cass.
Or $1 for more info. to: TRINITY SYSTEMS
1022 Grandview, Pittsburgh, PA 15237

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

PROGRAMS FOR THE TANDY 1000
Send $1 for list of educat. & entertainment
programs. Refundable with first purchase.
SODA POP SW, POB 653, Kenosha, WI 53141

FREE APPLE SOFTWARE
Over 1,000 Public Domain Programs on 50
diskettes, $5 each, plus $1 shipping per order.
Send $1 for catalog, refundable.
C & H ENTERPRISES
Box 29243, Memphis, TN 38127

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

SAVE MONEY! EASY TAX SIMPLIFIES THE 15
most common IRS tax forms. Faster, line
by line preparation on screen and printer.
Commodore 64, disk. Send $39.95 plus $2 s/h to
Hybrid Software, 1739 Schilder Lane,
Waverly, OH 45690

PROJECT PLANNING/MANAGEMENT using
the C64, SX, or C128. Data sheet for SASE -
Prgm for $106.95 (CA res. add 6% sls tx).
LAWCO, Dept. C, Box 2009, Manteca, CA 95336

Genealogy Program for the C64. **"FAMILY TREE"** will produce Pedigree Charts, Family
Group Records, Individual Files, Indexes, Searches
of Ancestors. LDS version available. "The Best"
genealogy program for the 64. $49.95,
GENEALOGY SOFTWARE, POB 1151, PORT
HURON, MI 48061, (519) 344-3990.

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

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

INTEREST CALCULATIONS.
MAI-2.10 lets your computer help analyze
investment decisions. Calc: future value, present
value, annuities, sinking funds, loan pymt
sched., + more! Menu driven/Simple. IBM
ck/mo. Munier Associates, Inc., Dept. A5,
P.O. Box 79314, Houston, Texas 77279
(713) 784-4348.

*** * IBM-PCjr. OWNERS * ***
Learn to unleash jr's hidden powers!
How-to info from jr. experts, software tips,
freeware, best buys and more! Send $1.50 for
single issue or $18/yr. (12 issues) to:
JR. NEWSLETTER, Crider Associates,
Box 163, Southbury, CT 06488

FREE PROGRAMS! Apple/IBM PC/ TI99/
Timex/C64/+4/16/V20/Adam/TRS80 III/
4/100/200/CoCo/PC3/MC10/PCjr. Send
stamps! EZRAEZRA, Box 5222 TM, San Diego,
CA 92105

TEACHERS - MAGNUM GRADEBOOK
prepares student reports of assignments,
scores, averages. J. KNOWLES, 1025 Darling St.,
Ogden, Utah 84403. Specify Commodore 64 or
Apple IIe. Check or Visa: $29.

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

MISCELLANEOUS

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

Maxell MD1, $1.29-MD2, $1.99. Dysan 104/1D,
$1.79-104/2D, $2.39. Shipping $3.75. Also
Verbatim, IBM, 3M, BASF. TAPE WORLD, 220
Spring St., Butler, PA 16001, 1-800-245-6000.
Visa, MC.

DISK SALE! - SS/DD 35-trk for Apple w/sleeve
& label-10/$5.80, bulk-100/$45. Standard
SS/DD w/sleeve & label-10/$7.50, bulk-
100/$59. DS/DD w/sleeve & label-10/$8.50,
bulk-100/$67. 3½" SS for Mac-10/$19.99.
PREMIUM QUALITY, LIFETIME WARRANTY!
Money-back satisfaction guarantee! Min. order
$20. Send check or pay by MC/VISA/AE $3
shipping, + $2 if C.O.D. - UNITECH, 20 Hurley
St., Cambridge, MA 02141. (800) 343-0472, in
Mass. (617) "UNI-TECH".

EARN MONEY, PART OR FULL TIME, AT
HOME with your computer-manual & forms-
$9.95. Write Computer Programs for Profit!
How-to guide with forms, letters, tips-$7.95.
Also-Computer Consultant Handbook. How to
earn $ consulting with business-$7.95. JV Tech,
P.O. Box 563, Ludington, MI 49431

Save money shopping via your Home Computer.
Use phone modem for automatic shopping.
Send $3.00 for catalog and details to:
ZAK, 7787 Stout, Detroit, MI 48228

PROGRAMMERS . . . COPYRIGHT YOUR PROGRAMS. Protect your work. Included: documentation, forms & list of SOFTWARE CO.S all for
$7. Blue Cavern Software, 558 J St., Chula Vista,
CA 92010

Discount computer printer ribbons for all
makes/models Ex: Epson 1500 Nylon $6.99.
Catalog: TWS 1314 4th Ave., Coraopolis, PA
15108 (412) 262-1482 Visa or MasterCard.

COMMODORE DUPLICATOR 64
Now you can back-up copy
guarded software on the C64.
Our Disk-Based Duplicator is Now
Available At A Remarkable Price . . .
$34.95 . . . plus $2.50 shipping.
Major Crdit Cards & C.O.D. orders
accepted. We ship within 24 hrs.
DUPLICATING TECHNOLOGIES, INC.,
99 Jericho Tpke., Suite 302A,
Jericho, New York 11753. Daily (516) 333-5808;
Eve/Wknds (516) 333-5950

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

Rates: $25 per line, minimum of four lines. Any or all of the first line set in capital letters at no charge. Add $15 per line for boldface words, or $50 for the entire ad set in boldface (any number of lines.)
Terms: Prepayment is required. Check, money order, American Express, Visa, or MasterCard is accepted. Make checks payable to COMPUTE! Publications.
Form: Ads are subject to publisher's approval and must be either typed or legibly printed. One line equals 40 letters and spaces between words. Please underline words to be set in boldface.
General Information: Advertisers using post office box numbers in their ads must supply permanent address and telephone numbers. Orders will not be acknowledged. Ad will appear in next available issue after receipt.
Closing: 10th of the third month preceding cover date (e.g., June issue closes March 10th). Send order and remittance to: Harry Blair, Classified Manager, COMPUTE!, P.O. Box 5406, Greensboro, NC 27403. To place an ad by phone, call Harry Blair at (919) 275-9809.
Notice: COMPUTE! Publications cannot be responsible for offers or claims of advertisers, but will attempt to screen out misleading or questionable copy.

Advertisers Index

COMPUTE! **SUBSCRIPTION SAVINGS CARD**

Check one box:
☐ Payment Enclosed
☐ Bill Me
☐ Check Here If Renewal

SAVE EVEN MORE:
☐ 2 Years/$45.00

Please check if you own a
APPLE ☐ 01 ATARI ☐ 02
64 ☐ 03 VIC 20 ☐ 04
IBM ☐ 05 TEX INS ☐ 06
OTHER__________ 99
☐ Don't yet have one

Name ____________________

Address ____________________

City ____________________

State __________ Zip ________

Foreign and Canadian subscribers, please add $6 (U.S.) per year postage. Offer subiect to change without notice.

You'll receive 1 year of COMPUTE! for only $24.00. That's a savings of 32% OFF the cover price.

COVER PRICE FOR 1 YEAR	**$35.40**
YOUR PRICE	**$24.00**
YOU SAVE	**32%**

J1205

Only $24.00 for 12 issues

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 7478 DES MOINES, IOWA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPUTE!

PO BOX 10954
DES MOINES, IOWA 50347

COMPUTE!'s
FREE Reader Information Service

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

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

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

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

COMPUTE!

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

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

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

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

Name ______________________

Address ______________________

City ______________________

State/Province ______________ Zip ______

Country ______________________

Please Include ZIP Code   Expiration 4/30/86   C0386

SUBSCRIBE TO COMPUTE!

For Fastest Service,
Call Our **Toll-Free**
US Order Line
800-334-0868
In NC call 919-275-9809

My Computer Is:

01 ☐ Apple 02 ☐ Atari 03 ☐ Commodore 64
04 ☐ VIC-20 05 ☐ IBM 06 ☐ TI-99/4A
99 ☐ Other______________ ☐ Don't yet have one.

☐ $24.00 One Year US Subscription
☐ $45.00 Two Year US Subscription

(Readers outside of the US, please see our foreign readers subscription card or inquire for rates).

Name ______________________

Address ______________________

City ______________ State ______________ Zip ______

☐ Payment Enclosed ☐ Bill me
Charge my: ☐ VISA ☐ MasterCard ☐ American Express
Account No. ______________ Expires ____ / ____

Your subscription will begin with the next available issue. Please allow 4-6 weeks for delivery of first issue. Subscription prices subject to change at any time.
The COMPUTE! subscriber list is made available to carefully screened organizations with a product or service which may be of interest to our readers. If you prefer not to receive such mailings, please check this box ☐

Place
Stamp
Here

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

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

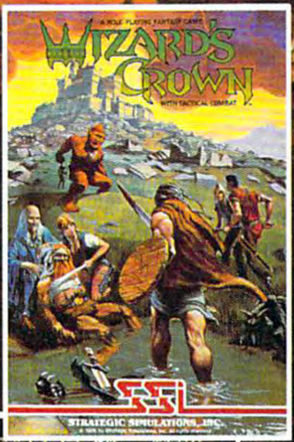
COMPUTE!
PO BOX 10954
DES MOINES, IOWA 50347

ONLY A FANTASY GAMER COULD CALL THIS HEAVEN.

If exploring eerie dungeons filled with monsters is your idea of fun, we've got two fantasy games that'll have you floating on cloud nine. Each breaks new ground in role-playing games with special features:

WIZARD'S CROWN™ lets you resolve combat two ways: The computer can do it quickly, or you can personally direct it with a multitude of tactical options.

RINGS OF ZILFIN™ adds unprecedented realism to fantasy gaming with its superb graphics. The fully animated scrolling screen grants you step-by-step control of the action.

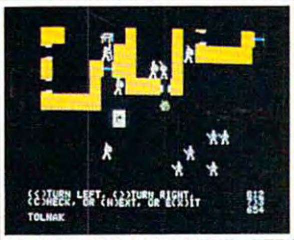


The gates of heaven are your local computer/software or game store. Enter them today.

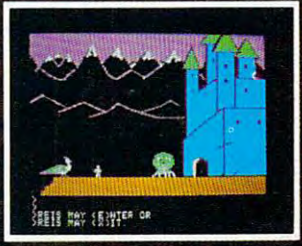
If there are no convenient stores near you, VISA & M/C holders can order these $39.95 games by calling **toll-free 800-443-0100, x335.** To order by mail, send your check to: **STRATEGIC SIMULATIONS, INC.,** 883 Stierlin Road, Building A-200, Mountain View, CA 94043. (California residents, add 7% sales tax.) Please specify computer format and add $2.00 for shipping and handling.

All our games carry a "14-day satisfaction or your money back" guarantee.

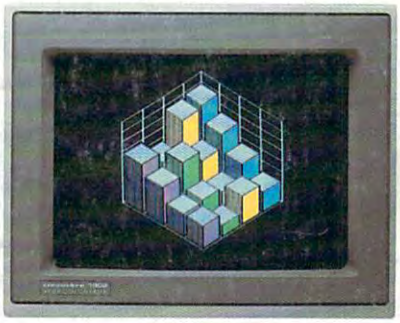
WRITE FOR A FREE COLOR CATALOG OF ALL OUR GAMES TODAY.



ON DISK FOR 48K APPLE® II SERIES AND C-64™.

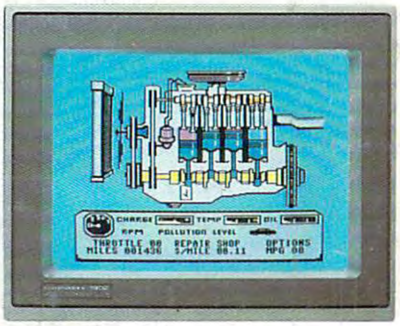


All you need to do this



graph a spreadsheet

write a novel



fix an engine

compose a song

paint a picture

your banking

learn to fly

organize a data base

tell a story

forecast sales

When it comes to personal computers, you want the smartest, at a price that makes sense.

The new Commodore 128™ system has a powerful 128K memory, expandable by 512K. An 80-column display and 64, 128 and CP/M® modes for easy access to thousands of educational, business and home programs. And a keyboard, with built-in numeric keypad, that operates with little effort.

Or if the Commodore 128 is more machine than you had in mind, you can pick up the Commodore 64®. The Commodore 64 is our lower-priced model geared to more fundamental, basic needs.

Discover personal computers that do more for you. At prices you've been waiting for. From the company that sells more personal computers than IBM® or Apple®

COMMODORE 128 AND 64 PERSONAL COMPUTERS

A Higher Intelligence