COMPUTE!'S GAZETTE

DISK MAGAZINE

FOR THE C-64/128

JANUARY 1995

```
*****  *****  *   *  *****  *   *  *****  *****     *  *****
*      *   *  ** **  *   *  *   *    *    *        *   *
*      *   *  * * *  *****  *   *    *    ***     *    *****
*      *   *  * * *  *      *   *    *    *               *
*****  *****  * * *  *      *****    *    *****       *****
```

G A Z E T T E   O N   D I S K

J A N U A R Y   1 9 9 5

Final Disk Magazine Published! Compute!'s Gazette Ends!!
Wrong! See Next Issue!

Check for other Commodore-Published Magazines & Disk-Magazines

Commodore World
(c/o "Creative Micro Designs, Inc.")
P.O. Box 646
East Longmeadow, MA 01028

Phone: 1-800-638-3263
** 8 Issues Per Year
** 54 Pages ** USA: $29.95
** Canada: $ 35.95, Other=57.95

SoftDisk Publishing
P.O. Box 30008
Shreveport, Louisiana 71130-0008
("Loadstar 64" or "Loadstar 128")

Phone: 1-318-221-8718
** LS-64 US: $89.95 (One-Year)
** LS-128 US: $39.95 (4 issues)
** Call for other countries.

DieHard Magazine
(c/o "LynnCarthy Industries")
P.O. Box 392
Boise, ID 83701-0392

Phone: 1-208-383-0300
** Magazine Only/6 iss.: $16.97
** Magazine w/Spinner: $ 55.97
** 34 Pages/Magazine

GAZETTE DISK *** January 1995

Feature:

The Numbers Game

By Don Radler

Take A Look At Line Numbers. They Can Make BASIC Programming Easier

Columns:

64/128 VIEW by Tom Netsel
COMPUTE magazine and th old COMPUTE's Gazette are holding a garage sale. Now's a good tim to stock you library.

D'IVERSIONS by Fred D'Ignazio
Where in the World is Fred D'Ignazio?

MACHINE LANGUAGE by Jim Butterfield
Flags and Branches

BEGINNER BASIC by Larry Cotton
Evolution of a Program

PROGRAMMER'S PAGE by David Pankhurst
Security

GEOS by Steve Vander Ark
Lost Trasures

PD PICKS by Steve Vander Ark
Life, Disk Checker, Disk Display, and Whatzit

64 Programs:

Text Scroll by Daniel Lightner
Text Scroll is a utility for writing fancy title screens that can be used in your own BASIC programs.

Drive Error Tone Generator by James T. Jones
Whenever you experience a disk error this utility sounds a tone to get your attention and prints the error message in reverse video in the upper left-hand corner of the screen.

Lodraw2 by Robert Quinn

Lodraw2 is a low resolution draw and paint program that uses virtually all the alphanumeric and graphic characters of the 64 character set, as displayed on the keyboard.

The Value of Money by Earl Woodman
The Value of Money is a program for the 64 that will allow the user to calculate present, future, and annuity values when given one of the other two.

128 Programs:

YOUR 3D by D.G.Klich
You may remember a program called Stereogram 128 that created a series of three dimensional pictures using the "random dot stereogram" technique. This redesigned version will let you merge up to twenty of your own screens into a stereogram picture each residing at a different depth.

CYCLEDATA128 by D.G. Klich
Use this utility on your multigeared bicycle to determine which gear to use and to determine the order of settings to get a smooth increase in speed or power.

Gazette, January 1995

Text Scroll

By Daniel Lightner

Text Scroll is a utility for writing fancy title screens that can be used in your own BASIC programs. Text Scroll displays a moving colorful border as it scrolls your own messages acrosss the screen. The messages scroll from the center of the screen line outward in both directions giving your program a touch of class.

Text Scroll is menu driven, which means that little instruction is needed to use it. Just load and run Text Scroll and follow the prompts for editing a screen. Here is a brief description of the Text Scroll features and commands accessed from the menu.

L to load an already created screen.

S to save a new screen.

E to edit a screen line.

V to view the text.
D to display the scroll effect.

* to write a new text screen.

C to change the character color.

Q to exit to BASIC.

Text Scroll allows 14 lines containing 32 characters each. Each line will be padded with periods to fill out th 32 characters, for example.

A LINE FROM TEXT SCROLL ....

Notice the row of periods on the end. To eliminat the priods, you may want to fill out a line with shifted spaces. You may want to start a line with several shifted spaces to center text. You'll get a number of ideas for displaying text once you run the program a few times. All nontext characters will be displayed in light blue. You may choose text color though.

After a screen has been created and saved, it can be loaded from a BASIC program by putting this line in the start of the program.

```
1 X=X+1:IF X = 1 THEN LOAD "FILENAME",8,1
```

FILENAME is used here just as an example, the files name can be any legal Commodore filename that you choose.

After you've loaded your screen into memory, you can display it at any time by entering SYS 49152. Text Scroll screens don't occupy any of BASIC's memory.

Gazette, January 1995

Drive Error Tone Generator

By James T. Jones

Have you ever been so deeply engrossed in programming that you didn't notice that the light on your disk drive was flashing? Well, I have, and more often than I'd care to admit.

It would be useful to have an interrupt driven, machine language program that would remain in the background while you are programming, until a drive error occurs. Then it would sound a tone to get your attention and print the error message in reverse video in the upper left-hand corner of the screen.

"IRQ DRIVE TONE.O" performs these functions. It resides in the casette buffer and has a starting address of 820.

The interrupt routine associated with the program can be turned off by typing SYS 823 and pressing Return. This is required if you are going to load a program from or save a program to disk or verify that a program in memory is the same as the one on the diskette.

Simply load IRQ DRIVE TONE.O with the ,dv,1 extension, where dv is the device number of your disk drive, and type NEW then SYS 820. Device numbers ranging from 8 through 15 are supported. As a demonstration, load a file having a fictitious file name. If the tone continues at a low volume, type the following poke in immediate mode and then press Return.

```
POKE 54296,0
```

Now you'll have a vigilant watchdog beside you to warn you, so that you can stop and decide why the drive error occurred.

If you like, load the associated program from the Gazette Disk menu and it will perform a short demonstration for you by trying to load a file that is not on disk. As stated above, to use th program with your own programming sessions, simply load IRQ Drive Tone.O and activate it by typing SYS 820.

The source file, IRQ DRIVE TONE.S, may be found on this disk in Merlin format and has a large number of comments included. The file is completely relocatable to any starting address where a sufficient number (168) of free, protected bytes is available.

The following discussion is for those programmers who do not have the Merlin assembler and are interested in the details of the program. From the discussion, a source file may readily be written using another assembler.

```
1      org   $0344
```

```
2 open      =$ffc0
3 setlfs    =$ffba
4 setnam    =$ffbd
5 chrin     =$ffcf
6 clrchn    =$ffcc
7 chkin     =$ffc6
8 chrout    =$ffd2
9 close     =$ffc3
10 status   =$90
11 plot     =$fff0
12 v        =$d400
13 irqrtn    =$ea31
14 vecirq    =$314
15 number    =120
16
17      jmp   init
18 reset     sei
```

No interrupts must occur while the irq vector is being reset

```
19      lda   #$31
20      ldy   #$ea
21      sta   vecirq
22      sty   vecirq+1
23      cli
24      rts
```

The irq vector points to the normal interrupt routine at $ea31 and the operating system is reset to normal interrupts with a SYS 823.

```
25 initsei
26      lda   #<drbell
27      ldy   #>drbell
28      sta   vecirq
29      sty   vecirq+1
```

Point the IRQ vector to the drive check, bell routine.

```
30      cli
31      rts
32 drbell    dec   count
33      beq   reload
34      jmp   irqrtn
35 reload    lda   #number
36      sta   count
```

Is the counter zero? If not, a JMP to the normal IRQ routine is made. If so, reload counter and check error channel. With number=120, the error channel will be checked about every two seconds.

```
37      lda   #1
38      ldx   $ba
39      cpx   #8
```

```
40      bcs   okdevice
41      ldx   #8
```

The accumulator is loaded with the logical file number (lfn=1). Location 186 ($ba) is the location of the device number. If the device number is 8 or greater, a branch is made to OKDEVICE. These lines are equivalent to the BASIC line,

```
DV=PEEK(186):IF DV<8 THEN DV=8
```

```
42     ;blank line
43 okdevice ldy   #15
44      jsr   setlfs
```

Set the logical file number (1), device number (dv) and command number (secondary address), 15.

```
45      lda   #0
46      jsr   setnam
```

There is no filename.

```
47      jsr   open
```

In BASIC, this would correspond to open 1,dv,15.

```
48      ldx   #1
49      jsr   chkin
50      jsr   chrin
```

Open the error channel for input, then get a character from the error channel.

```
51      pha
```

Save the contents of the accumulator.

```
52      cmp   #48 ;zero?
53      bne   errmsg
54      pla
55      jsr   clrchn
56      lda   #1
57      jsr   close
58      jmp   irqrtn
```

If there is no drive error, pull the value that the accumulator had from the stack, clear all channels and restore I/O channels, close the logical file number and return to the normal interrupt routine.

```
59 errmsg   clc
60      ldx   #$00
61      ldy   #$00
62      jsr   plot
```

```
        lda    #18
        jsr    chrout
```

The error message will be displayed in the upper left-hand corner of the screen in reverse video.

```
        pla
getbyt    jsr    chrout
        jsr    chrin
        bit    status
        bvc    getbyt
```

64=2↑6 so that bit 6 is set at end of the error message; if bit 6 is clear, then a branch to GETBYT is made to get more characters.

```
        lda    #146
        jsr    chrout
```

Turn the reverse video off. The following lines will generate a bell sound.

```
bellldx    #0
        lda    #0
l1      sta    v,x
        inx
        cpx    #24
        bne    l1
```

Clear the registers of the Sound Interface Device (SID).

```
        lda    #70
        sta    v+1
        lda    #249
        sta    v+6
        lda    #15
        sta    v+24
```

Maximum volume.

```
        lda    #17
        sta    v+4
        lda    #16
        sta    v+4
        jsr    clrchn
        lda    #1
        jsr    close
        jmp    irqrtn
```

Clear all channels and restore I/O channels, close the logical file number and JMP to the normal interrupt routine.

```
count     dfb    number
```

Lodraw 2

(Part 1)

By Robert Quinn

LODRAW2 is a low resolution draw and paint program that uses virtually all the alphanumeric and graphic characters of the C64 character set, as displayed on the keyboard. Includes a character editor so you can redesign characters and create your own character sets, with which to draw designs. You can also design and use extra large QUADE characters.Your designs and character sets can then be saved to and loaded from disk.

When you run Lodraw2 for the first time after loading, Lodraw2 will load a LODRAWSUITE3 file of machine code routines from the disk. You can quit Lodraw2 by holding down the Commodore key (C=) and pressing the Run-Stop key. If you re-run Lodraw2, LODRAWSUITE3 will not be loaded again. If you copy Lodraw2 to another disk, make sure you copy LODRAWSUITE3 to that disk also.

A flashing, rectangular pen will appear at center of a drawing pad that occupies all of the screen except for the bottom line.

The pen is moved around the pad (up, down, left, right) with a joystick in port 2 or by pressing the cursor keys (by themselves or with Shift held down). The comma key and the dot key can be used to move the pen left and up respectively without having to hold down Shift. If the pen is moved past the screen border, it will appear on the opposite side of the screen.

Pressing Return will move the pen to the left side of screen and down one line.

Hold down (C=) (the Commodore logo key at bottom left of keyboard) and press Control to move the pen five places to the right, for fast movement of the pen.

Press fire button on joystick and simultaneously push up, down, left or right on joystick to move pen to top, bottom, left or right edge of pad.

MENUS

Press Run-Stop then press M to access three screen menus that list just about all the facilities available on Lodraw2 and the key strokes that access them. A left arrow in front of a key in the menus means you have to press the left arrow key (top left of keyboard) first.

For further on-screen information about many of the facilities of Lodraw2, press Run-Stop then press the key that accesses the facility you want information about. A help message will scroll across the bottom of the screen. Most facilities are catered for, and for most of these you do not have to press the left arrow key first to obtain the help message.

Press the left arrow key then press M to display an introductory message.

BACKGROUND AND BORDER COLORS
The black background of the pad can be changed to any of the 16 colors available on the C64 by pressing INST DEL until the background you want appears. Hold down Shift and press INST DEL to cycle through the 16 border colors.

The bottom screen line is the Indicator Line, indicating the status of several dual-modes of Lodraw2, and displaying other information about Lodraw2. The line above it contains two pen color indicator bars.

The five dual-modes of Lodraw2 are Graf/Alfa, Rvs off/on, Shift/(C=), Draw/Keep and Type off/on.

Initially, Lodraw2 is in Graf, Rvs off, Shift and Draw modes, as shown on the Indicator Line. The number to the left of each mode indicator is the function key that switches between the dual modes. Lodraw2 is also in Type off, indicated by the rectangular shape of the pen.

Moving the pen in Draw mode will draw the character that has last been assigned to the pen (the pen character) and paint the last chosen pen color. The initial pen character is the pen rectangle, which has been assigned to screen code 0. The current pen character always displays at the center of the Indicator Line. The current pen color displays in the two color bars above the Indicator Line.

Press f7 or press fire button on joystick to switch between Draw and Keep. In Keep the pen can be moved around the pad without erasing the characters it passes over or their colors: the pen neither draws nor paints.

By holding down Shift and pressing Return you can switch to PainT only mode: in Paint the pen will not draw the pen character, but will change the color of those characters it passes over to the pen color. Press f7 to quit Paint.

The number that appears right of center of the Indicator Line, and which changes as you move the pen over different characters on the pad, is the screen character code of the character currently under the pen (32 is the code for the space character).

In Graf mode you can redefine the pen character to be any one of the graphic characters on the keyboard without having to hold down Shift or (C=): simply press a key on which a graphic character is imprinted. The new pen character will display in the middle of the Indicator Line. In Shift mode, the new pen character will be the graphic character imprinted on the right side of the key. In (C=) mode, the new pen character will be the graphic character on the left side of the key you pressed. Alternatively, you can select a graphic character for the pen by holding down (C=) or Shift and pressing the key on which the graphic character is imprinted. With Rvs on, the newly defined pen character will be the reverse form of the character.

PEN COLOR
Still in Graf mode, the number keys 1 to 8, pressed by themselves or with (C=) held down, select the pen color from the 16 colors available, displaying the pen color in the two color bars on the second line up from bottom of screen. Since this line is part of the pad on which you can draw, the pen color bars can be switched off by pressing the left arrow key (top left of keyboard: this is the Select key) and then pressing N. Pressing these keys again will switch the bars on without changing anything you may have drawn on this line. You can draw on this line even with the color bars switched on.

Graf/Alfa
Press f1 to switch between Graf and Alfa. In Alfa mode, pressing a key, by itself or with Shift or (C=) held down, will define the corresponding alphanumeric or graphic character as the pen character. Since pressing a number key 1 to 8 by itself now defines the corresponding number character as the pen character, the first eight pen colors are now selected by first pressing the left arrow key (the Select key) and then pressing a number key 1 to 8. Do not hold down the Select key. In Alfa mode, the comma (,) and period (.) are not cursor keys.

You can define the pen character by screen character code: press = and you will be prompted to enter a screen character code (0 to 255); press Return if the character code you enter is less than three digits long. The character assigned to that code will now be the new pen character. If you want to define the pen character as the equal sign (=), press Select (left arrow key) first, then =.

You can also define the pen character by placing the pen over any character on the pad, then hold down Shift and press space bar: the pen character is defined as the character under the pen.

VIEW CHARACTER SET
Press Select then Y to rapidly view all the characters in the character set, starting from the current pen character. Press comma (,) to decrement or press period (.) to increment through the character set. If you see a character you want, you can assign it to the pen by pressing Return. Press any other key to quit.

F3 switches between Rvs on and Rvs off. With Rvs on and a normal pen character, the pen will draw the current pen character in reverse form. If pen character is reverse then, with RVS on/off, the pen will draw the normal/reverse form of the character.

CLEAR SCREEN
Hold down Shift and press Clr-Home to completely erase the draw pad.

But don't despair if you have been too quick to get rid of your current design: you can recover your design from a buffer used by Lodraw2 by pressing Select and then pressing the up arrow key (to the left of the Restore key). This Undo function can also be used to recover a design after many of the design altering operations yet to be described have been carried out.

USER BUFFERS

You will have noticed a number 1 at the far right of the Indicator Line. If you press Select then press Q, this number will switch between 1 and 2. The number indicates which of two user buffers is currently available for storing and retrieving a design along with its color and the character set used by it. So, in addition to the current design on screen, and perhaps an older design in the Lodraw2 buffer, you can have a further two designs stored in the two user buffers.

PUT

To store the on-screen design and its color and character set in the current user buffer, hold down Shift and press the second function key from the top (f4).

GET

To retrieve a design/color/character set from the current user buffer, hold down Shift and press the top function key (f2).

PROJECTION NUMBER

This number, used in a variety of Projection facilities, displays left of center of the Indicator Line. It can be changed by pressing CONTROL until a projection #? prompt appears on left of Indicator Line, then pressing any of the number keys (0 to 9). You can add 10 to the current projection number by pressing 0, or Select, then 0 in Alfa mode or Type on. This gives a range of 0 to 39 for the projection number.

HORIZONTAL Shift

You can displace a row of characters left or right away from the pen: simply place pen on start character of row and press : to move row left or press ; to move row right (in Alfa mode or Type on mode, press Select first). Each press of one of these keys will shift the row one position. The number of characters shifted (the length of the row) is the projection number. If the projection number is 0, then all characters up to the first space character will be shifted. These Shift facilities wrap-around the screen.

VERTICAL Shift

You can likewise displace a column of characters up or down away from the pen. Press / (key left of right Shift key) to move a column down or press Select then press P to move a column up. Further details as for Horizontal Shift.

QUADES

A quade is a pattern of four successive characters in the character set which can be drawn in a square to the right of the pen by pressing

Clr-Home. The first character in a quade is the current pen character. To get the idea of the arrangement of a quade, define the pen character as 'A' or '1' and then press Clr-Home. Now define the pen character as the back-slash character on the 'M' key and draw the graphic quade that results from that. By using the Character Editor to redesign groups of four characters with successive screen character codes, you can design quade characters twice as tall and twice as wide as commodore characters.

SELECT ON/OFF SWITCH
There are many Lodraw2 facilities which are accessed through the Select (left arrow) key. Press Select key to switch Select on: a rectangle will appear on the left of the Indicator Line. Pressing Select again will switch Select off if you change your mind about using it: a will replace the rectangle. Select is also switched off whenever any operation you have selected is completed.

DISK SAVE
Hold down Shift and press the bottom function key (f8) to save to disk a copy of the current on-screen design. The letters SV (Save Video) will appear on the left of the Indicator Line and scrolling instructions will tell you what to do. You can terminate any scrolling message by pressing f7. File names can be up to 16 characters long. The first character of a file name is a left arrow character: this is inserted by Lodraw2, so you need only type in the remainder of the file name of your choice, up to 15 characters. If you make a mistake while typing the file name, you can press the right-cursor key to erase the file name one character at a time to the left. Pressing f7 will quit Save if you change your mind. A file contains the design on the pad, color information for the entire pad, background color and the current character set.

DISK LOAD
Hold down Shift and press the second function key from the bottom (f6) to load a Lodraw2 design/color/character set file from disk. Letters LV will appear on left of Indicator Line. The design will appear on screen with original colors, characters and background color. You do not type in the first (left arrow) character of the file name; Lodraw2 does this for you.

SELECT ON FACILITIES
For each of the following facilities you must first press the Select key to switch Select on, then press the key that activates the facility. A Left arrow will be used to indicate the keys to press for facilities that require Select on.

DISK SAVE/LOAD CHARACTER SET:
F8/

F6
Similar to previously described disk save/load facilities, but only the character set is saved/loaded. The first character of a file

name, inserted by Lodraw2 so don't type it, is a @ (within Lodraw2 the @ character appears as the pen rectangle character). This is used, rather than a left arrow, to distinguish character set only files from screen/color/character set files.

DISPLAY DISK DIRECTORY
D - Your design will be restored when you finish using this facility.

ERASE FILE FROM DISK
E - You must type the first character of the name of a file since Lodraw2 cannot know if it is a video/character set file (left arrow) or character set only file (@).

DRIVE NUMBER switch
* - Switches drive number between 8 and 9; the default disk drive number is 8. This is the number used by Lodraw2 to save and load files to/from disk. The switch does NOT change the actual device number(s) assigned to the disk drive(s) connected to your computer.

BOX PROJECTION
B - A square, made with the pen character & color, is drawn from the pen, with side length equal to the projection number (first projected character is under the pen). If, due to location of pen, a square of that length cannot fit on the pad, then a rectangle will be drawn that does fit on the pad.

ORTHOGONAL PROJECTION:
cursor or joystick - A line, made with the pen character & color, is drawn from the pen left, right, up or down, with length equal to the projection number (first projected character under pen). If necessary, the line will project to the other side of the screen.

(End of part 1. See Part 2 for more Lodraw instructions.)

Gazette

Lodraw 2

(Part 2)

By Robert Quinn

DIAGONAL PROJECTION:
f1 or
f3 or
f5 or
f7 - A diagonal line, made with the pen character & color, is drawn from the pen with length equal to the projection number (but does not exceed screen border and first projected character under pen).

f1 up-left;
f3 up-right;
f5 down-left;
f7 down-right.

There are three more specialised forms of diagonal projection, which are accessed by pressing Select then pressing either J or K or L then pressing one of four function keys (f1, f3, f5, f7). The J and K projections each use two pairs of characters. Where C is the screen character code of the current pen character:

J projection uses character codes C & C+1 with f1 and f7; and C+2 & C+3 with f3 and f5.

K projection uses codes C+4 & C+5 with f1 and f7; and C+6 & C+7 with f3 and f5.

L projection uses character code C-2 with f1 and f7; and C-1 with f3 and f5.

You may decide that this is all too complicated and never use these diagonal projection facilities. A Lodraw2 video file (filename RAYS) is included on the disk to demonstrate what can be done with them.

In all these projection facilities, with RVS on the inverse of the pen character will be drawn (the normal character and the reverse character are inverse of one another).

TYPE ON
A switches to Type on mode and the pen displays as a left arrow character. Lodraw2 is now used like a typewriter. Press any character key (except left arrow key which remains the Select key), by itself or with Shift or (C=) held down, and the corresponding character will appear at the current pen position; the pen then moves one position to the right. This typewriter action works in Draw and Keep. The pen, however, still draws the pen character & color when it is moved using cursor keys or joystick in Draw mode; switch to Keep mode if you do not want this to happen. Dot and comma keys are not cursor keys when

in TYPE on, and you must use Select and a number key to define one of the first eight colors as pen color, and use Select and = to define pen by screen character code. Otherwise, except for Shift/(C=) modes selection of graphic characters for the pen, all the facilities of Lodraw2 operate in TYPE on as they do in Type off. In Alfa mode, whenever a character is typed that character also becomes the pen character.

Press left arrow and then A to switch Type off.

HIDE/SEE PEN switch
H - to make pen invisible/visible.

V - Replace every normal/reverse character on the pad with the reverse/normal form of that character.

G - Same as Left Arrow-V except normal space characters are not reversed.

To use the following Replace facilities, you must first place the pen over a character on the pad that you want to change (replace) wherever it occurs throughout the entire pad.

S - Replace the color of all characters on the pad that match the character and color under the pen, with the pen color.

Z - Replace the color of all characters on the pad that match the character (only) under the pen, with the pen color.

C - Replace all characters on the pad that match the character under the pen, with the pen character.

F - Replace all characters on the pad that match the character and color under the pen, with the pen character.

X - Replace all characters on the pad that match the (normal/reverse) character under the pen, with the reverse/normal form of that character.

UNDO
up arrow (↑) - All the changes produced by a Projection, Replace or Get operation can be undone, restoring your original design. But keep in mind that the Lodraw2 buffer only stores video and color information, unlike the User buffers which also store the character set.

SWAP
W - This facility transposes the pen character with the character on the pad that is currently under the pen, without swapping their screen character codes. The changes are made in the character set to the data associated with the two character codes. Result: instant change of all characters on the screen associated with the two character codes.

ROTATE CHARACTER
R - Rotates the current pen character a quarter turn in a clockwise direction and, consequently, rotates all characters on the pad that have the same screen character code.

ROTATE QUADE
T - Rotates the quade that begins with the current pen character; all characters on the pad that have any one of the four screen character codes that define the quade will be changed accordingly.

CHARACTER EDITOR
Inst-Del
The character editor allows you to redesign any character in the character set used by Lodraw2.

You have two options of selecting a character for display.

Option 1 - Press f7 key, then type in the SCREEN DISPLAY CODE of the character (as listed in appendix E of the Commodore 64 User Manual). These codes, including reverse image codes, range from 0 to 255. If the character code is one or two digits long, you must press Return after typing the code. A three digit code is auto-accepted as soon as the third digit is typed in, and a bell sounds.

Option 2 - Alternatively, you may press any character key on the keyboard, by itself, or with (C=) or Shift held down. The corresponding character will then display.

The character with screen display code 0 has been redefined by Lodraw2 as a boxy graphic character, because this character is used by the character editor to display characters.

The display of the character you have selected lists the sequence of eight bytes that define the character in the character set and the address in RAM where Lodraw2 stores each byte.

The screen display code of the character appears to the right of the blue line, with the actual character displayed below the code, and to the right of this the quade pattern of four successive characters beginning with the selected character.

An eight byte by eight bit blow-up display of the character (the Edit) appears below the blue line. Each bit is either a blank (= 0) or a box (= 1), as determined by the byte that appears to the left of each line of the EDIT.

A flashing asterisk cursor will appear in the upper left bit position of the EDIT and a menu of Edit facilities will display in lower half of screen.

You can return to start of View Character selection by pressing f7. Pressing Inst-Del will quit the character editor and your design will be restored to the screen.

Or you can press f3 to display the next character in the character set (increment) or press f5 to display the preceding character (decrement).

If you choose to increment to the next character in the character set, then the quade that displays will be the same quade that appeared when you selected your first character. This allows you to redesign the four successive characters in the quade, one after the other, and to view the resulting quade pattern. If you increment four times in succession, the new character will become the start of the current quade.

You can move the cursor either from the keyboard or with the joystick.

Press space bar to move the cursor down.

Press comma (,) to move cursor left.
Press period (.) to move cursor right.
Press any letter key from D to L to move cursor up.

The cursor keys will also move the cursor.

All movements of cursor are wrap-around.

At any position on the Edit, press Return or fire button on joystick to switch the bit under the cursor. If the bit was 0 (a blank), it will be set to 1 (a box will appear); if the bit was 1 (a box), it will reset to 0 (a blank will appear). Changing any bit will change the byte that contains the bit, as displayed to the left of the EDIT, and change the corresponding character in the character set, as shown in the character display on the right.

Z BLANK CHARACTER
This zeroes the character, blanking all the bits, making a space character.

X FILL CHARACTER
This sets all the bits, making a full (reverse space) character.

A BLANK ROW
The row the cursor is located on is blanked (the byte is zeroed)

S FILL ROW
The row the cursor is located on is filled (the byte set to 255).

Q BLANK COLUMN
The column the cursor is in is blanked (the corresponding bit is reset for all eight bytes making up the character).

W FILL COLUMN
The column the cursor is in is filled (the corresponding bit is set for all eight bytes).

V REVERSE
Every set/reset bit is reset/set, changing the character to its reverse image; using this key again will restore the normal image character.

B VERTICAL INVERSE
The character is flipped up side down.

N HORIZONTAL INVERSE
The character is flipped left to right.

C COPY CHARACTER
This facility allows you to copy the current character being edited into any character (other than the character with screen code 0) in the character set. Simply type in the screen code of the character you want to copy to, pressing Return if the code has less than three digits. Press f7 if you change your mind and don't want to copy.

R ROTATE CHARACTER
Rotates the Edit character a quarter turn in clockwise direction.

P EXPAND TO QUADE
Enlarges the current edit character into the quade that starts from that character.

Y QUADE EDIT
This key sends you to a quade edit screen with a blow-up EDIT of the quade and a menu of quade editing facilities.
You can move the cursor to any bit in the 16 X 16 EDIT and switch it on or off. This allows you to design a quade as a single character, editing the four characters that make up the quade on a single screen.

The blinking curser is moved using the same keys (or joystick) that are used in the character Edit.

Return or fire button to set/reset the bit under the cursor.

Or press f7 to switch to cursor edit. Moving the cursor will now erase (reset) all the bits the cursor passes over.

Press f5 to switch to draw. Moving the cursor will now draw (set) all the bits the cursor passes over.

f5 switches between draw/erase. f7 switches between cursor edit on/off.

B moves all rows of the quade down one line, the bottom row going to the top.

S shifts all columns of the quade to the right by one column, the rightmost column going to the leftmost.

C COPY QUADE
Allows you to copy the four successive characters making up the quade to another location in the character set. Type in the screen character code (but not 0 or any code above 252) you want the quade copy to start from.

The other quade facilities (Rotate, Horizontal Inverse, Reverse, Blank, Fill) do the same to the quade as the corresponding character edit facilities do to a character.

R ROTATE QUADE
Rotates quade a quarter turn in clockwise direction. Handy to use with COPY QUADE: duplicate a quade and rotate the duplicate. Three copies of the original quade can be made so that each of the four has a different orientation.

Y QUIT
Quade edit and return to character edit.

M AUTO EDIT switch
The character editor is in auto edit on when this option appears white in the menu. If you have once before designed a new character and have a ready list of the eight bytes that code for it, you can press M to switch auto edit off. If you now press f7 (or f3 or f5) and select a character for editing, the character editor will give you two options: Byte Edit or Bit Edit. If you press Return for byte editing, you can type in your eight new bytes (pressing Return to complete entry of any one or two digit byte, with auto-entry for three digit bytes). A blow-up Edit of the new character will appear, line by line, as you enter each byte. For any of the eight bytes, if you press Return only, the old byte for the character will be entered unchanged.

At any time while typing in a byte, you can press f7 or fire button to quit Edit Character mode with all remaining bytes unchanged from old character.

When all eight bytes have been entered you can press f7 key or fire button or Return to go back to View Character mode with character changes made in Edit Character mode intact. Or press f3 or f5 for increment or decrement.

The character editor remains in auto edit off until you press f7 (or press fire button) to select the bit edit option and then press M to switch auto edit on again.

Gazette, January 1995

The Value of Money

By Earl Woodman

The Value of Money is a program for the 64 that will allow the user to calculate present, future, and annuity values when given one of the other two. Calculations can be made in either discrete compounding mode or continous compounding mode.

The options are:

1. Calculate the present worth of an amount of money, when you know the final worth. (COMPOUND AMOUNT)

2. Calculate the future worth of an amount of money, when you know the present worth. (PRESENT WORTH)

3. Calculate a payment per period amount, where you know the present amount. (SINKING FUND FACTOR)

4. Calculate a payment per period amount, where you know the future amount. (SERIES COMPOUND AMOUNT)

5. Calculate a future amount when you know the payment per period. (CAPITAL RECOVERY)

6. Calculate a present amount when you know the payment per period. (SERIES PRESENT WORTH)

7. Calculate a payment per period amount when you have a uniform increasing series of payments. (ARITHEMTIC GRADIENT - This option is not available with continuous compounding.)

Each of these selections will ask you for the monetary value, the interest rate per period, the number of periods, and then will calculate the desired answer.

For example, if you wanted to determine the how much money 1000 dollars would be worth in ten years time, at an interest rate of 8%, you would cursor up or down to the Compound Amount selection, press space, and then respond to the prompts. You would find that the result is $2158.93.

STARTING THE PROGRAM

The Value of Money is written entirely in BASIC. To start the program, type LOAD "VALUE OF MONEY" ,8
A screen will appear with seven choices:
COMPOUND AMOUNT, PRESENT WORTH, SINKING FUND FACTOR, SERIES COMPOUND AMOUNT, CAPITAL RECOVERY, SERIES PRESENT WORTH, and ARITHMETIC GRADIENT, corresponding to the 7 choices listed in the first section.

Use the cursor up and down keys to highlight the desired choice, and space to select it. You can also press H at the menu screen for a brief help screen, Q to quit, and C to toggle the compounding mode. The compounding mode currently in effect appears at the bottom of your screen.

Gazette, January 1995

PD PICKS
The public domain programs on this disk consist of Life, Whatzit, Disk Checker, and Disk Display. These programs are discussed in length in Steve Vander Ark's column "PD Picks."

LIFE
When you run Life from the Gazette Disk menu, the program loads an instruction file which in turn loads the actual game. Once you are familiar with the rules of the game, you can bypass the instruction file and simply load the game by loading the file LIFE. Abbreviated instruction are built into the game.

DISK CHECKER
DISK DISPLAY
Disk Checker and Disk Display take the raw data from your disk and show you what you really need to know from it. They do the interpreting for you. Disk Display, for example, will show you not only what information resides at a particular track and sector but also a list of the linked disk locations that make up a file and a visual representation of the disk's BAM.

Disk Checker actually tests things out for you. The whole process can take a while as the program tests start links, compares the file length as given by the directory to the actual length, checks for overlapping files, and so on.

WHATZIT
You use a joystick to zip around and choose letters in order to unscramble a word in the category of your choice. Your score is based on how fast you can put the word together.

Gazette, January 1995

64/128 VIEW: Garage Sale

By Tom Netsel

As I suppose most of you know by now, COMPUTE magazine is no more. It was bought and closed in on of those megabuck deals that happens quite frequently in the magazine business. I heard recently that the publishing company that bought COMPUTE was itself purchased by another larger corporation. I have no idea how it or its publications will be affected, but it just goes to show that even the sharks get eaten by bigger sharks.

As a result of the sale, we can no longer "sell" COMPUTE products, since technically we don't own the rights to them. This affects Gazette in so far as it was a part of COMPUTE magazine from September 1990 until December 1993. What all this means to you readers is that this is a good time to stock your library with back issues of COMPUTE and COMPUTE's Gazette.

We did an inventory and discovered a good number of magazine back issues and disks. The new owner doesn't want them and we can't sell them, technically. We can't afford to give them away either, so we are making them available to subscribers in exchange for our costs. If you want a back issue or two, we will send them to you for the cost of our labor and postage. Our hard-working people in customer service will try to find the issue or issues that you want,package them up, run it through the postage meter, and send it your way. You'll find the costs listed under Back Issues in the Gazette advertising section of this disk.

I'm sorry that we can't provide details about what's on each disk and what's in each issue of the magazine, but that information isn't available. And having someone go through each one and write up a description would take too much time. If you have the magazines but would like the companion disk, this is the time to order.

Looking over the inventory, I see some strange numbers. Hardly any disks or magazines before December 1985 are available, but other months and years are available in large quantities. Take 1988 for example. we have 25 copies of the April magazine and 109 copies of the disk. For May, we have only 11 copies of the magazine, but 1111 copies of the disk. Then in August, the numbers drop to 3 copies of the magazine and 14 copies of the disk.

Sorry, but most of the 1990 magazines are gone, but we still have plenty of disks from that year. Just remember that there were no issues published for July, August, or September of 1990.

If you have any questions regarding certain issues, please contact our customer service department at our Greensboro address, or call (910) 275-9809 between 9 a.m. and 4 p.m.

Of course, these savings don't apply to the past 12 issues of Gazette Disk. As you know, this issue marks our anniversary for this format. The price for 12 issues is still at $29.95, but the bookkeepers say that is a special price for subscriptions only. If you want to buy an individual disk, you have to pay the full price of $9.95 plus shipping and handling.

Gazette, January 1995

EFEDBACK

COMMODORE POKES

Could you provide me with a list of Pokes for the Commodore that disable the Run/Stop-Restore keys?
ANDREW KERRIGAN
MILWAUKEE, WI

Here is a list of Pokes for the 64 and 128 that you may find useful.

	64	128
Disable Run/Stop	808,239	808,100
Enable	808,237	808,110
Disable Restore	792,193	792,125
Enable	792,71	792,64
Disable LIST	775,191	775,139
Enable	775,167	775,81
Disable SAVE	819,246	818,180
Enable	819,245	818,78
Disable keyboard	649,0	2592,0
Enable	649,10	2592,10
No keys repeat	650,64	2594,64
All keys repeat	650,128	2594,255
Enable repeating keys (space, delete, cursor)	650,0	2594,0
Change character color (x = 0-15)	646,x	241,x
Remove line numbers during LIST	22,35	24,37
Enable	22,25	24,27

Just remembr that if you disable the keyboard in immediate mode, you won't be able to type in the Poke to enable it again because the keyboard will be inoperative. Run/Stop-Restore will still work unless you have disabled it too.

The Poke that removes line numbers during a listing gives you a simple word processor. You can type a message with line numbers as if it were a BASIC program, enter the Poke, and then list the program to the printer. Your message will print out without line numbers.

RANDOM NOISE

I have a number of programs that lock up when I try to run them on my 64, but they work well on my friend's machine. Other than having a bad SID chip, my 64 seems to be in good workin order. Any ideas as what might be wrong?
PAT WINTERS
EUGENE, OR

Th defective SID chip could be the culprit. Many machin language programs use th SID chip to generate random numbrs at the start of a game. If the chip is defective, then the program can't get its number and it hangs up.

The idea is that voice 3 can be set to make noise and since noise made by the computer is simply a string of random frequencies, a program can make use those random numbers.

Here is an example of how it is done.

```
10 POKE 54287,255: POKE 54290,128: POKE 54296, 128
20 PRINT PEEK (54299)
30 GOTO 20
```

This program prints random numbers until you hit the Stop key. Notice that all of the numbers are between 0 and 255. That's because they are random bytes. To get a ramdom integer between 0 and 9, use this line.

```
20 PRINT INT(PEEK(54299)*10/256
```

This procedure is seldom used with BASIC programs because BASIC already has a function (RND) for generating random numbers.

READER TO READER HELP
I do a lot of mailings and make labels for repeat addresses. Has anyone developed a ZIP code bar coding program for the 64 or 128?
JIM HOOD
1209 PRINCETON AVE.
SALT LAKE CITY, UT 84105

We have seen labels on user group newslettrs sent to Gazette with bar codes on them, but we have not seen the programs that generate them. Perhaps our readers can supply a source.

ML PRINTOUTS
Is there a routine that I can use on my 128 to send a machine language listing to my printer?
BILL CARWITHEN
COTTAGE GROVE, OR

The monitor program that is built into you 128 lets you disassemble programs with the output appearing on your screen. To send this information to your printer, you simply need to redirect the output.

Try entering the following line in BASIC before you enter the monitor.

```
OPEN4,4: CMD4: MONITOR
```

Now, when you disassemble you machine language code with the D command, the output will be sent to you printer. If you wanted to print the disassembly from $0C00 to $0C36, for example, enter th following line once you are in your monitor.

```
D 0C00 0C36
```

When the printout has finished, exit the monitor with X and then type PRINT#4: CLOSE4 to restore output to your screen.

WHERE'S THE JOYSTICK?

I'd like to write a program that will use a joystick to make selections. Is there a BASIC routine for the 64 that will read a joystick's position?
LARRY SWOFFORD
ORLANDO, FL

The following lines of code will read a joystick plugged into port 1. To read a joystick plugged into port 2, make B=56320 in line 10.

```
10 DIMJ (10): B=56321: FOR A=1TO10: READJ(A): NEXT
20 DATA 1,5,0,7,8,6,0,3,2,4
30 J=J(15-(PEEK(B)AND15))+128-8* (PEEK(B)AND16)
40 PRINTJ: GOTO 30
```

Lines 10 and 20 initialize the routine and should be placed near the beginning of your program. Line 30 reads the joystick position and places the value in variable J. The following table shows the possible values of J and the corresponding directions.

Value of J	Direction
0	center
1	up
2	upper right
3	right
4	lower right
5	down
6	lower left
7	left
8	upper left

If the joystick button is pressed, 128 is added to the value of J. So if the button is pressed while the joystick is pointing down, for example, J will have a value of 133. The value of J in this program is the same value returned by the function JOY on the 128.

Gazette, January 1995

DIVERSIONS: Where in the World is Fred D'Ignazio?

By Fred D'Ignazio

It has recently become very, very hard to catch up with me. Several people have told me they can never reach me on the phone or catch me in my office. My response has been that I have been spending a lot of time online in cyberspace. Now that my Gazette column is created, transmitted, and printed totally in cyberspace format, it makes me wonder if maybe my destiny is to spend less time in the real world, and more time in this strange new electronic world we are creating.

Also, I just read about the new intelligent browsers for the World Wide Web (WWW), and how they will help users navigate the web by automatically sending web robots to search out similar keywords and associations to the ones the user is currently investigating. And how to conserve web space other robots will keep relocating the web's files onto new machines, on a nanosecond by nanosecond basis, in order to cope with the flood of new web traffic.

According to the author these two kinds of robots will make the web "omnipresent and at the same time unlocatable." I like that phrase! I immediately wanted to write an article about how my spending more time in cyberspace is also making me "omnipresent and at the same time unlocatable." I decided to write the article as a mystery and set it in a "Carmen Sandiego universe." The detectives in the story are searching for someone even more elusive than Carmen -- someone called "Fred D'Ignazio" -- and there's no telling where in the world this Fred D'Ignazio might be.

***

It's a scene that could be set anywhere or in any time. There's a small group in a small office huddling around their leader. They are all detectives but they have the wholesome scrubbed look of bird watchers. The leader outlines their mission:

"You've heard of that strange little guy with the name that sounds latino or italiano? He's not a crook, but he's been spotted in Australia, the UK, Northern Saskatchewan, and in a small schoolhouse on the outskirts of Anchorage, Alaska. You could learn a whole lot of geography as you trail this guy through the pampas of Uruguay, the Inca ruins in Macchu Picchu, and the Grand Bazaar in Istanbul. He's been in all those places."

"But where is he now?"

"Where in the world is Fred D'Ignazio?"

The group grows agitated and excited. "Shh!" You say to calm them. "Be careful, he could be anywhere, perhaps right here in this very office."

You give them the scoop: "It's really a more a matter of ersatz folklore and crypto-myth than physical deduction. As my five-year-old tells it, Fred D'Ignazio is like an endangered species in a fragile habitat. Fred D'Ignazios are meant to be seen and not heard ...that is, to be heard and not seen. Perhaps neither seen nor heard. Anyway, grab your wireless handheld communicating computers, your notebooks, and come with me."

You tip-toe in single-file around the office. Suddenly a faint sound comes from the next room.

"Tippety-tap! Tappety-tip! Tap ... tap ... tap ... tip."

"We're in luck!" you exclaim. "It's the telltale sound of a Fred D'Ignazio! There must be one nearby."

You take a risk and peek around the corner, but all you see is a little black computer disk. Its label reads "Gazette Disk."

Now what's that supposed to mean? ("Write it down," you whisper. "It could be a clue. Maybe the object is the remains of a Fred D'Ignazio campsite. It could even be one of the creature's 'droppings.'")

"Tip ... tap ... tip ... tappety-tap!" (The sound is more distant now but is coming from another room.)

In a whisper you address your fellow detectives: "Do you hear it? Good. We're getting close to a Fred D'Ignazio. It says in this guidebook that Fred D'Ignazios can only be seen by the shadow they cast, only spotted by their reflection in the mirror, only peeked at briefly around tight corners. Don't ever try to look one in the eye. Even worse, don't expect to actually meet one! But you can be sure that a Fred D'Ignazio is nearby from the sound their small fingers make as they scamper across the plastic keyboard savanna.)

REACH OUT AND TRY TO TOUCH ONE!
(Spurred on by your good fortune in locating an alleged Fred D'Ignazio, your group takes a special risk: You know you can't confront it physically, so you try to call it up on the phone.)

"Brrrinng! Brrinng! Brrinng! Brrinng!"

In a guarded, detective-like whisper, you tell your group: "Please write that down: four rings. The phone in the den of the Fred D'Ignazio has made four rings. This may be signifi ....

"Hello! You have just reached the studio of Classrooms without Walls interactive TV."

In your excitement you throw caution to the wind and you shout: "He speaks! It's really him. Such good fortune, fellow detectives! This is more rare than coming face to face with a Himalayan snow leopard! We're about to ...."

"I'm sorry, but I'm not here right now."

(Collective frown.)

"I'm off somewhere training in space ... or cyberspace ... and I can't come to the phone right now. But if you'll leave a message, I'll get back to you when I can ... If I can. ... Beeeep!"

(You hang up hurriedly, sweat dripping from your brow, the strain and disappointment of this near-encounter wrenching and distorting your rugged features.)

"Oh, darn! What a shame!" you swear. "I thought we had him for at least a quick image grabber. But he's so elusive, so hermitic, so solitaire. You can't meet him. You can't see him. Instead you have to be content with that wretched robotic voice. But wait!" (A look of stunned amazement comes on your face.) "I think I've got it. I think I know how we can catch that perverse creature!"

THE BREAKTHROUGH!
You share your brainstorm: "Cyberspace! That's where we can find the Fred D'Ignazio, guys, in cyberspace."

Have you heard of it? It once was a lonely country with just a few reclusive geek and nerd inhabitants tucked away in little cyber-dens and cyber-burrows, with no one else for miles around. It was a barren, empty land with terrible gushing torrents of data, flash-floods really, that made you keep a nervous lookout for high ground.

Sure there were others around too, but they weren't the sort of people you'd like to meet: drug pushers, soldiers, and spies. Double-spies, too. A few greedy businessmen willing to go anywhere to make a buck, and, yes, quite a few bankers.

But now cyberspace is becoming respectable. It's catching on. These days you see politicians out there, scientists, and librarians. You see teachers leading around little bands of curious children. You see farmers, electricians, and homemakers. And they're building roads, post offices, and setting up shops, stores, and even malls. In fact there's so much settling and development going on that there's talk of building a superhighway, just to accommodate all the traffic that's expected there shortly.

You notice that looks of disbelief are appearing on the bland faces of your little band: "You're shaking your head in disbelief, detectives, like who in their right mind would ever want to go there?

"Well you're right! But don't let your personal preferences stand in the way of cold, professional scrutiny. Think about it: You can't touch cyberspace, you can't smell it, and you certainly can't taste it. But it's starting to change. It's already more than a few neutral bits and bytes out of the maw of some uncaring computer. There are those who are trying to draw cyberspace, to paint it, give it texture, color, and dimensionality. Others are bringing in sounds -- voices, music, habitat noises smuggled in from the real world. Yes! It's true! Even now as we sit here gabbing, cyberspace is buzzing with the sound of crickets, reverberating with the roar of lions, pulsating with the sound of pounding surf, and quivering with the whine of a 747. It's rocking to the sound of Pearl Jam. It's waltzing to Schubert and Shostakovitz.

"Sshh! Listen!" You all grow silent and gaze nervously in all directions. "Do you hear the faint sound of rain drops falling somewhere far away? It's a gentle sound, really. Almost inaudible."

"Drip ... drip .... drip."

"That's cyberspace. And that's where we'll find our Fred D'Ignazio."

It's clever of him how he's tricked us, you know. He puts up all these decoys: He rents an office. He circulates likenesses of himself on the covers of books, magazines, and propaganda flyers. He hands out business cards with a false address and a fictitious phone number.

Decoys all! Ruses and red herrings. All of them designed to lead us to the same dead end: empty space. The silence of no one home. The confusion and bewilderment that makes us wonder if Fred D'Ignazios are even real. We begin thinking maybe they're more like unicorns, elves, and ogres: the product of fancy or the result of indigestion.

But that's just what he wants us to think. While we're out here visiting an empty office and talking to a robot voice on a phone, the real Fred D'Ignazio is making tracks deeper and deeper into cyberspace. And if we stay here we'll never catch up with him. He probably only comes back on weekends now, maybe to visit his family or brush his teeth, then, whooosh! He's just another photon on a fiber-optic cable, an electron spinning dizzily to the South Pole, off on a whirl around the solar system, or holed up in some dim, murky carrel in some street corner library in Djakarta.

"If we're ever going to bag this fellow, and bag him we will, we've got to go there after him. And the sooner the better!"

The light fades. The little band of sleuths treks off into the distance still huddled around their fearless leader. Suddenly a bus pulls up. On its side is painted a tile-like mosaic pattern. Its name is "Internet Browser." Possibly some obscure bus company. The little group boards the bus, and it streaks away swiftly, giving the illusion of vanishing.

All is silent. Then from far away across the world, the wind begins to blow. And with the wind comes the gentle tapping sound of the

Fred's fingers, "tip ... tap ... tip ... tap," as they bound across the keyboard savanna.)

Gazette, January 1995

BEGINNER BASIC: Evolution Of A Program

By Larry Cotton

Happy New Year! I hope this year will bring you success, good health, and happiness.

This month we'll look at a program which can be used if you ever need to take a survey. It can be tailored to most any situation, and all responses can be neatly tucked away in a disk file, or you can print them on any standard Commodore-specific or properly interfaced printer.

Last month we covered disk loading and saving routines, so we won't go into much detail about them this time. I took some notes as I was writing this program and I thought you might like to see how I approached it. Of course your style may be entirely different, but maybe by studying the evolutionary process, you can avoid a pitfall or two.

To begin with, I wrote two lines -- arbitrarily numbered 100 and 110 -- which used PRINT to ask a question, and INPUT to get the user's answer. Then, to allow any type or any number of questions to be asked, I put a few in DATA statements and READ them.

Originally, I thought I'd need to include the type of question, such as open-ended, true or false, multiple choice, and so on. I later decided that that wasn't necessary, but you may think otherwise. If you do add a question type, just put it in along with the data and read it while the questions are being read.

We need a constant which represents the total number of questions. This constant will be used repeatedly to loop through the questions and answers while surveying, printing, or during disk activity. I chose to let the letter Q (fo Question) represent that constant.

If you plan to ask more than ten questions, a DIM statement is necessary early in the program. Here are two of my program lines:

```
50 Q=4
60 DIMQ$(Q),A$(Q),TEMP$(Q)
```

Since I've written only four sample questions, DIM isn't necessary, but it does allow you to use more questions. Next, we clear the screen and give some instructions to the user. Keep the instructions as clear and concise as possible. Offer the opportunity to load a previous survey from disk for studying onscreen or for printing it out. (I have a included a sample on the flip side with this program.)

One of the limitations in my program is that the answers should be one-liners, so I inform the user of that right up front. You may also want to warn the user not to use commas, quotation marks, dollar signs, and so on.

To begin the actual survey, put the program in a FOR-NEXT loop which reads the questions from the data statements, then asks for answers. Inside the FOR-NEXT loop, you'll need READ, PRINT and INPUT. You should probably use string-variable arrays for everything. If you need numeric answers, you can always use VAL to extract the numbers from the strings.

Recall that an array is like a row of lettered pigeonholes with the same basic variable name, such as Q$(X) or A$(X), where X is the index to the array.

I've used a trick to avoid the question mark that's normally associated with an INPUT statement: it's located at the subroutine in line 540. Here it is.

```
540 OPEN1,0: INPUT#1,A$(X): PRINT: CLOSE1: RETURN
```

If you'll take a look at the program itself called Surveytaker on the flip side of this disk, you'll see that the screen format of the questions and answers is neater and more user-friendly without the question mark.

To keep the program short, I didn't include much error-trapping. You may want to add some, depending on who might be taking the survey.

Multiple-choice questions posed some difficulty. (See the example question in line 570.) You may want to get creative and figure out a way the possible answers can be presented in a columnar format.

At this point in the programming procedure, enter a few data lines and check for syntax or other errors. The program should be able to loop through the questions, pausing for the user's answers.

In immediate mode, you can check whether your arrays contain the correct data by entering PRINT A$(1) and pressing Return. You should see the first answer you entered.

Opinion surveys usually don't have right or wrong answers, so your DATA statements probably need to contain only questions. However, you can always make the program more elaborate by reading additional data, such as a possible answer, and comparing the user's answer with that.

You may now want to pretty up the program a bit by selecting screen and text colors, when to clear it, where the questions are placed, and so on. Surveytaker is strictly bare-bones, so feel free to experiment.

After the survey answers are gathered, you should also give the surveyee the opportunity to change his or her mind. This can be accomplished (after some instructions) with another FOR-NEXT loop where the user sees the original questions and answers again. By simply pressing Return, the user can quickly scan through the answers he or she doesn't want to change.

You need to give the user the ability to end the scanning. I did this by looking for a press of the back-arrow key.

This gets a little tricky. Normally if the user presses Return at an input statement, the old answer is discarded and the array pigeonhole becomes empty. So I assigned a temporary variable to hold the old answer while INPUT was being used. You also need to look for a press of the back-arrow key:

```
240 TEMP$(X)=A$(X): GOSUB540
250 IFA$(X)=""THENA$(X)=TEMP$(X)
260 IFA$(X)="(back arrow)" THENA$(X)=TEMP$(X): GOTO280
```

After the survey is complete, we must present a little menu which offers the surveyee the opportunity to print the survey, save it to disk and quit, or just quit without printing or saving. So the last three subroutines in my program are printing, disk loading, and saving.

The print subroutine uses CMD to re-direct the output from what would normally be the screen, to the printer. Be sure to use PRINT# to send a blank line to the printer before closing the file.

As mentioned, the disk saving and loading subroutines are lifted almost directly from last month's program, the Christmas gift-lister.

The last lines in my program (550 on) contain four questions which illustrate open-end, yes-no, multiple-choice, and true-false questions. Be sure to change the variable Q in line 50 to match the number of questions.

After writing and checking the fortieth iteration of the program, I printed it out and carefully scanned it for places where I could use subroutines, where lines could be crunched, and so on. I then renumbered it using a renumbering utility.

Gazette, January 1995

MACHINE LANGUAGE: Flags and Branches

By Jim Butterfield

Here's how you make a decision in a program. First, you take some kind of action to affect a flag. Then you use a Branch instruction to test that flag.
It's a two-step operation: one instruction influences the flag, and then a following instruction, a branch, checks it. If the flag condition matches the specified state, the branch is taken and the program hops to a new location. Otherwise, the branch is ignored and the program continues with the next instruction in sequence.

This time around, I'll do a detailed outline of how flags and branches work together. Program Scroll64, found on the flip side of this disk, will show one aspect of this.

There are seven flags built into the microprocessor, but only four of them can be directly tested with branch instructions. Flags are like toggle switches: they are either off or on, set or clear.

The seven flags are packed together in an area called the "Status Register"; most machine language monitor programs will display the whole group as a hexadecimal value under the heading SR. Sometimes during testing it's useful to know the condition of these flags. If so, convert the hex number to binary. The eight bits, from high to low, are identified as NV-BDIZC. Here's what each letter means.

N	Negative flag (testable)
V	oVerflow flag (testable)
-	not used
B	Break flag, not testable
D	Decimal mode flag, not testable
I	Interrupt Disable, not testable
Z	Zero flag (testable)
C	Carry flag (testable)

So the flags that may be directly tested are N, V, Z, and C. The formal names can be misleading: for example, the Z flag sometimes tests for a zero condition, but at other times it tests for two values being equal to each other. Because of this, I often call flags by their letter rather than by their name.

Let's take in a little perspective: the N and Z flags are very busy indeed and change frequently. The C flag is modified fairly often. The V flag changes only occasionally.

FLAG WAVERS

What type of things influence the state of flags? There are four general categories.

REGISTER-CHANGERS

Any instruction which modifies the contents of a register -- loads, transfers, increments and decrements -- will certainly affect the N and Z flags. This also includes the PLA (pull A) instruction, which brings a value back into A from the stack. Also, the two instructions INC and DEC, which modify the contents of memory rather than a register, also influence N and Z. That's an exception, since Store instructions (STA, STX, STY) do not influence ANY flags.

Following a register-changing instruction, the Z flag indicates whether the result of the activity was zero; and the N flag indicates the state of the high bit of the result. (Bit 7, the high bit, is often called the "sign" bit; but it's only a sign if you choose to use it that way).

ARITHMETIC
ADC and SBC, the add and subtract instructions, influence all four flags. The Shift and Rotate instructions (ASL, ROL, LSR, ROR) correspond to multiplying or dividing by two; they affect flags N, Z, and C, but not the V flag.

Following an arithmetic instruction, the Z and N flags again indicate if the result is zero, and the state of the high bit. The C flag is a Carry as its name suggests; it indicates that a bit has "spilled out" of the operation.

The V flag signals Signed Overflow. If we are doing arithmetic on signed numbers, the V flag will be set if the sign (the high bit) has been invalidated because the number has gone beyond its allowable range.

When you do multibyte arithmetic, these flags may be meaningful only after you have handled the last byte of the sequence. This is especially true of the sign and overflow bits (N and V), since the sign is found only in the highest byte of the number.

COMPARISON
Any of the three Compare instructions (CMP, CPX, CPY, compare A, X, or Y with memory) will affect flags N, Z, and C. Although some texts describe comparison as "a type of subtraction", the V flag is never changed.

Following a comparison, the flags take on somewhat different interpretations. The Z flag tells if the register value is equal to that of memory. The C flag will be set if the register is greater than or equal to the memory value; that's in terms of unsigned values.

The N flag is curious. Although it is influenced in a consistent way, it is seldom useful. The simplest way I can describe what the N flag tells you is that it answers the followin question. Allowing for numeric rollover, would it be quicker for the memory value to count up or down to reach the register value?

SPECIAL
We can set or clear some flags directly. SEC sets the C flag, CLC clears it; CLV clears the V flag. PLP restores all flags from where they were stored earlier on the stack. The curious BIT instruction copies the high bit from the memory address directly to the N flag, and the next bit (bit 6) to the V flag; then it does a bit-masking test between memory and the A register. If no bits match, the Z flag is set. Most curious of all, the V flag can be set by hardware! A pin on the processor chip allows this; the feature is not used in Commodore's eight-bit computers, but is used in the disk drives.

BRANCH INSTRUCTIONS
The Z flag might sometimes be called the "equals" flag, and the names of the branch instructions reflects this. BEQ (Branch Equal) will take the branch if the Z flag is set; BNE (Branch Not Equal) branches if the flag is not set.

I often call the N flag the "high-bit" flag, since that term is more descriptive. The names of the branch instructions track the N-for-negative idea, however. BMI (Branch Minus) will take the branch if the N flag is set; BPL (Branch Plus) branches if it's not set.

The C flag often means "greater than or equal" after a comparison. The branches are simply called BCS (Branch Carry Set) and BCC (Branch Carry Clear), which describe pretty well what they do.

The V flag signals an overflow condition only if you're doing arithmetic on signed numbers. For unsigned numbers, the C flag would be the one to use in testing for overflow. Perhaps V is more often used to test a bit-6 flag in combination with the BIT instruction. In any case, BVS (Branch Overflow Set) and BVC (Branch Overflow Clear) do the appropriate testing.

BEGINNING TESTING
Want to test if A is equal to 5? Simple: CMP #$05 .. BEQ .... and you'll branch if a 5 is found.

Want to test if A is greater or equal to #$30 (hex 30, decimal 48, is the ASCII code for character 0). Again, no problem: CMP #$30 .. BCS .. and off you go, keeping in mind that BCS means "greater or equal" in this case.

Taking another step, to test if A is a numeric digit character in the range 0 to 9, hex 30 to 39, we would code:

```
   CMP #$30
   BCC NOPE
   CMP #$3A
   BCS NOPE
```

The program will branch to NOPE, wherever that may be, if the value in A is too low (BCC) or too high (BCS). Note that I had to add one to value $39 to make $3A so the numbers work right (BCS branches on equals, too).

TRICKS AND PITFALLS

An extra instruction might change the flag you were planning to test, but be careful! On the other hand, some instructions do not change some flags. The Store instructions don't change any flags, for example. Examine this code.

```
    LDA #$00
    LDX #$01
    STA $1234
    BEQ ...
```

A beginner might think, "I've just stored a 0 value, so the Z flag must be set." ..

Wrong, wrong. The last instruction to affect the Z flag was the LDX, and since we know the register value ended up as 1, we know the Z flag will be clear. That STA instruction did not affect any flags.

With care, you can sometimes use this trick of separating the test and branch instructions from each other. Let's take an example from a program we have seen before.

SLOW SCROLL PROGRAM

This 64-only program has a number of features, such as double-buffering, but we'll concentrate on one tiny piece of code. That's the place where we wait for the raster (the process that paints the display screen) to go back to the top of the screen. This will allow us to do screen work without jitter.

We can read the raster position by examining the video chip. The byte at location $D012, decimal 53266, zooms upward from 0 as the screen is painted. When the value reaches 255, it will "roll over" to 0, which we will see as a drop in the reading. A high bit may be found at $D011; we'll talk about that in a moment.

The raster continues to paint the screen, and the value seen in $D012 continues to increase. When the raster reaches the bottom of the screen, it flies back to the top; the $D012 value drops back to 0.

Our task is to catch that moment of screen retrace; that's the best time to do jitter-free screen work. We must look for the contents of the raster register to drop in value. It's not enough for us just to test for 0, we want to spot the transition from a higher value to a lower one.

When we find that transition, we should perform another test. If the high bit (sign bit) of $D012 is on, the screen is not in retrace yet. In that case, we go back and wait some more.

Try following this logic. Suppose we have stored the previous value from $D012 into memory. Now we have a new value in A, and we want to know if it is lower than the previous value. Easy. CMP will compare the value with memory, following which BCC would branch if the register value was lower; or BCS would branch if the value were the same or equal.

Here's the tricky part. No matter what we find, we should replace the old previous value with the new one. That way, the next time we do the test we'll be working with the real previous value. At first thought, this updating job seems complex, but examine the following code:

```
WAIT:
  LDA $D012  ; get the whole register
  CMP $03D0  ; compare to prev value
  STA $03D0  ; update prev value
  BCS WAIT   ; not yet, so loop.
  BIT $D011  ; is high bit set?
  BMI WAIT   ; if yes, go back.
  ...
```

Now we may do our screen work.

The program compares (CMP), then stores (STA) to update the "previous value"; and only then does it act on the result of the comparison. If the C flag is set (register greater or equal), we'll loop back and keep waiting. When the raster value takes a dive, we'll fall out of the inner loop.

At this time, we check the high bit of $D011. I'm using the BIT instruction, but a LDA would do just as well. If the high bit is on, the N flag will be set, and BMI sends us back to wait some more.

Note that whether or not we loop, the previous raster register reading is always logged.

SUMMARY
The names of flags can sometimes be misleading. But it doesn't take long to learn how to use them with the branch instructions and write truly decisive programs.

Gazette, January 1995

PROGRAMMER'S PAGE: Security

By David Pankhurst

As I looked at my car's broken window, and the wires from which speakers were once connected, I thought how appropriate that this month we would be dealing with security. Actually, that wasn't the first thought that came to my mind, but it was the most appropriate to mention here.

More is involved in security than just a good alarm or dogs on patrol; security also comes from knowing you have a plan if something does happen. We'll look at both of these issues for the Commodore user this month.

WHAT WILL YOU DO AFTERWARDS?

The unthinkable happens. You're in a fire. Fortunately, everyone is safe, and life slowly returns to normal. Or does it?

Most likely your insurance covers your furniture, expenses, and such, but what about your computer setup? The Commodore itself is replacable, but if you're like me, you've added switches and such that make it unique and consequently more difficult to replace. You may also have your favorite software or cartridge set up just right. All of which is now a melted lump of plastic lying in the middle of what used to be a home.

A tad melodramatic, I know, but melodrama serves to bring home an important point. Are you prepared for accidents with your Commodore? Although fire isn't likely, it gives you a worst-case senario to consider. There seems to be no such thing as slight damage when it involves computers. Even one disk chewed by the dog can be traumatic, if it's irreplacable.

This should highlight a preventative measure everyone can and should take, that of off-site backups. Keeping a copy of your disks (or the originals) elsewhere is good insurance. It needn't be special storage, such as a bank's safety deposit box. One easy solution is to store copies of all your software at a friend's place (and in all fairness, invite him to store his disks at yours). For added security, and to protect the disks from curious eyes or disk drives, invest in a small locking box to store them in.

What do you protect? Everything. Keep copies of all your disks. And this applies not only to purchased programs, but data and programming efforts of your own, including work in progress. Every byte of information involves time to replace or recreate. Save that time by keeping backups of everything. After all, disks are cheap, but re-creating your masterpiece isn't.

LOCK UP YOUR FILES

Now that you're terrified to light a match, let's move on to day-to-day security. Accidents happen, such as entering the format command, or erasing too many files, or the wrong ones. It's possible to lock files, rendering them safe from this accidental erasing. The program "File Un/Lock", when run, asks for a file name, and locks it on disk. If the file is already locked, running the program unlocks it, allowing you to delete or rename it.

AN ANCIENT DISK PROTECTION SECRET

The above program provides protection for individual files easily, but gets cumbersome when protecting more than a few, or a diskful. A method for protecting the whole disks' contents comes from a trick from prehistoric Commodore days.

The 2040 disk drive, an ancestor of the 1541, was read compatible, but not write compatible. It identified its disks with a special byte in the first block of the directory. The program Disk Lock changes this byte on the 1541 disks, fooling your disk drive into treating them like the old 2040 disks, so they can be read from, but not written to.

To use Disk Lock, load it and then run the program after placing the disk to protect in the disk drive. The only thing that can then affect the disk (aside from stray magnetism) is a complete formatting (N0:DISK,ID) - even the shorter form of the disk NEW command (N0:DISK) won't work. To unlock the disk, simply re-run the program.

SAVING YOUR SAVES

Years ago, there was great discussion about the infamous 'save @' bug. Under certain conditions, saving a file with the @ sign in front of the file name wouldn't work properly. Normally the command worked by saving the new file, then erasing the old one; but at seemingly random intervals, it would garble the resulting file. This was a pity, since it was such a handy command.

Vince Tagle of California makes things safe and secure again with his program, Scratch/Save. It's a relocatable machine language routine you load and run to place in memory. There it waits until you do a save, whereupon it saves the file, after first erasing any duplicate (older) file on disk. If there is no old copy on disk, just the save is performed. Best of all, the routine is transparent; as long as you don't disable it (such as with Run/Stop Restore) it will do its job with every save.

VINCE'S VERIFY

Vince also solved another security issue, with his program Save/Verify. Although the 1541 does a low-level verify when writing to disk, sometimes the only warning of a problem is the flashing drive light. I've missed it, only to find the copy didn't go as planned. Vince's program waits in the background, and after a save, it does a BASIC VERIFY. The result of this command is displayed onscreen, giving you added protection and peace of mind.
Like his other program, this one is relocatable, and is run to install itself in memory, where it waits for the SAVE command. Note however,

since both of his programs use the SAVE vector, it's best not to run them on the computer simultaneously.

CHECK OUT THIS FILE!
My biggest worry with backups is that somehow they won't be exactly the same file. On occasion I've encountered files that don't match the original, usually when I need the backup desperately. To avoid reacurring trauma, I wrote Checksum.

This program reads a disk file and does a checksum on it, yielding a number from 0 to 65535. This checksum will be the same on identical files, otherwise it will be different, letting you check if files are identical. One caution: since the file to be checked is copied into memory for speed, the maximum file size that can be checked by this program is 187 blocks long.

Checksum, which is on the flip side of this disk, loads and runs from the Gazette Menu. Although it is used to check different versions of a program, you can try it out by entering "CHECKSUM" as a filename for testing purposes.

A HIDDEN HAVEN FROM HACKERS
Sometimes someone is out for michief. There are people who hack just to cause problems -- look at the number of computer viruses springing up. Although you can't easily prevent a serious hacker from looking through your files, you can confuse the casual snoop with Hide Dir. Kirk Fontenot of Lousiana suggested this program that hides relevant files on long disk listings.

By linking the start of the disk directory to the end, all of the middle files effectively disappear. The way the 1541 directory is organized, up to eight of the first filenames and up to eight of the last filenames are displayed, but every filename in between is hidden from view. (And if you can't see it, it's hard to load it, rename it, and so on.)

To make the filenames available again, rerun the program. A serious hacker can still get at your files, but a casual look over the directory listing won't reveal anything amiss.

Although we can't be perfectly secure in this world, with some advance planning, we needn't be caught unprepared and at a disadvantage. Take time to analyze how you can protect your computer system, and do something about it. In the modified words of Smokey the Bear, "Only you can prevent loss of data"!

Since Gazette Disk is write protected and most of these files must write to disk, we recommend that you copy them to a work disk for running and testing. The files included on the flip side are as follows.

"FILE UN/LOCK"
"DISK LOCK"

"SCRATCH/SAVE"
"SAVE/VERIFY"
"CHECKSUM"
"HIDE DIR"

Gazette, January 1995

GEOS

By Steve Vander Ark

Every so often, as I plan this column, I poke around in my overflowing disk drawer and start pulling out disks I haven't seen in years. It's kind of fun to do that, popping them into my 1571 and paging through the disk directories. I have spent a lot of time on QLink and GEnie over the years, downloading files left and right, and I would often sock them away on one of these disks that I put in a drawer, planning to try out the files soon.

Now, three or four years later, I find these files every so often, still unconverted into GEOS format, just waiting for me. I never know what I'll find when I convert them over and double-click, since the file descriptions are long gone. Sometimes I find myself entertained; sometimes I laugh and toss the file into the trash; and other times my mouth drops open and I hit my head in the classic "Duh" manuever and say, "Why didn't I try this file years ago?"

So this month I figured I'd just pull out some of these lost treasures and share them with you. I'll even include them on the disk in converted format; you won't be able to load them from the menu, you'll need to copy them over to a GEOS disk and use Convert or geoPack to turn them into GEOS files.

MOUSER
by Douglas Adams
Do you use geoPaint or geoPublish? Then you need this little desk accessory. When you select if from the "geos" menu, you'll see a dialog box that lets you temporarily change the speed and acceleration of your mouse. Adjusting the mouse speed downwards can make it much easier to do fine mouse work in those graphics programs.

Mouser is a 40-column desk accessory only, which is ok when you consider that many of the graphics programs work mostly in that format. When you leave your program, the mouse setting reverts to the setting stored in your preferences file. If you don't have a preferences file on your disk, the adjusted settings will remain in effect. Look for Mouser on this month's disk under the filename MOUSER.CVT; that CVT extension tells you that you need to convert the file into GEOS format. I'll also include the documentation in geoWrite format as MOUSERDOCS.CVT.

GeoCHORD
by Douglas Adams
I don't suppose many of you will even begin to understand what this program is up to. I'm not saying you're out to lunch here, just that this program is designed for a very select group of GEOS users. I happen to be one of those, so I get pretty excited about geoCHORD. As a result, I'm tossing it your way; if nothing else, it will give you

an idea of what unusual things GEOS can do.

GeoCHORD is designed for musicians among who dig into music theory a little bit. As a guitarist for many years, I've found the structure and relationships of notes in chords to be fascinating. GeoCHORD lets me experiment with those structures by taking any set of notes I enter (by clicking on a keyboard at the bottom of the screen) and listing the chords that use those notes.

Unfortunately, the program doesn't actually play the chords, which is a shame. It would make it a lot more interesting to actually hear the relationships between the various chords which share some notes. But since I can play those chords myself on the guitar (and since my playing sounds even better than a three-voice SID chord, in my humble opinion) that doesn't really wreck things for me. Of course, if you have no idea what a G7sus4 chord is and don't care, you won't get a whole lot out of geoCHORD.

If you think you just might find it interesting, though, you can find it on this month's disk as CHORD.CVT and CHORDDOCS.CVT.

TUNERv2
by Randy L. Smith
I'm not through with music yet, folks. Don't panic, this program isn't as esoteric as geoCHORD. It is actually very useful indeed. The fact that it's a desk accessory makes it all the more easy to use. What it does is play the notes you need to properly tune your guitar.

That's it. But if you play the guitar, sooner or later you have to tune it. And when you tune your guitar, knowing the correct notes to which to set the strings is pretty important. Tuner will give you those notes, and you don't even have to leave geoWrite to get them, which makes Tuner pretty easy to use.

Ok, so if you're hanging around strumming your guitar you probably aren't writing at the same time, but the point remains. And you'd be surprised how often I don't kick back for a few minutes during a marathon writing session and play through part of "Desperado" with my guitar, just to get my creaky brain back on track.

Tuner includes the notes for both standard tuning and open chord tuning. The notes hold indefinitely (even when you close the desk accessory) or until you click on the little button at the top of the picture of the guitar on the screen. Tuner is on this disk with the file name TUNERV2.CVT.

DICE
by Michael E. Landon
Ok, all you non-musical readers can come back into the room now. This next little program is one that should appeal to an even wider range of users. It's a desk accessory that will roll dice for you, dice of many shapes and sizes. All the typical role-playing game dice are here, from 4-sided ones all the way up to 30-sided. Percetile dice are

also included. The program even presents you with a graphics representation of what you rolled.

Just like Tuner, I suppose the argument could be made that if you're busy using, say, geoWrite, you probably don't need to roll dice for anything. But then again, mapping a game as you go on the computer screen using geoPaint (and geoSTAMP, maybe, for predefined images...) is a pretty exciting idea, and then having Dice around as a desk accessory could come in very handy. I've always dreamed of a full-service game master utility for GEOS anyway, so maybe this is a good first step.

Dice is on the disk with the file name DICE.CVT.

It looks like I'm to the end of my allotted words. That went pretty fast! I have several more programs that I found on the same disk as those I just talked about, and there are plenty more disks in that drawer of mine. Maybe I'd better do this again real soon. In the meantime, let me know how you like having the files included on the disk. Would you rather have them already converted into GEOS format for you?

Send me mail in care of the magazine or send e-mail. You can reach me on GEnie (S.VANDERARK) or via the Internet at S.VANDERARK@GENIE.GEIS.COM.

Gazette, January 1995

PD PICKS: Life, Disk Checker, Disk Display, and Whatzit

By Steve Vander Ark

Well, it's finally happened. I have found a PD program that keeps me even more engrossed than SuperRockfall.

Here's the real shocker: Nothing blows up, you don't get to shoot missles at anything, there isn't even any sound effects or music. Slamming back a few cans of Mountain Dew won't improve the score, since there isn't any score, really. This program is nothing like any of the other programs I've talked about in this column.

Now I have to be honest here. Many of you might try this program and wonder what in the world I can be thinking. Maybe you'll never try it again. I almost didn't. The first time I tried it, I sat there and stared and wondered why anyone would waste time on such a thing. My wife had the same reaction.

I came into contact with this program first on my Newton, a hand held computer which I carry with me everywhere I go. I had placed a version on my Newton and left it there after trying it once or twice. Soon after that I tried the Commodore version that you'll find on the flip side this disk.
At that point, I was completely unimpressed. But when you carry a computer with you everywhere you go, eventually you'll get reach a situation where you're so bored you'll try anything you have loaded on it. That happened to me at a conference a few weeks ago. I had nothing else very exciting loaded onto my Newton, so I tried this program. This time I actually read the instructions and tried a few of the things it suggested. All of a sudden, I was hooked.

When my wife looked over my shoulder the next day to see what I was doing, she just shook her head. What a dumb program, she informed me. So I let her try it. I told her how to create a glider. I let her experiment with a few interesting arrangements of those simple little dots. Wouldn't you know it, she became hooked too.

This program is called Life, and maybe some of you have already seen it. There isn't much to it, just a blank screen with a way to place dots wherever you like. When you start up the thing, some mathematical algorithm takes over, and the dots you placed take on a life of their own.

They multiply, they die out, they just sit there, and all the while the patterns change and convolute and rearrange themselves until either the entire colony dies out or one of six or so permanent shapes appears and the colony becomes perpetual. On my Newton, you place the dots with the stylus, since the Newton is a pen-operated computer. The field is pretty small, though. On the Commodore, Life fills the entire 40-column screen, and the patterns and permutations are fascinating indeed.

There really is a strict logic to why things happen on the screen, of course. A cell with three adjacent dots creates a new dot. A dot with less than two neighboring dots dies. All the rest just sit there. What makes Life so addicting is the fact that every dot pattern you start with will create its own unique sequence of dot patterns, its own structures of life and death. Some flourish in a few rounds, then die off. Others blossom in intricate detail for several hundred rounds.

There are some structures of dots that have names--the glider, the spaceship, or pi, to name a few--which behave in particularly interesting ways. But any pattern is exciting to watch.

The Commodore version of Life suffers a little in the way its draw mode works. The joystick is klutzy and turning dots on and off is a bit confusing. But what this version lacks in drawing power it makes up for in speed. The beauty of the patterns comes alive on the 64's screen as the generations of dots slide by.

Ok, maybe this isn't the game you've been dreaming of, but I've given you plenty of shoot 'em ups in the last year or so. Life is something different and, I think, something very special.

DISK CHECKER
DISK DISPLAY
Here we go, proof positive that I write about some things besides games. These two programs are utilities, hacker-style ones at that. Between the two of them you, can get a good solid handle on what's going on with just about any disk.

Now you can get this down and dirty with a good disk editor, of course, but if you're like me, using a disk editor can be pretty confusing at times. It's hard to feel like you really understand what the editor is telling you, since all you get is the raw data. The advantage, of course, is that the editor lets you also go in and change things, so if you do figure out that dizzying array of numbers on the screen, you can actually affect your disk's information.

Disk Checker and Disk Display don't give you editing capabilities. Instead, they each take the raw data from your disk and show you what you really need to know from it. They do the interpreting for you.

Disk Display, for example, will show you not only what information resides at a particular track and sector, but also a list of the linked disk locations that make up a file and a visual representation of the disk's BAM. Even a novice hacker like myeslf can understand the displays.

Disk Checker goes one step further and actually tests things out for you. The whole process can take a while as the program tests start links, compares the file length as given by the directory to the actual length, checks for overlapping files, and so on. In the end you, will be given a list of files which are problematic, those which have possibly become damaged in some way.

Disk Checker shows you what it's up to all along the way, which is interesting to see, but never really changes anything. That's a good thing, since some disks might have files which don't follow the rules and therefore show up as damaged when they're not.

Both of these programs run in 64 mode and work, as far as I can tell, only on 1541, single-sided disks in drive 8.

WHATZIT
I just have to throw a game in here. This little scrambled word puzzle isn't flashy or anything, although the sound effects are pleasant enough. You use a joystick to zip around and choose letters in order to unscramble a word in the category of your choice. Your score is based on how fast you can put the word together.

I like games like this. I have played it quite a few times now, and my only complaint is that I'm running out of words. I listed the program and discovered that all the words are to be found in data statements at the end, so maybe I'll try to add a few somewhere along the line and turn the game loose on my third graders. Hey, maybe I could even put their spelling words in there!

Anyhow, I think you'll like WHATZIT. I've found that it is a great way to take a break from Life. Pun intended.

Gazette, January 1995

The Numbers Game

Take A Look At Line Numbers. They Can Make BASIC Programming Easier

By Don Radler

Unlike other high-level computer languages such as FORTH, Logo, and C, BASIC starts every line with a line number. And anyone who has ever written a program in BASIC, no matter how simple or how complex, tends to take those ubiquitous line numbers completely for granted.

This is one of those cases in which the best place to hide something is right out in plain sight; familiarity seems to breed invisibility, if not contempt.

The Commodore User's Guide says little about line numbers, declaring that they tell the machine in what order to work with each statement, and that they serve as a reference point whenever a program needs to go to a particular line. (For example, GOTO and GOSUB are followed by a line number.) The User's Guide adds that it is good programming practice to number lines in increments of ten, leaving room to insert additional lines should they prove to be needed.

The Programmer's Reference Guide also recommends increments of ten, but then goes on to suggest that once a program is completed, all line numbers should be made as small as possible (1, 2, 3, and so on), to conserve memory. It is true that a three-digit number such as 100 uses three bytes of memory, while a one-digit number uses only one byte. But in the very short programs that most of us write most of the time, memory conservation means little. And in a program of several hundred lines or more, starting with 1 and incrementing by one saves relatively little memory. In any event, that's all your two official Commodore reference works have to say about line numbers.

Riffling through an old issue of COMPUTE's Gazette, you'll see that the type-in programs in the back, after the copyright notice in line 5, begin with 10 and go up by tens. This is the style that Larry Cotton used in his BEGINNER BASIC column. Jim Butterworth, when he writes a BASIC loader in his MACHINE LANGUAGE column, starts with 100 and uses increments of ten. David Pankhurst's PROGRAMMER'S PAGE uses both styles, probably because the material he gets from readers is submitted in both styles. And both styles work, of course.

If you do any programming, the odds are that you use one of these two styles. You likely learned it right here in Gazette or saw it used in assorted programs from other sources. The odds also are that you never gave line numbers any thought at all -- they're just there.

For anything except the shortest of programs, I often install an auto-number routine (many have been published in Gazette through the years) and set it to increment by ten. Then I number my first line 100. That way, when I hit Return after entering text, 110 is automatically printed for the next line, and so on. When the program is finished, with all the bugs removed, I use another routine to renumber it the same way, starting from the top with 100 and incrementing by ten. The finished program, when listed, just looks better to me.

I don't renumber by ones starting with 1 because I think it's ugly. Since much of what I write gets compiled anyway, memory use in the BASIC source code is of little concern to me.

Because I do compile a lot, I also don't GOSUB or GOTO a line number that carries only a REM statement naming a subroutine. You can get away with that in BASIC since the program just falls through to the next line and goes on. But most compilers strip away REM lines in their first pass. Then, on the second pass, some compilers balk at going to a nonexistent line, bogging down the compilation process completely and sending a most unwelcome error message. If a subroutine is included in a program, I start it at a line such as 4000, putting any REM line identifying the subroutine at 3090. Then I GOTO or GOSUB 4000, not 3090. My compilers treat me much more kindly as a result.

All of that, however interesting, is by way of introduction. The real meat of this article starts here. It concerns some techniques I've learned with which you can speed up much of your program-writing by using something other than the standard methods of line numbering. These techniques don't work with an auto-number utility, but, once the programs they produce are complete, those programs can be renumbered with any utility that performs that chore.

Because some of those who read this article will be complete beginners, I'll include explanations that may seem needless to the more experienced programmers. But I believe there can be value in these techniques even for experienced programmers. If you are one of that happy breed, I ask you to bear with me while I try to make all of this meaningful to less experienced people. Meanwhile, let me acknowledge my own debt to those from whom I have learned, including all those colleagues who write for this magazine whom I have named above.

We'll start with how you can emulate a word processor to avoid breaking words in the middle while you create neat screens of text that can be viewed one at a time -- and that can also be sent to a printer in one smooth column of type.

Let's say you want to write a help screen, or two or three screens of instructions for a game or an application. Maybe you also want to send those screens to your printer. Or let's say you're writing an article for display on screen and for printing in hard copy form as well.

Sure, you've got a word processor, but does the person using your program have the same word processor? To reach a range of users, you have to write a stand-alone program that can be read on screen, or sent to a printer, without any word processor at all. That's where this first variation on line numbering can help you do.

It's an easy way to enter text for display on screen and/or for hard copy from your printer. (Yes, you can use the same text, in just one program, for both purposes. You don't have to write one program full of PRINT statements that send text to the screen and then edit it to create a whole new program full of PRINT# statements that send text to the printer and then clutter up your disk with two completely separate programs. You really don't.)

This may be old-hat to many, but new to others: you also don't have to enter your program line by line, and as you write text you don't have to be in quote mode, which restricts the movement of your cursor. You don't even have to use PRINT statements or PRINT# statements! Try this.

CREATE SCREENS
Type 101 D Shift-A followed by a quote mark. Press Return. Cursor up and turn 101 into 102, then into 103, and so on, up to 123, pressing Return after each change. Then type LIST and hit Return. You will end up with 23 numbered DATA statements, each with a quote mark right after the word DATA. Now you can enter text after each quote mark without ever being in quote mode, thereby retaining freedom of cursor movement.

Wherever you want a blank line between paragraphs, simply hit Return on that line and leave an empty space after the quote mark; no final quote mark is needed. (And leaving out all those one-byte ending quote marks on all those lines really does save memory.)

Most of you know that once you enter a line and hit Return, that line is in memory. Some beginners may not realize that this means you can change the line number to create a new line, hit Return again, and then you have two lines in memory, each with the same text but with a different line number. This wonderful peculiarity of a computer language that uses line numbers is mentioned nowhere in the introductory guides and texts, and continues to amaze beginners. But it leads to some intriguing possibilities, among them the following.

For a second screen of text, just LIST a screenful of lines numbered in the 100's and change the initial 1 to a 2. You can do this by slapping a 2 over the first digit in each line and hitting Return, running down a whole screenful in a split second. (With one finger of your left hand tapping the 2 and one finger of your right hand hitting Return, it's amazing how quickly you can do this!) Now you have 23 lines numbered from 201 to 223, and you can simply type new text over the old to create copy for another screen.

Whatever number of hundreds you are in at a given moment, you know where you are on that particular screen -- the first line ends with 01 and the last line ends with 23, whether it's 101 - 123 or 601 - 623.

Just make sure that each line of text ends just below the second A in the word DATA, or before. That way, when the screen routine below takes over, you won't have to worry about words being broken in the middle, and every line of text will take up no more than one line on the screen. Call it word processing without a word processor.

PRINT SCREENS
Now you need a routine to print individual screens. Here it is:

```
5000 PRINTCHR$(147);
5010 FORI=1TO23
5020 READA$
5030 IFA$="-1"THENEND
5040 PRINTA$
5050 NEXT
5060 GOSUB5070:GOTO5010
5070 PRINT"PRESS ANY KEY TO CONTINUE";
5080 GETB$:IFB$=" "THEN5080
5090 PRINTCHR$(147);:RETURN
```

This screen routine will display one screen of text at a time. At the bottom of each screen, it will print a line telling the user to press any key to continue. For emphasis, you can center that line if you like, or change its color, or put it in reverse, or all of the above.

Line 5030 will keep looking for the DATA entry, "-1". When it sees that entry, it will end the screen printing program. So enter "-1" for your last DATA statement, whatever its line number. That's where the last screen will end (without any line telling the user to hit a key to go on.)

I'll explain the routine line by line. Experienced programmers may want to skip the explanation.

5000 clears the screen. You could add text, screen and border color commands in this line.

5010 tells the program to handle just 23 lines at a time.

5020 says that when the program encounters a "flag" identified as "-1" it should quit.

5040 tells it to print each DATA statement to the screen.

5050 says this process continues until the preset limit of 23 lines is reached.

5060 tells the program to go to line 5070, execute the instructions that begin there, and then come back to 5060. Then the line tells the program to resume the loop that began at 5010.

5070 instructs the program to print a line at the bottom of the screen that says "PRESS ANY KEY TO CONTINUE."

5080 waits for the key press; when the key press comes, the program goes on to 5090.

5090 says clear the screen and start all over.

To make this program useful for more than one purpose, I added a print routine that will send the text that has been viewed onscreen to the printer for a hard copy. Here's that brief routine.

```
6000 OPEN 3,4
6010 FORI=1TON: REM MAKE N ANY NUMBER LARGER THAN NUMBER OF DATA
STATEMENTS
6020 READA$
6030 IFA$="-1"THENPRINT#3:CLOSE3:END
6040 PRINT#3,A$
6050 NEXT
```

Here's how the print routine works.

6000 opens a channel to the printer rather than the screen. The number 3 represents the channel; 4 stands for the printer.

6010 starts the data-reading loop. Instead of typing "N" after the word "TO" make that a number greater than the total number of DATA statements to be printed.

6020 says read the DATA statements.

6030 tells the program that when it encounters "-1" its work is done.

6040 uses PRINT# (pronounced "print pound") to send text to the printer rather than the screen.

6050 says keep printing.

At the end of your text, place a line telling the user to enter RUN 6000 for a hard copy and the program will send a neat 40-character column to the printer. The line telling the user to "Press Any Key To Continue" won't appear in the printed version at ll.

You can, by the way, use embedded color codes which change text color on screen but don't affect the hard copy in any way. (Reverse type will, however, print in reverse. Reverse and color together will show as reverse only in hard copy form.)

KEYBOARD GRAPHICS
For text screens, whether you want hard copy or not, I've found this DATA statement method much faster and easier to work with than any method using PRINT statements. But you're probably better off with PRINT statements if you want to use keyboard graphics (those symbols on the fronts of your keys) to sketch a background or a title.

For such purposes, I've found that using 22 lines works best for a single screen, starting with 101 and ending with 122. Make your first line 101? (The question mark (?) is the abbreviation for PRINT) plus a quote mark followed by 28 spaces, then another quote mark. Press return then cursor up and turn 101 into 102, then 103, on up to 122, and then LIST.

You'll end up with a PRINT statement "canvas" on which you can draw with keyboard graphics (and uppercase letters as well.) Don't enter any embedded color codes until last, so that your drawing, when you LIST it, will be a "What you see is what you get" representation of what shows up when you run the program.

For more than one drawing, you could type new hundreds digits over the old and then draw over your original sketch for screen #2. This strike-over technique works well with text, but not so well with graphics. It can be really confusing to the eye as you work, and you're probably better off just setting up a new canvas by creating new lines numbered 201 - 222, each with 28 spaces to type in.

To hold a drawing on screen until a key is pressed, use

```
123 GETA$:IFA$=" "THEN123
```

You could use this line.

```
123 POKE198,0:WAIT198,1
```

SIMPLE ANIMATION
Now suppose you want to draw some stick figures with your keyboard graphic keys and then set them into motion. The line numbering approach for this is slightly different.

Let's say that your figures are five lines high for a human, one line for the head, one for the trunk and arms, two for the legs and one for the ground on which he or she stands.) Set up PRINT statement lines (with the quote mark following the word PRINT) ranging from 100 to 160 and incrementing by ten.

Use line 100 to clear the screen and set your colors for screen, border and text. Then, in lines 110 - 160, draw your figures as you want them to look in the first frame of your "movie."

Once that first frame looks the way you want it to, renumber 100 - 160 to 200 - 260. The easy way is to LIST the lines, then slap a 2 where the initial 1 is, hit Return, put a 2 on the second 1, and keep running down the lines until you reach 260. After that, make small changes in your drawing to represent changes of position. Repeat the process for the frame in 300 - 360 and subsequent frames, if any.

Type a line 180 with a delay loop.

```
180 FORX=1TO500:NEXT
```

Then change the hundreds digit in that line to make 280, 380, and so on. This will give you a uniform delay between frames; the timing can be adjusted later as needed by changing the 500 to a larger or a smaller number.

Finally, set up loops to repeat motion as needed.

```
90 FORI=1TO3
```

The add a NEXT statement.

```
380 NEXT
```

As with title screens, don't lay in embedded color codes and other frills, flourishes, or furbelows until your entire action program runs the way you want it to. That way, you'll preserve the WYSIWYG character of your program listing and be able to make modifications, if needed, quite easily.

SOUND EFFECTS
If you want something like a sound effect to accompany your little animated movie, put it in a subroutine and go to it with a GOSUB. I've found that these GOSUB lines are best placed before the delay loops, which is why I used line numbers 180, 280 and 380 for delay loops, leaving 170, 270 and 370 for the lines that GOSUB to any subroutine.

Here's a suggestion. It's not difficult to set up any of these special line-numbered programs, and one you try them you'll quickly become quite proficient at doing it. But you could use any program you create this way as a template for yet other programs. After all, the DATA lines and the PRINT lines contain strings of text that can be overwritten or erased completely. Even the keyboard graphics you use to draw a screen or sketch a stick figure are, in fact, bits of text, subject to editing or erasure.

Gazette, January 1995