COMPUTE!'s

COMMODORE 128 SIXTY FOUR & COLLECTION

Now for the Commodore 64 and the Commodore 128, this collection brings together some of the best games, applications, and utilities from COMPUTE! Publications. All programs run on the 64 and the 128 running in 64-mode. Additionally, there are sections detailing the advanced special features of the powerful, new 128 computer.

A **COMPUTE! Books** Publication

$12.95

COMPUTE!'s COMMODORE SIXTY FOUR & 128 COLLECTION

COMPUTE!™ Publications, Inc. abc
One of the ABC Publishing Companies
Greensboro, North Carolina

The following article was originally published in *COMPUTE!* magazine, copyright 1983, COMPUTE! Publications, Inc.: "Ultrasort" (September—originally titled "Ultrasort for Commodore").

The following articles were originally published in *COMPUTE!* magazine, copyright 1985, COMPUTE! Publications, Inc.: "Advanced Sound Effects on the 128" (February—originally titled "Advanced Sound Effects on the 64"); "Mindbusters" (April); "TurboDisk: High-Speed 1541 Disk Loader" (April—originally titled "TurboDisk: High-Speed Disk Loader for Commodore 64 and Expanded VIC-20").

The following articles were originally published in *COMPUTE!'s Gazette*, copyright 1983, COMPUTE! Publications, Inc.: "Disk Defaulter" (November—originally titled "VIC/64 Disk Defaulter"); "UnNEW: Program Lifesaver" (November—originally titled "VIC/64 Program Lifesaver); "Foolproof INPUT" (December—originally titled "Foolproof INPUT for VIC and 64").

The following articles were originally published in *COMPUTE!'s Gazette*, copyright 1984, COMPUTE! Publications, Inc.: "Making Calendars" (April); "Ultrafont +" (July); "Campaign Manager" (August); "Sprite Magic: An All Machine Language Sprite Editor" (August—originally titled "Sprite Magic: An All-Machine-Language Sprite Editor"); "Quiz Master" (October—originally titled "Quiz Master for the 64"); "Function Key" (November).

The following articles were originally published in *COMPUTE!'s Gazette*, copyright 1985, COMPUTE! Publications, Inc.: "Debugging BASIC Programs" (January and February—originally titled "Debugging BASIC: Part 1 and Part 2"); "Trap 'Em" (January); "Commodore 128 Peripheral Ports" (March—originally titled "Commodore Peripheral Ports"); "Disk Directory Sort" (March); "Heat Seeker" (March); "Commodore 128 CP/M Plus" (April—originally titled "What Is CP/M?"); "NoZap: Automatic Program Saver" (April); "Triple 64" (April); "Inside the Commodore 128: A Hands-On Look" (June—originally titled "Inside the 128: A Hands-On Look at Commodore's Newest Computer"); "Squares" (June).

The following article was originally published in *COMPUTE!'s Machine Language Routines for the Commodore 64*, copyright 1984, COMPUTE! Publications, Inc.: "64 Freeze."

The following article was originally published in *COMPUTE!'s Second Book of Commodore 64 Games*, copyright 1984, COMPUTE! Publications, Inc.: "Writing Text Adventures for the Commodore 64 and 128" (originally titled "Puzzles, Palaces, and Pilgrims: Writing Text Adventures for the Commodore 64").

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-942386-97-3

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

Contents

Foreword

COMPUTE!'s Commodore 64/128 Collection contains programs that you can type in and run on your Commodore 64 or 128 in its 64 mode. These programs, originally written for the Commodore 64, are some of the best ever published by *COMPUTE!* and *COMPUTE!'s Gazette* magazines.

But in addition to great games and utilities, you'll find discussions of programming in BASIC and writing adventure games, and a special section on the new features of the 128: CP/M, BASIC 7.0, 128 mode, and the peripheral ports.

The articles are clearly written and easy to understand. There are short programs for beginners and fast, commercial-quality games as well. And as usual, "Automatic Proofreader" and "MLX" will help you avoid errors as you type in the programs. As with all COMPUTE! books, each program has been thoroughly tested.

If you prefer, you can purchase a disk with all the programs found in this book from COMPUTE! Publications by using the coupon found in the back or by calling toll-free (800) 334-0868, in North Carolina call (919) 275-9809.

1
Inside the 128

1

Inside the Commodore 128: A Hands-On Look

Charles Brannon

The new Commodore 128 Personal Computer has generated quite a bit of interest, especially by current owners of the popular Commodore 64. Is the 128 a significant enhancement or just a warmed-over 128K version of the 64? A hands-on look at the 128 provides a new appreciation for this intriguing machine.

The 64's Reign

Soon after it was introduced, the Commodore 64 proved to be the leader of a new wave of home computers. Even at the original price of $600, the 64 came equipped with as much memory as $2,000 business machines, along with arcade-quality graphics, detailed animated sprites, and a unique sound synthesizer that brought realism to what was formerly just bleeps and tones. The 64 became one of the most popular computers ever, selling over 2,000,000 units worldwide.

The 64 is firmly established, with over 6000 programs to its credit. But as good as the 64 is, we've been waiting three years for an encore. Although it's been high time for an enhancement, no one wants to give up his or her personal software library. Commodore's answer, the Commodore 128 Personal Computer, provides true 64 compatibility, plus a real advance in power and flexibility. The Commodore 128 is literally three computers in one: a Commodore 64 with the familiar 40-column display, sprites, SID chip, and BASIC V2; an enhanced 64 with 128K and all 64 features, plus 80 columns and BASIC 7.0; and a true CP/M-compatible machine, promising the ability to run off-the-shelf CP/M software. And all at a price almost anyone would call reasonable: under $400.

Compared with the 64, the 128's console is much bigger, perhaps to imply more power, but probably necessary to hold the hardware of three computers. The main part of the keyboard is identical to the 64's, except that the function keys have been moved to the upper-right corner and rearranged horizontally. There is a numeric keypad with +, −, ., and an ENTER key (synonymous with the RETURN key). Along the top of the keyboard are ESC, TAB, ALT, CAPS LOCK, HELP, LINE FEED, 40/80 DISPLAY, and four separate cursor keys.

None of these additional keys, not even the keypad or separate cursor keys, functions in the 64 mode, for the sake of true compatibility. Adding extra programming in ROM to support these keys in 64 mode might be just enough to prevent some 64 software from working properly. Commodore is staunch on this; anything less than 100 percent compatibility isn't good enough.

The New King

In the 128 mode, the 40/80 DISPLAY key selects which screen mode is used as the default. This key is checked at power-on, when RUN/STOP–RESTORE is pressed, or when the RESET button (found next to the power switch) is pressed. This key has no meaning in 64 mode since 80 columns are not available, again for the sake of compatibility. In either 128 or CP/M mode, the same VIC chip used on the 64 displays 40 columns, graphics, and sprites. The 40-column screen can be seen only on a television or composite monitor, not on the RGB display.

The RGB monitor displays twice as many pixels and characters as 40 columns, and achieves color purity since the signal is separated into the red/green/blue color components. (A composite signal has all the color information mixed together, which makes it difficult to cleanly separate these colors.) A special video chip is used for 80 columns, and the 80-column screen can be seen only on the RGB monitor. All 16 colors are available in 80 columns (although the Commodore-1 color, normally orange, appears as dark purple) as well as reverse video and underlining. Unlike the 40-column mode, there are 512 characters available in 80 columns, which means you can get both uppercase, lowercase, and all keyboard graphics simultaneously.

This 80-column chip is for text only—it does not support bitmapped graphics or sprites. You can redefine the character set, though, and set up a small 640 × 48 simulated bitmapped window. The 80-column video chip uses 16K of dedicated screen memory, but none of the 128K memory is used for 80 columns, so in effect this machine actually has 144K of total RAM.

There are three ways to switch between 40 and 80 columns: toggle the 40/80 switch and press RUN/STOP–RESTORE, press ESC-X in BASIC, or enter the command SCREEN 0 for 40 columns, or SCREEN 5 for 80 columns. Remember that these screens are independent. If you have two monitors hooked up, these commands reroute screen printing to the appropriate monitor (although both screens remain displayed). Commodore's 1902 monitor is ideal for the 128; it has built-in color composite video, split-signal composite video (as used on the rear connections of the 1701/1702 monitor), IBM-compatible RGB, and analog RGB (for use with the Amiga). With the 1902, you must manually flip a switch after you change screen modes.

This can be cumbersome, but Commodore feels that you'll probably stay in one mode or the other, a reasonable assumption. This scheme does let you have two simultaneous displays. Perhaps one screen could show color graphics, while your program listing is displayed on another. One can envision dual-perspective games with players having their own independent screens.

The 1902 composite/RGB display will probably sell for under $400. The least expensive route, though, is to use a television for 40 columns and a monochrome (black and white) monitor for 80 columns. Commodore will sell a special cable to connect the RGB port to a monochrome monitor. The cable can be used with Commodore's inexpensive 1901 monochrome display and with other monochrome monitors.

A Smarter, Faster Drive

The new 1571 disk drive further amplifies the power of the 128. In 64 mode, the 1571 behaves just like a 1541. The 1571 we worked with was not quite ROM-compatible with the 1541 (our "TurboDisk" program did not work with it), but we were assured that 1541 compatibility, a high priority, was being improved. In the 128 mode, the 1571 shows its true power, boosting storage capacity to 360K (as opposed to 170K on the 1541), and transferring data from seven to ten times faster than the 1541.

The enhanced storage is due to the 1571's double-sided design (there are two read/write heads), so you'll have to use the somewhat more costly double-sided disks. You can still use a 1541 in the 128 mode, and the 1571 can be programmed to be 1541-compatible in the 128 mode. So you don't have to write off your current disk drive when you upgrade to the 128. Other 64 peripherals also work with the 128, so hold on to your printer and modem if you upgrade.

The 1571 is also optimized for the CP/M mode, although you can use a 1541 drive in the CP/M mode. In CP/M mode, the 1571 can store 410K. Commodore has designed a new version of CP/M called CP/M Plus, which gives newly written CP/M applications the ability to access VIC-chip graphics and sprites, RGB color 80 columns, and the SID sound synthesizer—snazzy features for a CP/M machine. Unlike Commodore 64 CP/M, CP/M Plus is a true native Z80 implementation. The entire system resources are available to CP/M Plus, since the Z80 stays in control. Commodore is busy converting CP/M disks to 1541 format so that they will run both on the 128 and on 64 CP/M with a 1541 drive. But the new drive can be reprogrammed to read many disk formats. A configuration program can be used to let the drive read common CP/M formats, including disks formatted for Osborne and Kaypro machines.

As long as programs conform to CP/M portability guidelines, you'll be able to insert off-the-shelf CP/M software and boot it up (though this won't take advantage of the enhanced options of CP/M Plus). When we visited Commodore to test prototypes of the 1571, we took some Osborne disks along with us, but the 1571 drive we used was not modified to read our disks, so we were unable to verify this. Commodore indicated that several CP/M software manufacturers were interested in developing new CP/M software for the 128.

Not So BASIC

We were most impressed by BASIC 7.0 in the 128 mode. It's the most powerful version of BASIC we've seen for personal computers, topping even IBM's Advanced BASIC. With Commodore 64 BASIC as its foundation, it combines the best of *Simons' BASIC*, *Super Expander*, Plus/4, and Disk BASIC 4.0 commands, as well as new commands written especially for the 128. There are over 80 new commands and functions. At the time we visited Commodore, programmers were adding even more commands. And all 128K is available for programming: 64K for the length of your BASIC program, and 64K for storage of variables, strings, and arrays (minus the memory used by the operating system and 40-column screen map). The only thing missing is long variable names; you're still limited to two significant characters.

All disk commands from BASIC 4.0 are supported, permitting 128 owners to run some CBM 4032/8032 programs. These commands replace the need for OPEN 15,8,15: PRINT#15,"command": CLOSE 15. Most disk commands can be used with a dual-drive disk system (with the drives called 0 and 1), and with several drives addressed with different device numbers. SHIFT–RUN/STOP defaults to the disk drive, loading and running the first program on the disk. DLOAD and DSAVE are used to retrieve and store BASIC programs. CATALOG or DIRECTORY displays the disk directory without erasing any program in memory. SCRATCH lets you erase files from disk, but first asks ARE YOU SURE? The HEADER command is for formatting disks.

COLLECT performs a validate, freeing up any improperly allocated sectors. COPY and CONCAT let you copy or combine disk files on the same disk or between drives on a dual-drive system (but not with separate drives addressed with different device numbers). BACKUP can also be used only with a dual drive to copy one disk to another. APPEND lets you add new data to an existing file. DOPEN and DCLOSE make file handling easier, and RECORD makes relative files a breeze. The reserved variables DS and DS$ let you examine the disk error channel. DCLEAR clears all open disk channels.

There's a complete set of programming tools. AUTO starts automatic line numbering, DELETE erases program lines, HELP shows the offending statement after an error message, RENUMBER permits you to renumber any part of a program, TRON and TROFF toggle trace mode, and KEY lets you display the current function key definitions or define your own function keys. You can also conveniently convert from hexadecimal to decimal or vice versa with the functions HEX$ and DEC. In addition to AND and OR, you can now perform a bitwise Exclusive OR (XOR).

What ELSE?

Structured programming enthusiasts need never use GOTO again. IF-THEN now has an ELSE clause, as in IF A=1 THEN PRINT "A IS 1":ELSE PRINT "A IS NOT 1." BEGIN/BEND lets you set aside a block of lines that are executed only if a preceding IF-THEN works out as true. DO:LOOP UNTIL, DO:LOOP WHILE, DO UNTIL:LOOP, and DO WHILE:LOOP all execute a block of commands *while* a certain condition is true, or *until* a certain condition proves to be false. EXIT can be used to skip out of a loop.

RESTORE can now be followed by a line number to let you start reading any section of DATA.

TRAP transfers execution to a specified line number when an error occurs. Your program can examine the error number in the reserved variable ER, the number of the line that caused the error in EL, and the error message with the function ERR$. After you've handled the error, RESUME returns control to the statement after the error or to any line number.

Text processing is enhanced with INSTR, which finds the position of a substring within a larger string. PRINT USING lets you define a format field for printing, making it easy to set up columnar tables and forms. WINDOW sets up a smaller screen that scrolls independently from the rest of the screen. WINDOW can be used to emulate simple Macintosh-style windowing.

Machine Language Aids

Machine language (ML) programmers will appreciate the built-in ML monitor, entered from BASIC with the MONITOR command. The monitor pretends that the 128K of memory is contiguous and permits five-digit hexadecimal addresses. It makes full use of 80 columns if selected. The monitor works much like 64 *Supermon,* with commands to assemble, disassemble, fill, go to address, hunt through memory for a hexadecimal string, load, display memory with ASCII equivalents, display registers, save, transfer a block of memory, verify a saved program, exit to BASIC, modify memory, modify registers, and display disk error status.

BASIC commands for ML include BLOAD and BSAVE to load and save ML programs or other binary files, and BOOT to load and run an ML program. The familiar USR, WAIT, POKE, PEEK, and SYS commands can now be used to reference the second 64K of memory with the BANK command. SYS can be followed by four parameters that are transferred respectively into the accumulator, X register, Y register, and status flag register. On return from SYS, RREG can be used to transfer the contents of A, X, Y, and the status register into four variables. This makes it much easier to pass information back and forth between BASIC and ML.

The 8502 microprocessor used in the 128 mode is opcode-compatible with the 6502 and 6510, but can now function at two megahertz (MHz), twice the speed of the 6502. All VIC/64 Kernal routines are supported, making program translation much easier. New Kernal routines support special features of the 128, including special routines for memory management.

A reset switch near the power switch can be used to cold start the machine. Holding down RUN/STOP with the RESET key initiates a "lukewarm" start. It's a more thorough reset than RUN/STOP–RESTORE, but your program is not lost. This reset puts you into the ML monitor, where you can exit back to BASIC with no harm done.

Sound and Graphics

No more POKEs for SID chip sound. BASIC 7.0 includes several commands for music and sound effects. SOUND sets the frequency, duration, and waveform of a sound effect. You can also specify a sweeping effect. PLAY is a minilanguage of its own. You can use it to play strings of notes, specifying note names, durations, sharps/flats, dotted notes, and rests. You can use it to synchronize three-voice music, set the filter, and control individual volume for each voice. Each voice can play from a set of predefined envelopes that simulate one of ten musical instruments: piano, accordion, calliope, drum, flute, guitar, harpsichord, organ, trumpet, and xylophone. You can customize these preset instruments with ENVELOPE, customize the programmable filter with FILTER, set the overall VOLume, and the TEMPO of music.

BASIC 7.0 offers a rich vocabulary of graphics commands. GRAPHIC is used to enter either the multicolor 160 × 200 graphics screen, the hi-res 320 × 200 graphics screen, the 40-column text screen, or the 80-column text screen. GRAPHIC allows you to define a text window and can either clear the screen or leave previous graphics in place. SCNCLR can also be used to clear the screen. When you enter a graphics mode, the start of BASIC is moved beyond the end of the graphics screen. GRAPHIC CLR is used to deallocate the memory used by the graphics screen. RGR returns the number of the current graphics mode.

DRAW is used to plot a single point, or draw a single or a connected line to create complex shapes. LOCATE is used to set the position of the graphics cursor without plotting any point. BOX can draw any rectangle or filled rectangle, at any angle. CIRCLE is used to draw circles, ovals, arcs, or any regular polygon, at any angle of rotation. You can place text anywhere on the graphics screen with CHAR. You can also use CHAR on the text screen to simulate PRINT AT. COLOR is used to set any of the color registers, and the function RCOLOR reads which color is assigned to a color register. PAINT can fill any shape with any color. GSHAPE can "pick up" any block of the screen and store it in a string. This shape can then be copied back to any place on the screen with SSHAPE.

A pixel can be tested with the function RDOT, which returns the color of the pixel at the specified row and column. The WIDTH command specifies the size of pixels plotted. A WIDTH of 2 makes all lines double-wide. And finally, the SCALE command lets you pretend that the screen is actually 1024 × 1024 pixels across and down. You can use this range in your drawing statements, and the coordinates are automatically scaled to fit the actual screen size.

BASIC 7.0 just wouldn't be complete without sprite commands. If you've been stymied by POKE and PEEK for sprite control, as well as the infamous "seam," you'll really appreciate the following sprite commands.

Sprites and Gaming

First, BASIC 7.0 includes a simple sprite editor. Just type SPRDEF, and a box appears on the screen. Enter which sprite you'd like to define, then use the cursor keys and the number keys 1–4 to draw squares on the grid. When you're through, the sprite is stored into a reserved section of memory. This memory can be saved to disk with BSAVE, then recalled within your program with BLOAD, eliminating the need for DATA statements.

To set up sprite parameters, use SPRITE. This command turns on the sprite; sets its color, priority, initial X and Y positions; and sets hi-res or multicolor for that sprite. You can then use MOVSPR to position the sprite anywhere on the screen. MOVSPR can also be used to set the sprite into motion. After you specify the speed and angle, the sprite moves on its own. Your program continues in the meantime. (Sprites are updated in this mode during the IRQ interrupt.) While a sprite is in motion you can read its position with RSPPOS. You can transfer the sprite pattern into any string or copy a sprite pattern from a string into any sprite. In combination with SSHAPE and GSHAPE, you can "pick up" a block of the screen and turn it into a sprite, and "stamp" the sprite pattern anywhere on the graphics screen.

SPRCOLOR sets the multicolor registers shared by all sprites, and the function RSPRCOLOR reads the sprite multicolor registers. The COLLISION statement transfers control to a specified line number when two sprites touch or when a sprite touches part of the screen background. Your collision routine can see what caused the collision with the function BUMP.

No longer are PEEKs, POKEs, or ML necessary to read the game controllers. The function JOY returns the status of either joystick. POT returns the position of one of the four paddles, and PEN is used to read the X,Y coordinates of the light pen.

A few miscellaneous commands: SLEEP is used as a delay loop, pausing from 1 to 65,535 seconds. GETKEY is like GET, but waits for a keystroke. GO64 exits to the 64 mode, but first asks ARE YOU SURE?, since anything in memory in the 128 mode will be lost.

BASIC 7.0 has almost every command a programmer would need. There are almost too many commands, extending the time it takes to learn a programming language. However, you need not memorize every command; just learn commands as you need them. You'll at least want to be aware of the commands that are available so that you won't reinvent the wheel by POKEing your way to sound or graphics.

Memory

Using an external memory cartridge, the 128 can be expanded up to 512K. This memory is not directly available for programs, though, but is used as a RAM disk, which simulates the functions of a disk drive, using memory chips as the storage medium. This provides faster throughput than a hard disk, but all information is lost when the power is turned off. You need to dump the contents of a RAM disk to a more permanent form of mass storage at the end of each session with the computer.

A special memory management unit (MMU), located at $FF00, is used to control the 128's complicated memory map. The MMU interprets memory addresses even before the microprocessor sees them. It permits the programmer to swap between 64K banks of memory, but can leave a small portion of memory as common memory. For example, you don't always want zero page and the stack to disappear when you change banks. The MMU permits you to bank between four 64K banks, and allows multiple banks of 256K, up to one megabyte of memory.

The MMU controls whether the VIC chip or 80-column chip controls screen display, and even senses the position of the 40/80 DISPLAY switch (though the software must interpret this switch). The MMU controls access to RAM or ROM, allowing either to be visible in the memory map. A programmer can set up a series of preset memory configurations and quickly select them by writing to the MMU. The address of the VIC chip can be relocated anywhere within the virtual 256K memory space.

The MMU also controls the fast serial port used with the 1571 disk drive (and conceivably with other fast peripherals). It determines the clock speed of the 8502, and controls which of the three microprocessors (6510, 8502, Z80) is in control. And although not supported in ROM, it's possible to have all three microprocessors running by quickly switching between them.

The 128 is a logical upgrade of the 64. Without sacrificing 64 compatibility, the 128 fulfills almost anyone's wish list. BASIC 7.0 gives programmers freedom to program without POKEs or cumbersome ML routines. The 80-column display, two-megahertz microprocessor, 128K of memory (theoretically expandable to a megabyte), CP/M Plus, and fast double-sided disk drive make the 128 a capable business machine, competitive with the much more expensive IBM and Apple computers. As usual, though, we'll still have to wait for software to be written that takes advantage of these features. Although you can use existing 64 and CP/M programs, it looks like you'll have to write your own 128 mode programs for awhile. But that's not all bad, is it?

Commodore Peripheral Ports

Ottis R. Cowper

The Commodore 128 has several connectors which allow you to communicate with disk drives, modems, and other peripherals. Many of the connectors are compatible with previous Commodore 64 peripheral ports, but there have been a few additions and modifications. This introduction to peripheral ports includes information on the VIC-20, 16, Plus/4, 64, and 128.

Commodore computers provide their users with a variety of methods for communicating with the outside world. The devices from which the computer receives input or to which it sends output (or both) are generically called *peripherals,* and the connectors where peripherals are attached to the computer are referred to as *ports.* Each of the several ports has distinctive characteristics that make it suitable for particular applications.

For some ports, the computer's operating system—the ROM which controls the machine's functions—provides routines that handle much or all of the "dirty work" of communicating with peripheral devices. To use other ports, you must program all the necessary support routines yourself. That task can range from very easy (for example, reading a joystick) to quite complex (interfacing with a parallel printer through the user port, for example).

The Serial Port

For most users, the serial port is the major data artery of the computer. As the connection point for disk drives and printers, it's the port through which most information exchanges take place. This is the one port that is the same on the Commodore 128, 64, VIC-20, Plus/4, and 16. Well, almost the same—there are some signal timing differences. (The VIC-20 transfers data at a slightly faster rate than the others, which is why the VIC is listed as incompatible with some Commodore printers, and why the original 1540 disk drive was only for the VIC.) However, when the 128 is used in either 128 or CP/M mode with the Commodore 1571 disk drive, its serial port is also capable of high-speed operation in which data is transferred through the port many times faster than in any of the other computers.

Obviously, this port is bidirectional—data can flow both in and out with equal ease. The signal format used to exchange data serially over the six lines provided through this port is unique to Commodore. The format should not be confused with the more standard RS-232 serial communications format used by numerous peripherals; RS-232 communication is handled through the user port (see below). The serial port is essentially a stripped-down version of the parallel IEEE-488 port used for most data communications in Commodore's earlier PET/CBM models. As the term *serial* implies, data can be transferred only one bit at a time (and in only one direction at a time, either in or out). Three of the other lines control the direction of data flow, and whether the signals on the data line are to be interpreted as data or as commands to the peripheral device. The computer's RESET line is also present at this port, which explains why the disk drive resets whenever the computer is turned on or off.

The operating system fully supports communications through this port. By addressing a peripheral attached to this port with a device number, and using OPEN, CLOSE, PRINT#, INPUT#, and related routines provided by the operating system, you can avoid worrying with the details of controlling the individual signal lines. Any peripheral addressed with a device number between 4 and 31 (the highest device number allowed) is assumed by the computer to be connected to this port.

Commodore has established several standards for device numbers: Printers are usually device 4, although some can be changed to device 5, the 1520 Printer/Plotter is designed to be device 6, and device numbers 8 and above usually refer to disk drives. Device 8 is the default number for the disk drive, and almost all software assumes the disk drive will have this device number; device 9 is the most common choice for a second drive. Commodore 1541 and 1571 drives allow you to select any device number via software, or numbers 8–11 via hardware.

The use of a unique signal format for communication with the disk drive is not unusual; almost all computer manufacturers use a proprietary disk interface compatible only with their own products. What is unusual is that this same nonstandard format is also used for communications with printers. Since so much software assumes that printers will be connected through the serial port (as device 4), most third-party interfaces for non-Commodore printers also attach to this port. These interfaces act as interpreters, reading the Commodore-format serial signal from the port and converting it to the more standard parallel (eight bits at a time) format used by most printers.

The Memory Expansion Port

This is often referred to as the cartridge port, since ROM cartridges are the peripherals most often attached through this connector. The lines available at this port include most of the address, data, and control lines of the microprocessor chip that is the heart of the computer. Thus, any peripheral which needs to be intimately tied to the workings of the computer—for example, ROM that must be addressed by the microprocessor—is connected through this port. The operating system does not support any devices through this port; in essence, anything attached here is no longer a peripheral, but part of the computer itself.

Many of the same lines are available on corresponding pins of the expansion port connectors used in the VIC, 64, and 128, but the connectors themselves are different sizes, so cartridges designed for the 64 and 128 cannot be used on the VIC, and vice versa. However, cartridges for the 64 can be used on the 128, and if a 64 cartridge is installed in the 128 when it is turned on, the 128 will come up in 64 mode and start executing the program from the cartridge. The Plus/4 and 16 have identical 50-pin connectors for this port (as opposed to the 44-pin connectors used in the VIC, 64, and 128), so while there is some compatibility of cartridges between these two models, no VIC, 64, or 128 cartridges or memory port peripherals can be used with the Plus/4 or 16.

Commodore has announced a 512K memory expander for the 128, which would be connected through this port. The additional memory is addressable in 64K blocks and can be set up to act as a RAM disk—allowing lightning-speed saving and loading. Of course, any programs would have to be transferred to disk or tape for permanent storage, as all data disappears from memory when the power is turned off.

The User Port

This port (sometimes called the RS-232 or modem port) was designed with the experimenter in mind. Just as the memory expansion port gives you access to a number of the microprocessor's control lines, this port gives you access to many of the control lines of one of the interface adapter chips. Using these lines, a wide variety of peripherals could be connected, since both serial and eight-bit parallel communications are available.

Unfortunately, most of this flexibility goes unused since it isn't supported by the operating system. Most home computer users today are more interested in software than in tinkering with hardware projects, so this port is most frequently used for its one function supported by the operating system: RS-232 serial communications.

RS-232 is the name of the most common serial communications standard. If you use the operating system to address device 2, data directed to that device will be transferred through the user port in an approximation of RS-232 format. Actually, the signal format is true RS-232, but the voltage levels are different from those prescribed. The RS-232 standard calls for voltage levels of −12 to +12 volts, and the user port only provides levels of 0 to +5 volts. Adapters are available—from Commodore and other sources—to convert the signal voltage to the proper levels. These adapters are not necessary if you're going to use Commodore's modems, but they are required to use any standard RS-232 equipment.

The 24 pins of this port have a similar configuration on the VIC, 64, and 128, so many devices designed to interface to this port—the VICmodem and 1650 Automodem, for example—can be used on any of these models, although the software to run the devices will generally be different. The Plus/4 also has the same 24-pin connector, but the computer casing around the connector is smaller, so neither the VICmodem nor Automodem can be plugged into the Plus/4. (Commodore's new Modem300 works with the VIC, 64, 128, and Plus/4.) The Commodore 16 has no user port, so it is as yet unclear how (or if) a modem may be used with that computer.

Since eight-bit parallel data communications is available through this port, it might seem surprising that it's not commonly used for interfacing with printers. After all, it would appear on the surface to be simpler to write a machine language program to simulate the commonly used Centronics parallel format through this port than to go to all the trouble of designing the hardware interface to convert the data from the serial port to the proper parallel format.

The reason this isn't often done is that almost all Commodore software expects the printer to be device 4 on the serial port, and in the long run it proves easier to seek a hardware solution to allow you to use the built-in operating system routines as provided in ROM. That way, you don't have to worry about having to load your printer handler routine into memory before you can use it, finding a safe place in RAM to store the handler routine, and so forth.

For more information on interfacing through the user port, see the article "Using the User Port" in *COMPUTE!'s First Book of Commodore 64.*

The Control Ports

These ports (or this port, in the case of the VIC, which has only one) are usually referred to as the joystick ports, since they are most commonly used for joysticks. BASIC 2.0—in the VIC, 64, and 128 in 64 mode—does not support any devices through these ports, so you must communicate with this port by using PEEKs and POKEs. However, BASIC 3.5 in the Plus/4 and 16 and BASIC 7.0 in the 128 in 128 mode both have built-in statements for reading the status of controllers connected to these ports.

Joysticks are simple devices consisting of five switches—one for each of the four principal directions, plus one for the fire button. The switches are normally open; pushing the joystick in one of the principal directions closes one of the switches, while pushing the stick toward one of the diagonals closes two switches simultaneously. Pressing the fire button closes the switch connected to that line. In each case, closing a switch grounds the associated line at the port, which causes the value of the bit associated with that line to change from 1 to 0. For example, in the 64 (or the 128 in 64 mode) where the port must be read with a PEEK, pressing the fire button on a joystick connected to port 1 causes the value in memory location 56321 to change from 255 to 239 as bit 4 changes from 1 to 0. Using the 128 in 128 mode, BASIC 7.0 provides a simpler system. The JOY function returns a value from 1 to 8 indicating toward which of the eight possible directions the stick is being pushed, or 0 if the stick is at the center position. An extra 128 is added to the direction value if the fire button is pressed. JOY(1) is used to read the joystick connected to port 1, and JOY(2) reads port 2.

In addition to joysticks, the ports can be used to read any other device that behaves like a joystick, such as a trackball or the 128's "mouse" controller (which is essentially a trackball turned upside-down). Atari and Coleco joysticks are functionally identical to Commodore joysticks and can be used interchangeably. However, owners of other Commodore computers should avoid controllers designed for the Plus/4 or Commodore 16. On those computers Commodore has abandoned the widely used DB-9 joystick connector in favor of a nonstandard connector, so existing joysticks cannot be used.

In addition to the joystick, these ports in the VIC, 64, and 128 can be used to read paddle controllers. (The Plus/4 and 16 have no circuitry for reading paddles.) Paddle controllers, which always come in pairs, are actually just variable resistors which provide variable voltage levels to two lines on the port. Special circuitry within the computers (in the VIC chip in the VIC-20, and in the SID chip in the 64 and 128) calculates a digital value corresponding to the voltage level. The value ranges from 0 to 255 as the voltage on the lines changes from 0 to 5 volts. With the 128 in 128 mode, BASIC 7.0 provides the functions POT(1)–POT(4) to read each of the four paddles: POT(1) and POT(2) from port 1, and POT(3) and POT(4) from port 2. Other devices which operate like paddles—providing a varying voltage input—can also be read through these ports; graphics tablets are a good example.

Each paddle usually also has a button, but instead of being read like the joystick buttons, the paddle buttons are connected to the lines for two of the joystick directional switches. One paddle button corresponds to the joystick's right directional line, and the other to the line for reading joystick left. By convention, the paddle that uses the right directional line for its button is called the right paddle, and the one that uses the joystick-left line is the left paddle. In BASIC 7.0, 256 is added to the value returned by the POT function if the fire button on the paddle is being pressed.

Unlike Atari joysticks, Atari paddles are not completely interchangeable with those made by Commodore. While Atari paddles can be used with Commodore computers, they have a higher resistance and thus are less accurate for Commodore systems. (A half turn on Atari paddles corresponds roughly to a full turn on Commodore paddles.)

One additional type of peripheral—the light pen—can also be connected to this port. (On those models with more than one joystick port, the light pen can be connected only to port 1.) The pen contains a phototransistor that switches when it detects the electron beam of the video display sweeping past. A line is connected from the phototransistor through the port to the chip that generates the video signal (the VIC chip in the VIC, the VIC-II chip in the 64 and 128, and the TED chip in the Plus/4 and 16).

When the video chip receives the signal from the pen, it latches (stores) the current position of the raster (electron beam) in a set of registers (memory locations within the chip). The stored value can then be read, and the position where the pen is touching the screen can be calculated. In 128 mode, the 128's BASIC 7.0 provides the functions PEN(0) and PEN(1), which return the x- and y-coordinates, respectively, of the light pen's position on the screen.

The Audio/Video Ports

These connectors are not really ports in the true sense of the word, since data cannot be transferred through them. Instead, they provide a connection point to the computer's video and audio signals. With the exception of an audio input line on the 64 and 128, all lines at these ports are outputs only. The audio/video port of the VIC, 64, Plus/4, and 16, and the Video1 port of the 128 are all compatible, but compatible doesn't mean identical. The VIC and early models of the 64 used a five-pin socket for this port, while the 128, Plus/4, 16, and later 64s use an eight-pin socket. In either case, the port provides a composite video signal and an audio signal. Corresponding video and audio inputs are found on most black-and-white or color video monitors. The eight-pin versions of the port also provide separate chrominance (color) and luminance (brightness) signals. When used with monitors that can accept this signal format (such as the rear connections of Commodore's 1701 and 1702 monitors), the eight-pin version can provide much sharper color contrast.

The audio input line of the 64 and 128 allows you to mix sound from external sources with the sounds created by the SID chip in the computer. However, this line runs directly to the audio input pin on the SID chip, so you must be careful to feed in only low-level (unamplified) sound sources. There's no way to process the incoming sound, but it can be mixed with the sound of the SID chip, and the SID chip's filters can be used as a programmable equalizer for the sound coming in.

In addition to the standard audio/video port, the 128 also has a second port, Video2, for 80-column output. This port is connected to the separate 80-column video chip in the 128. This chip—which can be used only from 128 or CP/M mode—provides output in RGBI format, which means that it provides separate control signals for the red, green, and blue (RGB) electron guns that produce the color video display. This allows for much sharper displays than the standard video format, where the composite signal is a blend of the RGB signals. Alternatively, a monitor can be connected to this port using only the intensity signal (the *I* in RGBI) for an extremely crisp monochrome display. It's even possible with two monitors to have simultaneous displays on 40-column and 80-column screens, since the two displays are maintained by separate video chips. However, if you wish to avoid having to purchase two separate monitors, you'll need a unit such as Commodore's 1902 color monitor which can handle both composite video and RGBI input.

The Cassette Port

This port is designed for one particular peripheral, the Commodore Datassette recorder. There are now two models of the Datassette, the 1530 (or equivalent C2N) for use with the VIC, 64, and 128, and the 1531, for use with the Plus/4 and 16. As with joysticks, the only difference between the two is the plug on the end of the connecting cable. Commodore has used a new and incompatible type of connector for this port on the Plus/4 and 16.

Three of the six lines from this port are used for writing a signal to the tape, reading a signal from the tape, and testing whether a button is pushed. Note that since there is only one line (labeled Cassette Sense) to test the buttons, it's possible to check only whether *any* buttons are pressed, not which particular button or buttons are pressed. Thus, if you're supposed to press PLAY and RECORD and accidentally press only PLAY, the computer won't be able to detect the mistake. Other lines supply power to the tape motor (9 volts) and for the electronics in the Datassette (5 volts). Some other peripherals—for example, several brands of printer interfaces—also make use of the 5-volt power source available here.

Communication through this port is fully supported by the operating system, with the Datassette being designated as device 1. Device 1 is the default storage device; unless you specify otherwise, all your SAVEs and LOADs will be directed to the Datassette. In addition to SAVE, VERIFY, and LOAD, the OPEN, CLOSE, PRINT#, and INPUT# statements provide all the features necessary for storing and retrieving data on tape, so programmers rarely need to worry about the intimate details of interfacing to this port, such as what sort of magnetic pattern is actually used to represent a byte of data on tape. Nevertheless, it's possible to program several of the individual lines of this port to achieve special effects; for an example, refer to the "TurboTape" articles in the January and February 1985 issues of *COMPUTE! m*agazine.

Commodore 128 CP/M Plus

Charles Brannon

One of the three operating modes of the Commodore 128 is the CP/M mode, utilizing a Z80 microprocessor. Commodore owners have generally not paid much attention to CP/M, and though it has been available for the 64 for some time now, many people still don't know what it is or what it does.

The CP/M System

CP/M is an acronym for Control Program for Microcomputers. In essence, CP/M is merely an operating system, primarily for controlling disk access. An operating system is the base software for a computer. It takes care of routine system tasks and provides a link between the computer and any other software you may be running.

CP/M began when Gary Kildall, working for Intel, developed a package of compactly written subroutines for the tiny four-bit 4004 microprocessor. These useful subprograms could be used by other programs, simplifying the work of a programmer. As technology advanced, CP/M became a full-blown operating system for the Intel 8080 microprocessor, and was upgraded for the 8080-compatible Zilog Z80 microprocessor. Curiously, Intel, the designer of the 8080, was not interested in CP/M, and gave Kildall the go-ahead to market it on his own. He started up a company called Digital Research. (Digital is still going strong; they recently developed GEM, the Macintosh-like desktop metaphor and graphics operating system that runs on the IBM PC series, the new Atari ST, and most likely will be available for the Commodore Amiga.)

Before CP/M, there was no real operating system for these early computers, so it was quickly seized upon by most users and manufacturers of Z80 computers. There were no successfully competitive operating systems, and CP/M easily became a standard. Since almost everyone had CP/M, all the Z80 machines had more in common with each other. CP/M made it possible for one program to run on many different computers.

Most Z80 computer systems included a keyboard and monitor (or terminal), one or two disk drives, and 48K or 64K of memory. These computers were not designed to be compatible with each other, but CP/M took care of that.

The BIOS

Built into CP/M is a library of subprograms for performing such tasks as printing a character to the screen. Each computer might use a different kind of video display, so some portions of CP/M, the BIOS (Basic Input/Output Subsystem), were customized for each machine, but BIOS acted the same way on every machine. Because of the BIOS, programmers could write their routines to use these universal subprograms instead of directly programming their particular computer's video chip. The program, if written properly, could run on any computer with CP/M. Machine-specific tasks became standardized routines.

A CP/M software market thrived, since developers could write a single program that would run on many different computers. Woe be to the computer that lacked CP/M. Even though the TRS-80 used a Z80, it took the efforts of third-party developers to bring CP/M to this machine. For awhile, TRS-80 owners were isolated from the mass market, with a separate, smaller library of software. CP/M was the leader of the eight-bit world, and most small businesses used Z80 CP/M computers. CP/M machines occupied the niche that the IBM PC and PC clones control today.

The Debate over Obsolescence

The boom went to bust with the introduction of the IBM PC. CP/M machines just couldn't keep up with advances in hardware and software. Although the IBM PC was not a real breakthrough, it expanded the memory ceiling from 64K to 640K. Disk storage jumped from 100K to as much as 370K (double-sided disks). The faster and more powerful 8088 microprocessor made it easier to write better programs in less time. IBM's open architecture encouraged additional power as more and more hardware companies enhanced the IBM with add-ons.

The microprocessor used in the IBM could not run CP/M, so a whole new standard was forged. (Digital Research's CP/M-86 was not available in time for the release of the PC, so it failed to establish itself as a standard. Microsoft's MS-DOS, which is much like CP/M, beat out CP/M-86, not because it was better, but because it was first.) The 8-bit Z80 world of CP/M was replaced by IBM's 16-bit 8088 world. Software developers jumped on the bandwagon, and CP/M was put on the back burner.

Since CP/M is no longer the dominant environment for high-end microcomputing (although CP/M machines are still selling today), you may wonder why it is an issue on Commodore machines. It would seem the best bet would be an IBM MS-DOS emulator, with an 8088 instead of a Z80. Commodore probably went with CP/M because it is built around cheap, proven technology. The Z80 simply costs less than the 8088. And CP/M is more generalized and easier to adapt than the MS-DOS used on IBM PCs. CP/M may be Commodore's way of crossing over from home computing to small business computing. Commodore is even translating some IBM software to CP/M, taking advantage of the similarities between CP/M and MS-DOS.

Most CP/M programs are written in 8080 or Z80 machine language. CP/M takes care of the minor differences between Z80 machines, but you still have to have a Z80 microprocessor. CP/M could be translated to run on any computer, such as the 6502, but what good is a 6502 version of CP/M if all the programs that run under CP/M are written in Z80 machine language?

CP/M Plus

Digital Research has developed an enhanced version of CP/M 3.0 for the Commodore 128. This CP/M takes advantage of the VIC-chip graphics, color, 80-column RGB, and SID chip available to the 128 in CP/M mode. Unlike CP/M for the 64 (which uses the 6502 for machine-specific tasks, while the Z80 runs the bulk of CP/M), Commodore 128 CP/M runs solely on the Z80. Commodore's ingenious memory management unit (MMU) allows the Z80 full access to 128K and the graphics and sound chips. Programs written especially for Commodore CP/M Plus could really shine. Few CP/M computers in the price range of the 128 can do color graphics and sound synthesis. And Commodore has indicated that there are a few veteran CP/M software developers that are quite interested in a fresh market for their wares.

A Library of Programs

Since the Z80 is always in control, this allows the 128 to run off-the-shelf CP/M programs. Although the programs won't take advantage of any special 128 features, these plain vanilla programs will work just fine.

Getting these programs into memory is another story. Most CP/M disks are read and written to with the IBM System/34 format. This format is not compatible with the 1541 disk drive. So even though the program would run, you cannot load it into memory with the 1541 disk drive. Commodore has converted a tiny amount of CP/M programs to 1541 format, including the programming languages FORTRAN and COBOL, and the Perfect productivity series. These programs will work on both 64 CP/M and 128 CP/M Plus with the 1541. But these few packages are a far cry from the promise of thousands of programs.

Commodore's new 1571 disk drive solves the problem. The 1571 runs about ten times faster than the 1541 and can store 410K in the CP/M mode. It can use 1541 disks and behave like a 1541 when necessary. The 1571 can also be reprogrammed to read and write several common CP/M disk formats, including the disk format of the Osborne and Kaypro portable CP/M machines. So you can theoretically insert any CP/M disk, turn on the power, and the program will load and run. Even though you may be able to load the program into memory, not all programs strictly follow the BIOS guidelines. Some programs are optimized for a particular CP/M computer. It can be painfully slow to use the BIOS to fill the screen a character at a time, so some programs prefer to be machine-specific for the sake of speed or to take advantage of special machine features. These programs will not necessarily run on Commodore CP/M Plus. But there are still thousands of programs, many in the public domain, that will run just fine.

Why bother with CP/M at all? There are many good CP/M word processors, but there are several word processors for the 64 mode that are every bit as good. Enterprising programmers will surely write sophisticated word processors and business software to take advantage of 80 columns and 128K.

There's much more business software available to CP/M machines, but most home computerists won't really want to run an accounts receivable program. When the 64 was first introduced, CP/M looked like an excellent way to get around the paucity of available software, but now there are almost too many 64 programs to choose from. Does anyone really need CP/M on the 128?

A Business Bargain

Many people would say yes. CP/M may make the Commodore 128 a bargain buy for small businesses. No longer are the low-end Commodore machines restricted by a slow disk drive and small memory size. The price of the Commodore 128 with the 1571 disk drive is quite competitive with the IBM PCjr, which is now no longer being produced anyway. CP/M software has been around long enough to be time-tested and bug-free. There's so much CP/M software that there's a good chance you'll find special-interest programs—those that wouldn't have mass appeal, but could be just what you're looking for. For example, some programs are customized for particular businesses, such as a bookkeeping system designed especially for a dental practice. You'll likely find special-interest programs for the home, such as a database that helps you track your family's roots.

CP/M promises a cornucopia of software. Some of this software may be useful to you, but unless you're in business, most of it probably won't be. It remains to be seen, though, with all the technological advances in hardware and software, if anyone still wants to run five-year old software.

2
Programming

2

Debugging BASIC Programs

Todd Heimarck

Program bugs have a thousand faces. No matter how experienced a programmer might be, there's almost always a time between finishing and really completing a program—debugging time. In this article, we'll see what the computer does when you make a mistake, and we'll look at some useful debugging methods as well as some of the mistakes a computer can make.

Some program bugs are easy to recognize: The program crashes and you are told what kind of error you made. Or worse, the computer locks up. These are the deadly, or *fatal,* bugs.

Other bugs are sneakier and not as easy to recognize. Perhaps you've made a slight mistake and the program seems to run, but is actually making incorrect calculations (like figuring interest rates on a 13-month year). Subtle bugs are sometimes worse than fatal ones; at least you can recognize something's wrong when the program crashes.

A bug happens when the programmer says one thing and the computer either doesn't know how to do it or does something very different.

Some people say that computers never make mistakes, that all bugs are caused by people. That's not always true. After all, computers are designed and manufactured by people who can make mistakes. It may be a hardware bug or one built into the operating system. Or maybe the programmer just didn't understand how the computer would interpret a line—a misunderstanding rather than a mistake. But a computer is pigheaded. It knows how programs should look and won't compromise. First, though, let's concentrate on some of the mistakes we, as programmers, can make and how the computer deals with them.

SYNTAX ERROR

There are over two dozen error messages, but SYNTAX ERROR seems to appear most often. The line number is always included. The first thing to do, of course, is to LIST the offending line. Take a good look at it. If there are parentheses, make sure they match up. There should be an equal number of opening and closing parentheses.

Also, check all the BASIC keywords to make sure they're spelled correctly. You may have mistakenly abbreviated LEN as L SHIFT-E (which turns out to be the abbreviation for LET, not LEN). If you are writing to disk or tape files, you should note that the command PRINT# is distinct from the ordinary PRINT. The abbreviation for PRINT# is P SHIFT-R (not ?#); using a question mark won't work when you're working with tape or disk files.

Look at the punctuation, a common source of errors. It's easy to accidentally type a period instead of a comma, a semicolon where you meant to put a colon. Or there may be mismatched opening and closing quotation marks in a PRINT statement. If you're copying a program from a book or magazine, look closely at the look-alike characters (*I* and *1*, *O* and *0*).

Tokens, Keywords, and Reserved Variables

When you type a program line and press RETURN, you must have at least one BASIC *keyword* (command or function) in the line for it to be legal. Even the do-nothing REM is a BASIC keyword. The computer reads the line from the screen and turns all keywords into tokens before the line is stored in memory. A token is a single number between 128 and 255 which represents the command or function.

You must avoid including keywords in variable names. Perhaps you're writing a simple accounting program which figures out the profit margin you make on different items. You need a variable for the price you pay (call it COST), another variable for the amount for which you sell it (call it LIST, for list price), and one more for the profit (MONEY). Then you calculate the margin with the formula MONEY = LIST − COST. Right? Wrong.

All three variables are illegal and will crash the program. LIST is a keyword used to list a program; you can't use it as a variable. COST doesn't look like a keyword, but the first three letters spell COS; your computer will try to find a cosine of an angle, although it will stop when it can't find parentheses and an argument. And MONEY is a problem because the keyword ON (as in ON-GOTO or ON-GOSUB) is embedded in the variable name. When you find an embedded keyword in one part of a program, there's a good chance the same variable is used elsewhere, in other sections. You'll have to find all of them and change them to something legal.

In addition to the many keywords, there are three reserved variables which you can't use in your programs. They are TI, TI$, and ST. The first two are used for timekeeping, and ST is short for STatus, which is used in input/ouput operations. Stay away from these variable names, unless you know you want to check the time or status. You can't define TI or ST, although you can print them to the screen and use them in IF-THEN statements and logical operations. You *can* define TI$, which is useful for timing programs, but it has to be a six-digit string (for example, TI$= "103000" sets the clock to 10:30 a.m.).

A Commodore computer is a little more forgiving with the reserved variable names than with keywords. You can't use a variable TIPS because TI is included in the first two letters of the name. But you *can* use a variable name like ITIN, which has a TI in the middle, because only the first two letters of a variable name count. ITON, on the other hand, is not acceptable as a variable name, because there are two BASIC words in it (TO and ON), and BASIC words cannot appear *anywhere* in a variable name.

Program Glue

Need a program line inserted between 10 and 20? No problem. Type a line 15 and press RETURN; the computer automatically inserts the line in its proper position. LIST 10–20 will prove that the line is there where you wanted it.

It's almost as if the computer broke the program in two and pasted the line in its proper place. But this cut-and-paste feature, usually quite handy, can become a curse which results in two kinds of program bugs.

The first bug, truncated lines, is relatively easy to find. It usually occurs when keywords are abbreviated. If, for example, you use a question mark (?) instead of PRINT, or P SHIFT-O instead of POKE, you can create logical lines which are legal when they're entered, but exceed the limit when listed. Later editing of the line leads to problems.

If you use abbreviations and multistatement lines, the result is sometimes a line which looks longer than should be possible when listed. A question mark takes up only one space on a line, but LIST detokenizes and changes that single letter into five: PRINT. List such a line and you may see two full screen lines plus a few characters on the third line. And the program runs without errors. But go back to edit the extra-long line, press RETURN, and the input buffer will read only the first two screen lines into memory. The result is truncated—or chopped off—program lines. You lose the last few characters. To get around the limit, you have two choices. Either retype the keywords using abbreviations or break the long line into two shorter lines.

This limit on line length means it's a good idea to press RETURN only when you're editing a line. To move around the screen, use the cursor keys (or SHIFT–RETURN, which does not enter the line in program memory and is also a way to get out of quote mode).

The second bug, which is more difficult to find, happens when your computer seems to glue two program lines together. Say, you're writing a program using 40 columns and the line is 40 characters long. You type the line, but forget to press RETURN. The cursor is positioned at the beginning of a screen line, so you type the next line and press RETURN. The computer treats the two lines as one because it has received only one RETURN.

Some Other Common Errors

POKEs and SYSes can wreak havoc if improperly used. Most lockups are caused by one or the other of these powerful commands. When you're debugging, watch for transposed or missing digits in POKEs and SYSes (POKE 53820 instead of POKE 53280, or SYS 59152 instead of SYS 49152, for example).

Duplicate variable names can cause all sorts of problems. You might use a variable called A to hold a value at the beginning of the program, and then inadvertently use the same variable name later on. If the program returns to the beginning, the value has changed. FOR-NEXT loops sometimes lead to duplication. When you're using a variable like A, make sure you don't use it as an index in a FOR-NEXT loop. And remember, only the first two characters of a variable count; the computer thinks ALT is the same variable name as ALIEN.

To avoid doubled variables, it helps to pick certain letters to be used only in loops and as "temporary" variables. For example, decide ahead of time that you will always use J, K, and L in FOR-NEXT loops.

Be careful with additional statements after an IF-THEN. If the condition (between IF and THEN) is *not* true, the program jumps to the next BASIC line; it doesn't fall through to the next colon. For example, in this line:

```
55 IF A=1 THEN B=15: PRINT "NEXT
   QUESTION?" :INPUT Q
```

the PRINT and INPUT statements will happen only if A equals 1. If not, everything after the THEN is ignored. This feature is useful if you want multiple actions under certain conditions. But it can catch you if you don't know about it.

The error message RETURN WITHOUT GOSUB is usually the consequence of the common practice of putting subroutines at the end of a program. The computer finishes the main routine and continues through to the first subroutine until it reaches the RETURN statement. The quick fix is to place an END statement between the main routine and the first subroutine. For example, if subroutines begin at line 5000, add a line 4999 END.

NEXT WITHOUT FOR, an infrequent error message, generally comes from improper nesting of loops. Loops are like onions: You can build layers which completely enclose other layers. In other words, the first loop to begin has to be the last to end.

```
1 FOR J=1TO5: FOR K= 3TO15: NEXT K: NEXT J
2 FOR J=1TO5: FOR K= 3TO15: NEXT J: NEXT K
```

Line 1 is correct because the K loop is inside the J loop. But line 2 spells trouble because loops cannot overlap.

The use of arrays can lead to easily rectified errors. It's best if you DIMension all arrays at the beginning of a program or in a one-shot subroutine. Once you use DIM, you can't use it again on the same array name or you'll get a REDIM'D ARRAY error.

Order of Operation

Most of the mistakes described above will cause your program to stop with an error message on the screen of your TV or monitor. They're situations where you tell the computer to do something and it doesn't recognize what you want. Program-crashing errors are inconvenient, of course. But it's nice to have the computer tell you what kind of mistake you made and which line was wrong.

Less convenient are errors of procedure, where you write a program to do one thing, but it ends up doing something completely different. It doesn't crash, but it does strange things to the screen or gives seemingly impossible results.

You have probably used instructions which you interpret one way, but the computer interprets another. And you can't change the way your computer does things, unless you want to completely rewrite the operating system (even changing the rules of BASIC means you have to follow the rules of machine language). Some programmers wish they could have a new BASIC command DWIM (Do What I Mean), which would instantly straighten out procedural errors.

One of the most common problems with mathematical calculations comes from the way the computer evaluates equations. There is a definite order of operations, sometimes called the hierarchy of operators (the items at the top of the list have a higher priority):

( )	Parentheses
↑	Exponentiation (up arrow)
+ −	Positive and negative signs
* /	Multiply and divide
+ −	Add and subtract
=	Equals (assignment)
= < >	Comparisons: Equals, less than, greater than
NOT	Logical NOT
AND	Logical AND
OR	Logical OR

Note that some operations, like NOT, work on a single number; they're called *unary*. Most need two numbers and are called *binary* functions. Plus and minus signs can be either unary (in the number −3, the minus sign works on a single number) or binary (the minus sign connects two numbers in the expression 10 − 6).

Because the higher operations are calculated first, you can always figure out the results of an equation. For example, J = 4 + 5 * 3 assigns 19 to J because the multiplication is done first, binary addition second, and assignment-equals third.

When you're debugging a program and one of the variables is being consistently miscalculated, there's a good chance you're a victim of the hierarchy. The quickest way to fix such an error is to liberally sprinkle parentheses throughout the suspicious equations. Your other choice is to trace through the line step by step to find how the computer is evaluating the equation.

There's a slight chance that using too many parentheses in debugging can lead to one of the more puzzling errors, stack problems, caused by one of the various limits you have to live with.

Memory Limits: The Stack

There are two causes of OUT OF MEMORY errors. The first is programs and variables filling up all available BASIC RAM. The second is a stack overflow (the likely cause of OUT OF MEMORY errors).

Let's look at the stack first. The *stack* is a special section of memory just above zero page. It takes up most of page 1. The stack is used by the operating system for notes to itself.

When BASIC begins a FOR-NEXT loop, it writes a note about where in memory the loop begins, pushes it on the stack, and forgets about it until it comes across a NEXT statement. NEXT tells the computer that somewhere earlier in the program a FOR started a loop. It then pulls the information it needs off the stack and jumps back.

Something similar happens when there are parentheses in an equation and when you use GOSUB. To illustrate, type NEW and try running the following program:

```
10 A=A+1: PRINTA
20 GOSUB 10
```

A very short program with only one variable counts up to 24 and then crashes. How could it possibly run out of memory? The key is the GOSUB. Every time you go to a subroutine, the return address is saved on the stack. Since there are no RETURNs in the program, more and more addresses are saved, until finally there is no stack space left.

Type NEW and enter this program:

```
10 FORA=1TO20
20 FORB=1TO20
30 FORC=1TO20
(and so on, up to 130 FORM=1TO20)
```

Don't worry about adding any NEXTs, the computer will never get that far. Run the program and you'll get an OUT OF MEMORY error after only ten loops have begun. A FOR-NEXT loop uses up a lot of space on the stack—for a pointer to beginning of the loop, step size, highest value, and variable names.

When stack problems pop up, they're often caused by a GOTO in the middle of a subroutine. It can leave some garbage on the stack. The same goes for jumping out of a FOR-NEXT loop. And too many parentheses can give you either a FORMULA TOO COMPLEX error or contribute to an OUT OF MEMORY message. As the garbage on the stack builds up, it eventually reaches the limit.

Programs and Variables

The other way to run out of memory is fairly straightforward. You simply use too much BASIC RAM for the program and its variables. Try the following program:

```
10 T$="ABCD":U$=""
20 L=FRE(0):IFL<0THENL=L+2↑16
30 L=INT((L-30)/3):DIM A$(L)
40 FORJ=1TOL
50 A$(J)=T$+U$
60 PRINT J; LEN(A$(J)), FRE(0)
70 NEXT J
```

You'll run out of memory almost right away. Now change line 50:

50 A$(J)=T$

Run it again and there's no loss of memory. It will run all the way through (press RUN/STOP if you don't want to watch hundreds of strings go by). The first program wasn't able to create even ten four-letter strings; the second created hundreds. The only difference is that the first program added a null string (which has a length of zero); the second did not. The first created *dynamic strings;* the second created *static strings.*

If you define a string by concatenating (adding two strings together), by dissection (dividing a string with MID$, LEFT$, or RIGHT$), or by inputting it (from a tape or disk file, or from the keyboard), the string is called dynamic. It has to use up part of BASIC memory. If you define it in BASIC, assigning it (A$="ABC") or reading it from DATA statements (READA$), the computer saves memory by remembering where the definition was in program memory. Your computer doesn't have to use free memory to store static strings. They're already in BASIC memory.

If you define a lot of variables (as in the above program), available memory can dwindle to nothing. When you find your program running out of memory, you can try a number of things:

1. **Check free memory**. If there seems to be a lot left, you may have a full stack, caused by too many unresolved FOR-NEXT loops or GOSUBs.
2. **Eliminate unnecessary program lines, especially REMarks**. Or combine two or more statements on a single line separated by colons (every line uses five bytes for overhead, whether it has one statement or eight).
3. **Cut back on variables**. If you're using arrays, remember that integer arrays use less than half the space of floating-point arrays.
4. **Completely rewrite your program**. It sounds drastic, but once you've figured out the procedures you're using, the second version of a program is often faster and uses less memory.
5. **Try chaining programs**. If you have a lot of instructions in a game program, you can write a loader program which prints the instructions and then loads the main program.

We've covered some of the limits which affect memory and the stack. Variables, too, have limits. They can lead to a variety of problems. You can employ three types of variables in a program: string, floating-point, and integer. Certain restrictions apply to each of the three.

Precision, Accuracy, Magnitude

Floating-point (FP) numbers, so-called because the decimal point can "float" to either end of the number, use up five bytes of memory. The variable name needs two additional bytes, so an FP variable fits into seven bytes of memory.

Three limits apply to floating-point numbers: precision, accuracy, and magnitude. Floating-point numbers are allowed up to nine digits of precision. Go beyond nine, and your computer automatically rounds to the nearest nine-digit number.

The following program illustrates the limits of precision:

```
10 A$="1":B$=A$
20 FOR J=1TO20
30 A$=A$+B$: PRINTA$,VAL(A$)
40 NEXT
```

Note that we're working with strings, which can be longer than nine characters. But in line 30, the strings are converted to a VALue, which succumbs to the nine-digit limit. After the loop runs nine times, we see the letter *E*, which represents exponentiation (for example, 10 to the power of *X*). We've hit one of the limits. You can make calculations on large numbers, but they will be rounded to the nearest nine digits of significance.

Another limit, accuracy, sounds like it might be the same as precision, but it's not. Limits on accuracy are built into almost any numbering system.

Computers calculate in binary (base two). Fractions which can be expressed as a combination of halves, fourths, eighths, sixteenths, and so on, are accurate. Others have to be rounded to the nearest binary value.

People do the same thing with decimal fractions. The number 1/3 is translated to a never-ending series of threes, 0.3333333....

The limits on accuracy can sometimes lead to errors of rounding. Try the following program:

```
10 X=.1
20 FORJ=0TO50:Y=Y+X:PRINTY:NEXT
```

A couple dozen times through the loop and the answers start to vary from what they should be. The number in computer memory is just about 1/10, but is a little off. It's only an approximation. As the numbers add up, so does the slight inaccuracy.

Magnitude is the final limit. It's the culprit in OVERFLOW errors. The operating system stores floating-point numbers in five bytes. What happens when all the bytes fill up? The number is a little beyond 10 to the thirty-eighth power, a one followed by 38 zeros; the computer cannot count any higher.

You can force an OVERFLOW error with this program:

```
10 X=10: FORJ=1TO50: PRINTJ,X: X=X*10: NEXT
```

The program stops when the computer reaches a number beyond which it cannot count. Change X=10 to X=−10 to find the limit on the negative side.

How do these limits affect BASIC programs? Precision is not really a problem, unless you want to count past a billion. If you sacrifice precision, you can count a little beyond a billion billion billion billion before reaching the highest number allowed. Accuracy can adversely affect a lot of programs, however. In a financial program, for example, you might add and subtract some numbers, ending up with a number like $517.120001 or $517.119999 instead of $517.12. Such programs should include a rounding function, DEF FN R(X) = INT(X*100+.5)/100 to strip off those extra numbers.

Integer Limits

Integer variables have their own limits. Integer variables are always whole numbers and are signified in programs by a percentage sign (%) suffix. A%, B%, and Y8% are some examples. You can also use them in arrays—A5%(6), YZ%(15), P%(0), and so on.

Magnitude, rarely a problem with FP numbers, can be a serious limit on integers. Integers are stored in only two bytes. The highest integer allowed is 32767, the lowest is −32768.

Accuracy is never a problem with integers, and the limits of precision never become a problem, either.

String Limits

Strings, collections of characters, are subject to only two limits, both related to length.

First, when INPUT, a string cannot exceed 80 characters (two screen lines worth). Second, strings cannot be more than 255 characters long. Concatenation (or adding together two strings) allows strings to exceed the input limit. This program demonstrates:

```
10 A$="Z"
20 FORJ=1TO400: B$=B$+A$: PRINTJ,B$: NEXT
```

The string variable B$ is not initialized and so begins as a *null string* (a string containing nothing) with a length of zero. Each trip through the loop adds the variable A$, which holds the single letter Z. As B$ grows larger and larger, it reaches the limit of 255 characters, and the computer prints an error message.

File Errors

Sequential disk files operate much the same as tape files. You begin with the first item and continue until you reach the last. Reading and writing these files can lead to a variety of errors—some subtle, others not so subtle.

There are two commands for reading (INPUT# and GET#) and one for writing (PRINT#). (Note that there is no space before the number sign.) These three BASIC keywords differ from the usual INPUT, GET, and PRINT. If you abbreviate, don't use ?# for PRINT#; it won't work. P SHIFT-R is the correct short form for PRINT#.

If you open a file for reading and try to write to it, or vice versa, you'll get a NOT INPUT FILE or NOT OUTPUT FILE error. If your disk drive is not plugged into the serial port or is not turned on, the computer will tell you DEVICE NOT PRESENT. If you press PLAY on a Datassette (to load a program), and leave it on PLAY, then try to write a file, it will seem to work, but the file isn't actually there. There's a sensor that can tell if a button is pressed, but it doesn't distinguish between PLAY, PLAY/RECORD, or even F.FWD or REWIND. Writing a file while PLAY (but not RECORD) is pressed won't write anything.

You can close a file which is already closed, but you can't open a file which is already open. To be safe, you can precede an OPEN with a CLOSE. For example, CLOSE2: OPEN2,8,2, "filename,S,W" will make sure the file is closed before it is opened.

If you don't close a file before ending the program, you can run into big problems. A disk drive has its own microprocessor, which keeps track of open files. If you open a file, write to it, and turn off your computer without closing the file, the result is a "poison" file, which can corrupt other files on the disk. Poison files are marked in the directory with an asterisk. You should never scratch a poison file; you have to use the Validate command to get rid of it. Before you end a program, be certain to close all files.

STRING TOO LONG

A very common file error is STRING TOO LONG, mentioned above. For strings in a file which are longer than 80 characters, you'll have to use GET# rather than INPUT#. GET# reads in characters one by one. INPUT# bites off a chunk at a time. In many cases, GET# is more reliable than INPUT#.

Another mistake you can make is writing a file of strings and then trying to read back numeric variables (for example, PRINT#1,A$ to write the file followed by INPUT#1,A when reading it).

Checking Variables

Now let's see how you can track down and eliminate program bugs. When you type RUN, all variables are cleared. Variable values then build up as the program runs.

If the program stops, the variables are still intact, but you lose them the moment you change a line or add a new one. Even if you simply press RETURN over a line, making no changes, you'll lose all variable values, until the program is run again.

Let's imagine a program which stops in the middle and says ILLEGAL QUANTITY IN 300. The first thing to do is type LIST300. You might then see something like this:

```
300 FOR A= S TO E: READ B: POKE A,B: CK% = CK% + B
    : NEXT
```

One of those variables holds an illegal quantity of some kind. Type PRINT B to discover the value of B. If it's greater than 255 or less than 0, B is the culprit. When you POKE a number into memory, it has to be between 0 and 255. If B is 519, for example, the program will crash. In this case, the number is coming from a DATA statement. Maybe you left out a comma, or two lines got stuck together when you forgot to press RETURN after a line. Whatever the cause, you'll have to find the incorrect DATA statement.

Testing variables can help you find a good number of bugs, especially when you have duplicated variable names (for example, using the name J in two different sections of a program). But remember, as soon as you press RETURN over a line, all variables will be lost.

If you want to rerun a program and still preserve the current variable values, you can choose a line number (call it *xxx)* and type GOTO*xxx*, as long as you haven't pressed RETURN over a line. GOTO does not destroy variable values as RUN does.

Simplify and Isolate

The most elusive bugs are the ones which don't happen right away. Rather, they appear after the program has run 20 or 30 times, seemingly without flaw. Just when you thought it was all finished, the program crashes—or locks up.

You must simplify and isolate, find the one situation that causes the problems. If possible, try to duplicate the error. If you know what happens just before a crash, you're halfway to finding the bug.

Besides PRINT (to check variables), there are four BASIC commands which are great aids when you're hunting down an elusive bug: STOP, CONT, REM, and GET.

Perhaps you've narrowed it down to a certain FOR-NEXT loop. An important variable, K8, is somehow being changed. So you add a line PRINT K8:STOP, and every time the program reaches that line, it prints the value of K8 and stops.

If you want to continue, type CONT. These two commands work in tandem, one stopping the program, the other starting it up again. While the program is temporarily stopped, you can examine any other variables you want, using PRINT.

STOP Radar

STOP can also be used as a pointer. Start with a 100-line program with a bug (in this example, let's assume it's straightforward and doesn't use any subroutines). The first line is 10, the last 1000, in increments of 10. Put a STOP halfway through the program, just before line 500. Run it, and it crashes before it even reaches line 500. You now know the problem—or at least one of them—happens somewhere in the first half of the program. Now put a STOP in line 250. This time the program stops, but not because of an error. You type CONT (for CONTinue), and again the computer freezes before getting to 500. With just a couple of lines, you've zeroed in on the general area of the bug. It's after 250, but before 500. A couple more STOPs and you can narrow the possibilities to just a few lines. STOP is like radar used to pinpoint the bug.

Now you suspect the bug is in a certain line. But you don't know for sure. The line does some calculations followed by a POKE or two. You can make the line invisible with a REMark. REM is generally used to add comments because it makes the computer ignore everything up to the next line. But it's also good for temporarily removing a line, so the line, as usual, is ignored.

Finally, GET can sometimes substitute for the STOP-CONT debugging duo. If you'd rather halt the program temporarily instead of stopping it, add a line XXX GET G$: IF G$ =""THEN XXX. Whenever the line is executed, everything pauses until you press a key.

Timeout to Clean the Blackboard

Have you ever written a program which usually runs well, but sometimes pauses before starting up again? You don't have a bug. You can put the blame on a process called *garbage collection*, especially if the program contains a lot of string variables.

As variables are defined, they are put into memory just after the end of the program. But strings can contain 1 letter or 5 or 160.

Say, your program has a variable A$ and you define it, A$ = "HELLO, " + N$ (where N$ is a person's name). You've created a *dynamic string*. Later on, the program changes A$ to "HELLO AGAIN, " + N$. One way to store this new string on the memory blackboard would be to erase the old one and put this one in its place. But the new A$ is longer, so the computer would have to move a lot of memory around to make room. Instead, the computer marks the old variable as "garbage," drawing an imaginary line through it, and puts the new variable into an empty space.

But if memory fills up completely (from all the garbage strings), it's time to get rid of all the strings no longer being used. And that takes time. To illustrate, look at this program:

```
10 DIMA$(255)
20 FORX=1TO255: B=INT(RND(1)*26+65)
30 B$=CHR$(B): A$(X)=A$(X-1)+B$: PRINTB$
40 NEXT:GOTO20
```

Enter it and type RUN. It takes some time before available string memory fills with garbage. But eventually, you'll see the program pause while it frees up some space. There's nothing wrong with the computer, it's doing just what it's supposed to.

The process of garbage collection is another quirk of the operating system. Asking the computer how much free memory is left—using FRE(0)—forces garbage collection, so you can force it to occur when it matters least.

Lockup Bugs

If your computer locks up, consider the possibility that your computer is *not* locked up. A FOR-NEXT loop that counts to a million takes a lot of time. So does POKEing a few thousand numbers into memory. And it's possible to write an inefficient sorting routine that takes hours, even days, to complete. In cases like these, you might want to demonstrate that there's no lockup by printing to the screen or changing border color once in a while.

Hardware Errors

Hardware should be the last thing you blame. If something is not going right in a program, it's almost always the program's fault.

Nevertheless, hardware (especially moving parts as in a disk drive or printer) occasionally has problems. After many hours of use, disk drives can become misaligned; they'll read disks they've written to, but not disks formatted on other drives (commercial software, for example). And the head on a cassette drive can become dirty or magnetized.

Two rare bugs you may encounter involve disk access. The first is a documented problem with relative files. If you read a short record from a file that begins on a sector boundary and then later read a subsequent file that is longer than the first and spans two sectors, the second read may be corrupted because a pointer is not updated. The solution is to set the record pointer before *and* after reading a file.

The second is undocumented; it's one of those full-moon bugs. The disk SAVE WITH REPLACE option works almost as it should. It scratches the old program and saves a good version of the replacement program. But it may corrupt another file on the disk, especially if the disk is almost full. So far, it has not been proved without a doubt that on a 1541 SAVE WITH REPLACE (SAVE"@:filename") is flawed. In fact, there are two people who have offered a reward to the person who proves the bug exists.

Nevertheless, hardware rarely causes problems, although sometimes a memory chip burns out or a soldered connection breaks. Generally, if your computer works for a day or two after you buy it, it will work for years.

MLX and Proofreader

The two COMPUTE! typing aids, "MLX" and "Automatic Proofreader," help immensely. But they can miss transposition errors.

Both programs work by adding up numbers. MLX, used for entering machine language programs, adds six numbers (plus the memory location). So you could type 000, 000, 000, 000, 013, 015 to get a total of 28. But 000, 000, 000, 000, 015, 013 also add up to 28. MLX wouldn't know the difference. The checksum matches, but the numbers are wrong. Unfortunately, machine language is extremely sensitive to incorrect numbers and there could be big problems with the program.

BASIC is more forgiving than machine language—it usually tells you the type of error and the line number. The Proofreader is also forgiving. It adds up the ASCII values of the line and calculates the checksum. So if you type PRINT+AB, rather than PRINTA+B, the Proofreader checksum number will come out fine. PRITN is a small problem, because it causes a SYNTAX ERROR. But a POKE with transposed numbers can lead to trouble, 132 instead of 123, for example.

Foolproof INPUT

Charles Brannon

This program overcomes some of the problems of the INPUT statement. It's a short machine language routine that requires no special knowledge of machine language. Easy to use, it reprograms BASIC's own INPUT routine.

Problems with INPUT

You are probably familiar with some of the problems with the INPUT statement. First, it will not properly handle input with commas and colons. If you entered the previous sentence, the computer would accept only the word *First* and would ignore the rest of the line (as the computer warns you with ?EXTRA IGNORED). This is because the comma is used to separate multiple INPUTs on the same line, as in this example:

INPUT "ENTER NAME:FIRST,LAST";A$,B$

The colon, too, triggers an ?EXTRA IGNORED message. Yet it cannot be used to separate INPUT items, so it appears to be some kind of bug (error) in the BASIC language itself.

You can get around these problems somewhat, but they become especially annoying when you are trying to read a file on tape or disk that contains these characters. In a mailing-list program, for instance, you need commas for address fields such as "Greensboro, NC, 27403".

There are other difficulties with the INPUT statement as well. Quotation marks are not handled correctly. Leading and trailing spaces are stripped away. INPUT also allows people to use all the cursor and color control keys. Theoretically, you can place the cursor anywhere on the screen where there is something you want to INPUT, and press RETURN. In effect, this is what happens when you edit a program (the same INPUT routine is used by both the system and BASIC). But it just makes no sense to allow cursor moves all over the screen when you simply want the user to answer a question. If the user accidentally presses a cursor key and then tries to move the cursor back, the entire line, including any prompts, is read.

This can also be a problem when you have carefully laid out a screen format with blanks or boxes into which a user is supposed to enter information. You have no way to control the number of characters that a user can type, so if your blank space is only ten characters long, there is nothing to prevent someone from typing more. Not only that, but also with the standard INPUT routine, someone can move the cursor out of the box you want to be used, clear the screen entirely, or otherwise destroy your carefully planned screen format.

Improving on INPUT

What we need, then, is a new INPUT routine that will not allow cursor moves. The INST/DEL key should still let the user delete characters to make corrections, however. Additionally, the ideal INPUT routine should let your program limit the number of characters typed, and allow commas and colons.

The usual solution is to write your own INPUT routine using the GET statement, which fetches one key at a time from the keyboard. With such a simple statement as GET, however, you have to reinvent the wheel anytime you need such a protected INPUT routine. And it certainly isn't as easy to use as a simple INPUT statement.

Well, it certainly wouldn't be fair to bring such gloom to the scene without presenting a solution. The accompanying program is the key. It's a machine language routine that replaces the standard Commodore INPUT with a protected INPUT such as described above. The beauty of it is that after you GOSUB 60000, all INPUT (and INPUT#) statements are redefined. You don't have to understand how the machine language works in order to use it, and you don't have to rewrite any existing programs other than to insert the GOSUB. You still have all the flexibility of the standard INPUT statement. Just add the subroutine to the end of your program.

The machine language program has a couple of niceties. After you GOSUB 60000, you can change the maximum number of characters allowed by POKEing memory location 252 with the length (don't POKE with 0 or more than 88). The cursor is an underline by default, but you can change the character used for the cursor by POKEing the ASCII value of the character you want into memory location 2. For example, to change the cursor into an asterisk, enter

POKE 2,ASC("*")

When you use the routine to INPUT data from files, just remember that it strips away all the control characters from CHR$(0) to CHR$(31) and from CHR$(128) to CHR$(159). This includes all special codes such as cursor controls, function keys, color codes, and so on. You'll rarely write these to a standard file anyway.

You may be intrigued to find that this special INPUT routine even works in direct mode. You can still LIST and RUN, but cursor controls remain disabled. Just press RUN/STOP–RESTORE if you want the special INPUT routine out of your way.

Foolproof INPUT

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
60000 IF PEEK(830)=133 THEN 60020               :rem 145
60010 FORI=828TO977:READA:POKEI,A:NEXT          :rem 127
60020 SYS 828:RETURN                            :rem 179
60030 DATA 169,000,133,252,169,080              :rem 135
60040 DATA 133,251,169,164,133,002              :rem 131
60050 DATA 169,083,141,036,003,169              :rem 142
60060 DATA 003,141,037,003,096,152              :rem 127
60070 DATA 072,138,072,165,252,208              :rem 144
60080 DATA 007,032,116,003,169,000              :rem 123
60090 DATA 133,253,166,253,189,000              :rem 143
60100 DATA 002,133,254,198,252,230              :rem 129
60110 DATA 253,104,170,104,168,165              :rem 133
60120 DATA 254,096,160,000,132,252              :rem 127
60130 DATA 165,002,032,210,255,169              :rem 130
60140 DATA 157,032,210,255,032,228              :rem 131
60150 DATA 255,240,251,164,252,133              :rem 135
60160 DATA 254,169,032,032,210,255              :rem 135
60170 DATA 169,157,032,210,255,165              :rem 145
60180 DATA 254,201,013,240,043,201              :rem 119
60190 DATA 020,208,013,192,000,240              :rem 120
60200 DATA 211,136,169,157,032,210              :rem 129
60210 DATA 255,076,118,003,041,127              :rem 132
60220 DATA 201,032,144,196,196,251              :rem 137
60230 DATA 240,192,165,254,153,000              :rem 131
60240 DATA 002,032,210,255,169,000              :rem 120
60250 DATA 133,212,200,076,118,003              :rem 123
60260 DATA 230,252,153,000,002,169              :rem 125
60270 DATA 032,032,210,255,096,013              :rem 129
```

Ultrasort

John W. Ross

This is one of the fastest sorting programs ever published for any home computer. It will alphabetize 1000 items in less than eight seconds. The test generates random "words" so you can see how the program works.

Sorting programs written in BASIC are generally acceptably fast for short lists. One method for sorting is the Shell sort, which is actually quite efficient, certainly far better than a bubble sort, for instance. Nevertheless, there are better sorts.

C.A.R. Hoarse's Quicksort algorithm, is possibly the fastest yet developed for most applications. So, here's a machine language sort program based on the Quicksort algorithm.

Speed Improvements

In order to test the program, I wrote a small sort test program (Program 2). This program generates a character array containing *N* items (line 110). Different items are generated depending on the value of the random number seed, SD in line 140; SD must be a negative number.

To test the sort, we generated six 1000-element arrays and sorted them using both the "Super Shell Sort" (a previously published sorting program) and "Ultrasort." Super Shell Sort required an average of 29.60 seconds to sort all 1000 elements, while Ultrasort required an average of only 8.32 seconds. The sorting time increased 72 percent. You probably won't find a faster sort for an eight-bit machine anywhere.

To run the sort, use:

SYS 49152,N,AA$(K)

Running the Program

Ultrasort can be used either from within a program or in immediate mode. Running Ultrasort causes *N* elements from array AA$, starting with element K, to be sorted into ascending order. The sort occurs in place; there is not additional memory overhead. Elements N and K can be constants or variables, and any character array name can be substituted for AA$.

Before running the sort, it must be loaded by BASIC. The appropriate loader is supplied in Program 1. The tradeoff for the increased speed of Ultrasort is increased complexity, especially in machine language. The increased size, of course, creates a greater possibility of errors when you enter the numbers. Make sure you read and use "Automatic Proofreader" (Appendix C). Save a copy of the program before you run it. You must use the BASIC loader before running the sort program.

Program 2 demonstrates how fast Ultrasort is. To watch the demonstration, load and run Ultrasort. Then load and run "Sort Test" (Program 2). One hundred random strings will be created, and after you press a key, they will be sorted and listed. The time required for the sort is displayed at the end of the list. To change the number of strings created, change the variable N in line 110 of Program 2.

Program 1. Ultrasort

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
10 I=49152                                      :rem 236
20 READA:IFA=256THENEND                         :rem 169
30 POKEI,A:I=I+1:GOTO20                         :rem 130
49152 DATA76,100,192,170,170,170,170             :rem 33
49159 DATA170,170,170,170,170,170,170            :rem 86
49166 DATA170,170,170,170,170,170,170            :rem 84
49173 DATA170,170,170,170,170,170,170            :rem 82
49180 DATA170,170,170,170,170,170,170            :rem 80
49187 DATA170,170,170,170,170,170,170            :rem 87
49194 DATA170,170,170,170,170,170,170            :rem 85
49201 DATA170,170,170,170,170,170,170            :rem 74
49208 DATA170,170,170,170,170,170,170            :rem 81
49215 DATA170,170,170,170,170,170,170            :rem 79
49222 DATA170,170,170,170,170,170,170            :rem 77
49229 DATA170,170,170,170,170,170,170            :rem 84
49236 DATA170,170,170,170,170,170,170            :rem 82
49243 DATA170,170,170,170,170,170,170            :rem 80
49250 DATA170,170,32,253,174,32,158             :rem 244
49257 DATA173,32,247,183,165,20,141             :rem 250
49264 DATA12,192,165,21,141,13,192              :rem 191
49271 DATA32,253,174,32,158,173,56              :rem 205
49278 DATA165,71,233,3,133,75,165               :rem 158
49285 DATA72,233,0,133,76,162,1                  :rem 45
49292 DATA173,12,192,157,20,192,173             :rem 252
49299 DATA13,192,157,40,192,169,1               :rem 161
49306 DATA157,60,192,169,0,157,80               :rem 156
49313 DATA192,189,60,192,141,16,192             :rem 255
```

```
49320 DATA189,80,192,141,17,192,189      :rem 6
49327 DATA20,192,141,18,192,189,40      :rem 202
49334 DATA192,141,19,192,32,47,195      :rem 208
49341 DATA173,11,192,48,4,202,208       :rem 142
49348 DATA221,96,189,60,192,141,16      :rem 211
49355 DATA192,189,80,192,141,17,192       :rem 8
49362 DATA169,1,141,18,192,169,0        :rem 102
49369 DATA141,19,192,32,101,195,189       :rem 5
49376 DATA20,192,141,18,192,141,14      :rem 195
49383 DATA192,189,40,192,141,19,192       :rem 7
49390 DATA141,15,192,32,47,195,173      :rem 205
49397 DATA11,192,48,3,76,167,193        :rem 119
49404 DATA32,131,195,173,16,192,141     :rem 244
49411 DATA3,192,173,17,192,141,4         :rem 93
49418 DATA192,173,14,192,141,5,192      :rem 203
49425 DATA173,15,192,141,6,192,32       :rem 148
49432 DATA132,194,32,180,194,173,11     :rem 245
49439 DATA192,48,218,173,16,192,141       :rem 6
49446 DATA3,192,173,17,192,141,4        :rem 101
49453 DATA192,173,18,192,141,16,192       :rem 0
49460 DATA173,19,192,141,17,192,169       :rem 4
49467 DATA1,141,18,192,169,0,141         :rem 98
49474 DATA19,192,32,101,195,173,16      :rem 204
49481 DATA192,141,18,192,173,17,192       :rem 2
49488 DATA141,19,192,173,3,192,141      :rem 207
49495 DATA16,192,173,4,192,141,17       :rem 157
49502 DATA192,32,47,195,173,11,192      :rem 202
49509 DATA16,35,173,14,192,141,3         :rem 97
49516 DATA192,173,15,192,141,4,192      :rem 202
49523 DATA173,18,192,141,5,192,173      :rem 203
49530 DATA19,192,141,6,192,32,132       :rem 144
49537 DATA194,32,180,194,173,11,192       :rem 1
49544 DATA48,152,32,47,195,173,11       :rem 156
49551 DATA192,16,18,173,16,192,141      :rem 202
49558 DATA3,192,173,17,192,141,4        :rem 105
49565 DATA192,32,132,194,32,31,195      :rem 204
49572 DATA76,241,192,234,189,20,192       :rem 6
49579 DATA141,3,192,189,40,192,141      :rem 209
49586 DATA4,192,173,16,192,141,5        :rem 107
49593 DATA192,173,17,192,141,6,192      :rem 211
49600 DATA32,132,194,32,31,195,173      :rem 193
49607 DATA16,192,141,18,192,141,3       :rem 147
49614 DATA192,173,17,192,141,19,192       :rem 1
49621 DATA141,4,192,32,81,195,189       :rem 157
49628 DATA20,192,141,18,192,189,40      :rem 206
49635 DATA192,141,19,192,32,101,195     :rem 251
49642 DATA173,11,192,48,15,189,60       :rem 158
49649 DATA192,141,18,192,189,80,192      :rem 15
49656 DATA141,19,192,32,101,195,169       :rem 2
49663 DATA1,141,18,192,169,0,141         :rem 96
```

```
49670 DATA19,192,173,3,192,141,16          :rem 153
49677 DATA192,173,4,192,141,17,192         :rem 212
49684 DATA173,11,192,16,52,189,60          :rem 160
49691 DATA192,232,157,60,192,202,189        :rem 55
49698 DATA80,192,232,157,80,192,32         :rem 215
49705 DATA101,195,173,16,192,157,20        :rem 249
49712 DATA192,173,17,192,157,40,192          :rem 1
49719 DATA32,131,195,32,131,195,202        :rem 246
49726 DATA173,16,192,157,60,192,173          :rem 6
49733 DATA17,192,157,80,192,76,128         :rem 217
49740 DATA194,32,131,195,232,173,16        :rem 250
49747 DATA192,157,60,192,173,17,192         :rem 11
49754 DATA157,80,192,202,189,20,192          :rem 4
49761 DATA232,157,20,192,202,189,40        :rem 249
49768 DATA192,232,157,40,192,202,32        :rem 253
49775 DATA101,195,32,101,195,173,16        :rem 251
49782 DATA192,157,20,192,173,17,192          :rem 6
49789 DATA157,40,192,232,76,162,192         :rem 13
49796 DATA160,3,165,75,133,79,133          :rem 165
49803 DATA81,165,76,133,80,133,82          :rem 156
49810 DATA24,165,79,109,3,192,133          :rem 154
49817 DATA79,165,80,109,4,192,133          :rem 164
49824 DATA80,24,165,81,109,5,192           :rem 107
49831 DATA133,81,165,82,109,6,192          :rem 157
49838 DATA133,82,136,208,223,96,160          :rem 4
49845 DATA0,140,11,192,177,79,141          :rem 152
49852 DATA7,192,177,81,141,8,192           :rem 115
49859 DATA200,152,205,7,192,240,2          :rem 145
49866 DATA176,13,205,8,192,240,21          :rem 153
49873 DATA144,19,238,11,192,76,30          :rem 159
49880 DATA195,205,8,192,240,2,176          :rem 159
49887 DATA62,206,11,192,76,30,195          :rem 163
49894 DATA140,9,192,160,1,177,79           :rem 117
49901 DATA133,77,200,177,79,133,78         :rem 213
49908 DATA172,9,192,136,177,77,141         :rem 220
49915 DATA10,192,140,9,192,160,1            :rem 93
49922 DATA177,81,133,77,200,177,81         :rem 211
49929 DATA133,78,172,9,192,177,77          :rem 181
49936 DATA200,205,10,192,208,3,76          :rem 145
49943 DATA195,194,144,184,76,224,194        :rem 69
49950 DATA96,160,2,177,79,72,177           :rem 124
49957 DATA81,145,79,104,145,81,136         :rem 217
49964 DATA16,243,96,169,0,141,11           :rem 105
49971 DATA192,173,17,192,205,19,192          :rem 8
49978 DATA144,6,240,8,238,11,192           :rem 111
49985 DATA96,206,11,192,96,173,16          :rem 171
49992 DATA192,205,18,192,144,244,208        :rem 56
49999 DATA238,96,173,16,192,24,109         :rem 228
50006 DATA18,192,141,16,192,173,17         :rem 190
50013 DATA192,109,19,192,141,17,192        :rem 241
```

```
50020 DATA96,169,0,141,11,192,56                 :rem 87
50027 DATA173,16,192,237,18,192,141              :rem 245
50034 DATA16,192,173,17,192,237,19               :rem 198
50041 DATA192,141,17,192,176,3,206               :rem 187
50048 DATA11,192,96,238,16,192,208               :rem 202
50055 DATA3,238,17,192,96,170,170                :rem 148
50062 DATA170,170,170,170,170,170,170            :rem 71
50069 DATA170,170,170,170,170,170,170            :rem 78
50076 DATA170,170,170,170,170,170,170            :rem 76
50083 DATA170,170,170,170,170,170,170            :rem 74
50090 DATA170,170,170,170,170,170,170            :rem 72
50097 DATA170,170,170,170,170,170,170            :rem 79
50104 DATA170,170,170,170,170,170,170            :rem 68
50111 DATA170,170,170,170,170,81,85              :rem 232
50118 DATA73,67,75,83,79,82,84                   :rem 21
50125 DATA32,76,79,65,42,32,32                   :rem 252
50132 DATA3,255,50,48,44,82,69                   :rem 254
50139 DATA65,68,32,69,82,82,79                   :rem 21
50146 DATA82,44,49,56,44,48,48                   :rem 12
50153 DATA0,170,170,170,170,81,85                :rem 134
50160 DATA73,67,75,83,79,82,84                   :rem 18
50167 DATA32,76,79,65,68,69,82                   :rem 25
50174 DATA16,255,256                             :rem 19
```

Program 2. Sort Test

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
100 PRINT"{CLR}"                                 :rem 245
110 N=100                                        :rem 174
120 DIM AA$(N)                                   :rem 178
130 PRINT"CREATING"N" RANDOM STRINGS"            :rem 47
140 SD=-TI:A=RND(SD)                             :rem 183
150 FOR I=1 TO N                                 :rem 37
160 PRINT I"{UP}"                                :rem 66
170 N1=INT(RND(1)*10+1)                          :rem 221
180 A$=""                                        :rem 127
190 FOR J=1 TO N1                                :rem 91
200 B$=CHR$(INT(RND(1)*26+65))                   :rem 81
210 A$=A$+B$                                     :rem 43
220 NEXT J                                       :rem 29
230 AA$(I)=A$                                    :rem 119
240 NEXTI                                        :rem 30
250 PRINT "HIT ANY KEY TO START SORT"            :rem 151
260 GET A$:IF A$="" THEN 260                     :rem 83
270 PRINT "SORTING..."                           :rem 26
280 T1=TI                                        :rem 249
290 SYS 49152,N,AA$(1)                           :rem 109
300 T2=TI                                        :rem 243
```

```
310 PRINT "DONE"                               :rem 139
320 PRINT "HIT ANY KEY TO PRINT SORTED STRINGS"
                                                :rem 71
330 GET A$:IF A$="" THEN 330                   :rem 79
340 FOR I=1 TO N:PRINT I,AA$(I):NEXT           :rem 27
350 PRINT:PRINT N" ELEMENTS SORTED IN"(T2-T1)/60"S
    ECONDS"                                    :rem 180
```

Writing Text Adventures

Gary McGath

Programming text adventure games, those popular interactive games where you communicate to the computer through words, is an art in itself. It's not quite the same as creating an arcade-style videogame. Here, Gary McGath, who has written a book on just this subject, explains some of the basics of writing text adventures.

A text adventure is an interactive computer game in which the player assumes the role of a character in a story. As the player, you control the character's actions by typing in commands, and the computer responds with a text description of what your character experiences.

The world of most text adventures is composed of a number of *rooms*, or locations. Your character moves from place to place, or from room to room, where objects or other characters may or may not be found. Sometimes these objects and characters aid you, other times they're dangerous. By using the appropriate commands, you can pick up, examine, and even use these objects and characters.

While professionally written adventure programs often comprehend complicated sentences as commands, many adventures get by with simple two-word commands. The vocabulary of even the best of them is quite limited, and they have to indicate to you whether they "understand" any particular command.

The following dialogue is typical of a text adventure. (Your commands are printed in boldface and the computer's messages in regular text.)

You are in a small room lined with shelves. There are doors to the north and west.
There is a gem on the shelf.

Take gem

Your hand is stopped by an invisible shield around the gem.

Examine shield

I don't know the word *shield.*

North

You are in a north-south hallway.

Writing a text adventure offers you a chance to exercise your imagination and set up logical puzzles for your friends. It requires no special screen formatting or sound effects, and the program is doing nothing between moves; these facts make text adventure programs easy to debug. And once you've written your first adventure, you can do more of them just by changing the rooms and puzzles in your old program.

Mapmaking

While the first steps in designing a text adventure are to create the *story line* (what will happen) and the *milieu* (where things will happen), we'll assume you've already done that. In this article, we'll be concerned mainly with the actual programming techniques you'll use, as well as some of the more practical design processes.

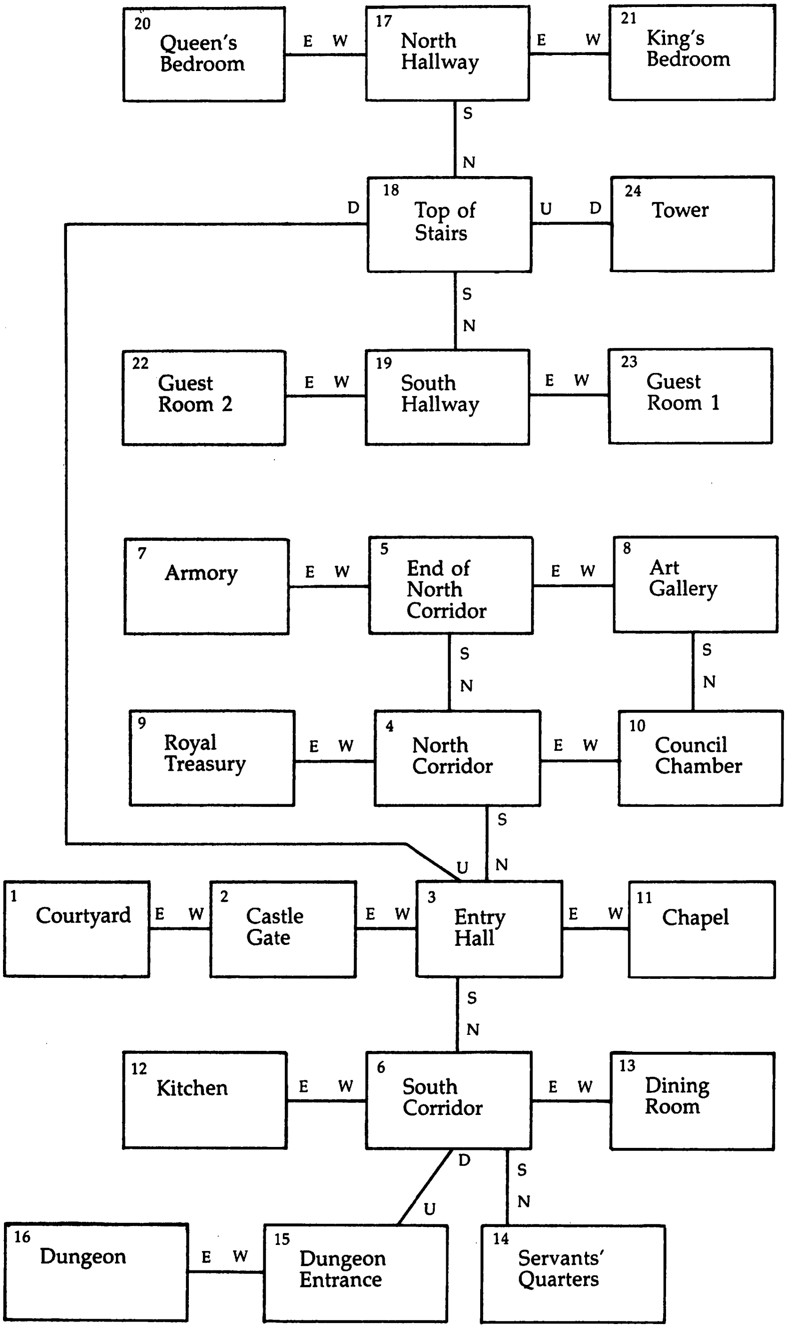
Once you've decided what your world is, and what will happen in it, you need to design a map of the rooms. (Remember that they don't have to actually be *rooms*—we're using that as a generic term. They can be places on a road, paths in a forest, or even corners of a field.) Draw a map with a box for each room and connecting lines labeled with the directions that lead from one room to another (north or south, for instance). Give each room a number and a short description. The room in which the character starts should be room 1. Figure 1 shows the map of an example text adventure game.

Objects, Verbs, and Consequences

In this planning stage, you also need to make several other decisions. Choose the objects that will be in the adventure and determine where each will be initially located. Some objects might not be in any room at all until the player does something to make them appear. You should also assign numbers to the objects.

Your program also needs a list of the verbs that will be accepted as commands. Certain verbs (or words that function as verbs) are almost mandatory, such as NORTH, SOUTH, EAST, WEST, TAKE, DROP, EXAMINE, LOOK, INVENTORY, and QUIT. Other verbs that might be helpful include ENTER, CLIMB, SHAKE, MOVE, TURN, FIGHT, OPEN, EAT, DRINK,

Figure 1. The Adventure's Map



CLOSE, and READ. Abbreviations, such as I for INVENTORY and N for NORTH, are easier for the player to remember and use. Allowing the use of equivalent alternatives, like GET and TAKE—which should mean the same thing—can reduce player frustration. Remember, the difficulties in an adventure should come from the logical puzzles, *not* from figuring out how to talk to the program.

What consequences do specific actions have? Will opening a box reveal a gem, or will it set off an explosion? Will pressing a switch start a machine? Will magic words transport the character into a new room? Consequences could include appearances and disappearances, changes in the character's abilities, alteration of the paths between rooms, and transportation from one location to another.

Some actions may have special consequences only under restricted circumstances. A special tool may be needed, such as a crowbar to open a crate. If this tool isn't in the character's inventory, the action won't have the desired effect and might even backfire.

Things may happen independently of the player's actions as well. A troll might be wandering around the adventure's world. Or the character's lamp might go out after a certain number of moves.

When you've considered all these things and made your choices, you know what you want the adventure to do. Only now should you worry about the details of the program. As you discover what's easy to program and what isn't, you might change your mind about which features to include. But just as when you program any game, you should start with an overall plan. It will save you countless hours of wasted time later on.

Assigning Variables

Now you're ready to actually begin programming your text adventure game. We'll go through the process step by step, outlining and illustrating exactly how to do it.

The first step is to to assign variables to the important parameters of the adventure. It's easier to remember what these variables mean than it is to recall a number; using these variables also makes it simple to alter the program if you later decide to change the parameters. One of the first statements of the program, even before the DIM statements, should look something like this:

```
10 NR=21:NV=14:NO=16:NI=10:ND=6
```

NR is the number of rooms, NV the number of verbs, NO the number of objects, NI the number of items, and ND the number of different directions the character can move in. (Note: An *object* is any word that can be used as the second word of a command, whether it corresponds to a physical object or not. An *item* is an object that is located in a room; it usually designates a physical object.)

Adventure Arrays

The next step is to translate the layout of your adventure into a set of data structures. Let's look at each of the required structures and the purpose it serves.

Access array. This is the translation of your map into terms the computer can understand. It's defined by the statement

DIM AC(NR,ND)

To use the access array (AC), the directions in which the character moves must be translated into numbers. Let's assume the following translation:

North	=	1	South	=	2
East	=	3	West	=	4
Up	=	5	Down	=	6

The value of AC(NR,ND) specifies which room is reached by going in direction ND from room NR. If this value is 0, it means the character can't go that way from that room.

Room description array. This array is defined by

DIM RD$(NR)

Each of its entries is a string that gives the description of the room—for example, "You are standing on a wide bridge."

Room flag array. Flags are indicators of whether a condition is true or false. The array is defined by

DIM RF(NR)

To conserve memory, all the flags for a room are stored as one value. The different flags are defined as powers of 2—1, 2, 4, 8, 16, and so on. A value of 1 might indicate that the room is too cold, 2 that magic works, and 4 that water is present. The value of RF(R) for room R consists of the sum of all the flag values that are true for that room. If a room is cold and allows magic, but doesn't have any water, then its entry in the array would be a 3 (1 + 2). Flag F for room R can be tested with the following line:

IF (RF(R) AND F) <> 0 THEN PRINT"FLAG"F"IS TRUE."

Verb array. This is an array of the possible first words of commands, defined by

DIM VB$(NV)

You should decide how many letters in a word are going to be significant and chop the verbs in this array down to that size. For instance, if two letters are significant, then TAKE must be stored as TA. It's a good idea to limit the number of significant letters so that two-fingered typists have less work to do. Many simple adventure games designate only two letters as significant.

Object array. This is an array of the possible second words of commands (objects), defined by

DIM OB$(NO)

Once again, all words in this array should contain only as many letters as are significant.

Verb token array. This serves to translate verbs into numbers. It is dimensioned by

DIM VT(NV)

The entries in this array correspond to entries in the verb array. The values stored consist of numbers from 1 to the *number of distinguishable verbs* in the game. This number is normally smaller than NV, since similar verbs such as GET and TAKE, or NORTH and N, are not distinguishable. If VB$(2)=N and VB$(3)=NORTH, then VT(2) and VT(3) will have the same value. This lets the program be indifferent to the word that was actually typed.

Object token array. This array translates the second word of a command into a number. It is defined by

DIM OT(NO)

Its elements correspond to the object array. However, the elements can be a little trickier than the verb token array's elements. Remember that not all *objects* are *items.* It's convenient to have the object tokens fall into two series. Items, which are objects that have a particular location, can be numbered from 1 to NI. Other objects, including directions and magic words, can be numbered starting with 101. This makes it easy to add new items without disrupting your numbering system.

Item description array. This contains a text description for each item. Its definition is

DIM ID$(NI)

The text description of an item could be the same as the word in the object array for it, but often it's a little more. For instance, the object array might have the word *LAMP* for an object described in the item description array as "Old oil lamp."

Item location array. This locates each item; it is defined by

DIM IL(NI)

There are three possibilities for where an item is located. It could be in a room, in the character's inventory, or nowhere at all. The third case indicates an item that's been destroyed or one that's not yet available. A positive number in the item location array indicates which room the item is in. A zero says that the character is carrying the item. A negative one specifies that the item isn't to be found.

Item flag array. This is similar to the room flag array in concept, except that it specifies conditions that are true or false of items rather than rooms. It is defined by

DIM FI(NI)

(It would make sense to call the array IF, but that's a reserved word in BASIC.) The flags are used to indicate such properties as whether the item can be carried or not.

More Variables

Finally, you'll need to set a few more variables, for example:

VB Verb token obtained from the last command entered.

OB Object token obtained from the last command. It can be 0 if only one word was typed.

RM Room the character occupies.

IC Number of items the character is carrying.

MI Maximum number of items the character may carry. IC may never exceed MI.

MC Move counter. This indicates how many moves have occurred since the adventure started. It can serve as a timer for various events.

DF Description request flag. This variable is set to 0 after the current room is described to the player. If a description is required before the next move (because the character went into a new room or decided to LOOK around again), it's set to 1 to get the description displayed. Leaving it at 0 saves having the same description repeated every move.

Specific situations will undoubtedly call for a few more variables, but the arrays and variables listed here will provide the major part of what a simple adventure needs.

The Main Loop

An adventure program consists of two parts: the initialization and the main loop. The initialization section includes dimensioning arrays and setting up data. We've already looked at some of the initialization section of our example adventure. It uses READ and DATA statements to set up all the initial values. Once the initialization is done, however, the main loop takes over. It runs until the game is completed. The overall flow of the main loop would be something similar to that shown in Figure 2.

The major portions of the main loop, as shown in Figure 2, are the room description, the automatic routines, the command INPUT and parsing, and the action routines. Let's consider how to program each of these in turn.

Room Description

Whenever the character moves into a new room, the surroundings change. If the player asks to LOOK at the room again, the room description routine provides this information. There are two things to be described: the room itself and whatever items it contains.

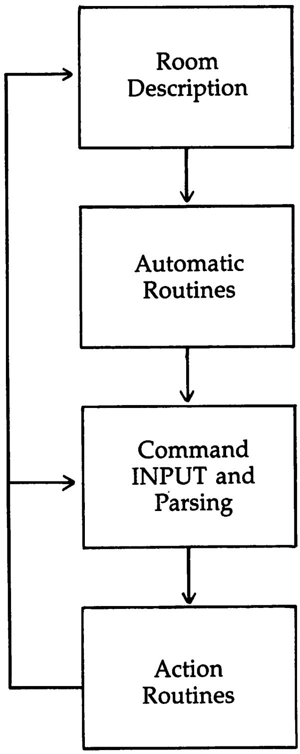
This routine isn't long and could look like the lines below *(the sections of BASIC presented here are brief examples that illustrate the concepts at hand, but do not add up to a complete working program)*:

```
400 IF DF=0 THEN 600
410 PRINT RD$(RM)
420 F=0
430 FOR I=1 TO NI
```

```
440 IF IL(I)<>RM THEN 490
450 IF F=0 THEN PRINT "YOU SEE:":F=1
460 PRINT ID$(I)
490 NEXT I
```

The description request flag in line 400 determines whether this section of the program is executed or skipped over. Remember that 0 indicates the latter. If it is 0, then, this entire routine is bypassed. If it *is* executed, describing the room consists simply of printing the appropriate element of the room descripion array. That's line 410. Then in line 430, a FOR-NEXT loop executes, which goes through each item in the item location array. For each item that's located in the current room (F=0), it prints the corresponding element of the item description array (done in line 450 and 460). This way the player will see what each room contains.

Figure 2. The Main Loop



Automatic Routines

The next section of the main loop takes care of events that aren't directly caused by the player's commands. We can call these routines *automatic*, for they happen independently of what's typed in. An adventure can be written without any automatic routines, but having even a few things outside the player's control gives a much greater sense of realism and excitement.

Automatic routines can be controlled by the move counter, random numbers, or a combination of the two. The commands the player gives can have an effect as well. A passage may close four turns after the character enters a room, or a wraith may start stalking the character only after a crypt has been touched. Extra variables can be used to indicate the move on which something will happen. In the following example routine, MM is a variable indicating the move in which a wall collapses, opening a new passage between rooms 8 and 9.

```
700 MC=MC+1
710 IF MC<>MM THEN 800
720 AC(8,3) = 9: AC(9,4) = 8
730 IF RM=8 THEN PRINT "THE EASTERN";
740 IF RM=9 THEN PRINT "THE WESTERN";
750 IF RM=8 OR RM=9 THEN PRINT " WALL COLLAPSES, O
    PENING A NEW PASSAGE."
```

MC is the move counter, our timer, so to speak. Each time through the main loop, it's incremented by 1 in line 700. Assuming we earlier set MM to the desired turn number (say, 8), then this automatic routine would not be executed until MC equals MM—in other words, on turn 8. Line 710 insures this. Line 720 actually creates the opening between the rooms. The message then displays, specifying which wall has crumbled. If the character is in room 8, for instance, the eastern wall has fallen, and the character can now move in that direction.

The position of automatic routines in the program is important. Usually, they should come *after* the room description so that the player finds out where the character is before being told what happens. Some automatic routines, however, are better placed after the player has completed the move . This conveys the feeling that what happened immediately followed the move. For instance, if a flock of bats carries the character out of a room every time he or she tries to enter, the player may not even see the room until it's discovered how to get the bats out.

Command INPUT and Parsing

At this point the program stops talking to the player; instead, it's the player's turn to communicate with the program. To do this, the program must accept a command and parse it. To *parse* a command simply means to break it up into its components and identify their relationships—an easy job when it consists of just two words.

Here's the first section of an INPUT and parsing routine:

```
1000 INPUT C$
1010 L=LEN(C$):IF L=0 THEN 1000
1020 C1$="":C2$="":C2=0:X=0
1030 FOR I=1 TO L
1040 A$=MID$(C$,I,1)
1050 IF A$<>" " THEN 1080
1060 IF C2$<>"" THEN 1200
1070 X=1:GOTO 1090
1080 IF X=0 THEN C1$=C1$+A$:GOTO 1090
1085 C2$=C2$+A$
1090 NEXT I
```

The program receives a command through the INPUT statement. As the player enters words, a string is created. Then the program separates the two words by looking for one or more spaces between them. (It's best that it be tolerant of more than one space between words as well as spaces after the command. INPUT automatically strips leading spaces, so they don't pose a problem.) The above program section receives the player's INPUT (line 1000) and creates two strings, C1$ and C2$ (lines 1080 and 1085). Spaces between words are also checked for in line 1050.

The following lines continue the routine:

```
1200 C1$=LEFT$(C1$,6): C2$=LEFT$(C2$,6)
1210 FOR I=1 TO NV
1220 IF VB$(I)=C1$ THEN VB=VT(I):GOTO 1250
1230 NEXT I
1240 PRINT "I DON'T KNOW THE VERB ";C1$:GOTO 1000
1250 IF C2$="" THEN OB=0:GOTO 1400
1255 FOR I=1 TO NO
1260 IF OB$(I)=C2$ THEN OB=OT(I):GOTO 1400
1270 NEXT I
1280 PRINT "I DON'T KNOW THE OBJECT ";C2$:GOTO 100
     0
```

The two strings, C1$ and C2$, are the first and second words of the command. The next step is to translate these strings into the verb token and the object token. This means looking them up in the verb array and object array and getting the corresponding elements of the verb token array and object token array. Lines 1220 and 1260 in the section of the routine above do this for the verb and object respectively. Note the checks and messages displayed if the verb and/or object do not exist in the appropriate array.

The two strings must be truncated to the number of significant characters in order to match the strings in the arrays. Line 1200 assumes truncation to six characters.

In the case of a one-word command, C2$ will be the empty string, so the object token will be set to 0 (line 1250).

Action Routines

Once the program has the command in the form of the verb token and the object token, it's ready to determine what those commands will do. We can call the parts of the program that do this the *action routines.* This section will be the largest portion of the program; however, since it consists of many small pieces, it isn't very difficult to write.

Before figuring out what a specific verb does, the program should do some general checking to determine whether the object is reasonable. If the object is an item, it has to be either in the room or in the character's inventory. If it's somewhere else, the character can't do anything with it. If the object isn't an item, then only a few verbs will work with it, so the program should make sure that the verb is an appropriate one. NORTH, for example, isn't something the character can TAKE, EAT, or OPEN. Only GO makes sense.

The following routine assumes that the direction object tokens (NORTH, UP, and so forth) are numbers 101 to 106, that GO is verb 10, that SHAZAM is object 107, and that SAY is verb 12.

(In a language that was more generous with names than BASIC, we could assign a variable name to each verb. Trying to think of a two-letter name for each verb that would mean anything, though, is a hopeless exercise. So at this point we resign ourselves to using numbers.)

```
1400 IF OB<100 THEN 1600
1405 REM IT'S NOT AN ITEM
1410 IF OB<=106 AND VB<>10 THEN 8000
1420 IF OB=107 AND VB<>12 THEN 8000
1430 GOTO 2000
1599 REM IT IS AN ITEM
1600 IF IL(OB)<>RM AND IL(OB)<>0 THEN PRINT "IT IS
     N'T HERE.":GOTO 1000
8000 PRINT "THAT'S SILLY!":GOTO 1000
```

Line 1400 of the routine checks to see if it's an item (with an object token less than 100). If it is, the program jumps to line 1600, where it's determined whether the item is in the room or in the character's inventory. If neither, then the message IT ISN'T HERE displays. The program chides the player with THAT'S SILLY! if a direction (NORTH, UP, and so forth) is requested and GO isn't used with it. The player will also see the message if the object is SHAZAM and the verb is not SAY (line 1420).

Notice that if the command is rejected, the program goes back to the command INPUT (through the GOTO 1000 statements in lines 1600 and 8000) rather than letting anything happen automatically.

If these checks turn up no problems, the program falls through to the action routine for the specific verb. The tool used is the GOTO statement found in line 1430 above. It sends the program to line 2000, shown below:

```
2000 ON VB GOTO 3000,3100,3200,3300,3400,3500,3600
     ,3700
2010 ON VB-8 GOTO 3800,3900,4000,4100,4200,4300,44
     00,4500
```

Several of these statements will usually be necessary because of the 80-character line limitation. Remember that an ON statement will simply fall through to the next statement if the variable is out of range. Thus, if the variable is 9, it falls through line 2000 to line 2010, where it would access the first line listed, 3800 (9 − 8 = 1). Using this technique, we can call up to 16 different verb routines in the above example.

Each of the line numbers in lines 2000 and 2010 is the start of the action routine for a particular verb.

Certain verbs will be standard in most adventures, so they can be discussed in some detail here. Others will have effects that are peculiar to the situation. They're the ones that make your adventure unique. Once you've seen how the standard verbs work, though, you shouldn't have much trouble adding your own special ones.

Directional verbs and GO. There are two ways a player might specify moving in a direction: Either a simple direction (for instance, EAST or just E) or GO and a direction (GO EAST) could be entered. It isn't much trouble to include both. A common area of the program can be used to handle all directional movement, using a direction variable that the specific commands set before accessing the actual movement.

For a one-word command, the direction acts as the verb. In this case, it just sets the direction variable and goes to the common routine. The line below illustrates the one-word command NORTH.

```
3100 D=1:GOTO 3620
```

You'll recall that earlier we decided to use 1 as the directional number for NORTH. All that's done in the above line is to set D (the directional variable) to 1 and then GOTO a line that checks to see if that direction leads anywhere. (More on that in a bit.)

However, the GO command has to translate its object into a direction before going to the common routine. It's easy to do this if the direction objects are numbered appropriately so that subtracting a number from the object token gives the right index into the access array. Take a look at the following lines:

```
3700 IF OB<=100 OR OB>106 THEN 8000
3710 D=OB-100:GOTO 3620
```

Notice that if the object (OB) is *not* a direction (checked for in line 3700), then the program jumps to line 8000, where the message THAT'S SILLY! is printed. The direction variable D is set in line 3710. If OB equals 101, for instance, signifying that the direction is NORTH, then D equals 1. The program then moves to line 3620.

The common routine uses the access array to determine where the move will take the character. The next segment is this common routine used by both one- and two-word commands.

```
3620 IF AC(RM,D)=0 THEN PRINT "YOU CAN'T GO THAT W
     AY.":GOTO 400
3630 RM=AC(RM,D):DF=1:GOTO 400
```

A value of 0, as mentioned before, means that a given direction doesn't lead anywhere. If the command does take the character somewhere, the description request flag is also set so that the player can see the new room. Both of the lines above take the program back to the routine that describes the room.

TAKE. This command transfers, or attempts to transfer, an item from the current room to the character's inventory. The program has to determine whether the item can be picked up and whether it can be carried. The character might already be carrying as much as allowable. Taking an item might also have side effects, for instance, making another item visible or setting off a trap. The program doesn't have to check whether the object is in the room since that has already been determined. However, it *does* have to check whether the character is already carrying the item. Take a look at the lines below to see how that can be programmed:

```
4200 IF (FI(OB) AND CF)=0 THEN PRINT "YOU CAN'T PI
     CK THAT UP.":GOTO 400
4210 IF IL(OB)=0 THEN PRINT "YOU ALREADY HAVE IT!"
     :GOTO 400
4220 IF IC=5 THEN PRINT "YOU'RE CARRYING TOO MUCH
     {SPACE}ALREADY.":GOTO 400
4230 IL(OB)=0:IC=IC+1:PRINT "TAKEN."
4240 REM SIDE EFFECTS GO HERE
4290 GOTO 400
```

This assumes that flag CF (in line 4200) in the item flag array indicates whether or not an item can be taken. If your character already has the item, then line 4210 prints a message to that effect. Note that a limit of five items is set in line 4220. If IC (the variable keeping track of the numbers of items carried) equals 5, the character can't take anything else. Line 4230 actually TAKEs the item by placing it in the character's inventory (IL(OB)=0), increments the number of items held, and prints a message that the TAKE was successful.

DROP. The reverse of TAKE, it's even simpler, since an item that is being carried can normally be dropped.

```
4300 IF OB=0 THEN PRINT "DROP WHAT?":GOTO 1000
4310 IF IL(OB)<>0 THEN PRINT "YOU DON'T HAVE IT!":
     GOTO 400
```

```
4320 IL(OB)=RM:PRINT "DROPPED."
4330 IC=IC-1
4390 GOTO 400
```

The only question is if the item is in the character's inventory; this is checked in line 4310. The object is transferred to the room (line 4320) and the inventory count is decremented (line 4330). Again, side effects are possible.

INVENTORY. All this command does is list the items the character is carrying. This involves going through all the items and listing the ones that have a location of 0.

```
4400 PRINT "YOU ARE CARRYING:"
4410 FOR I=1 TO NI
4420 IF IL(I)=0 THEN PRINT ID$(I)
4430 NEXT I
4440 IF IC=0 THEN PRINT "NOTHING."
4450 GOTO 400
```

Line 4420 PRINTs the items the character is carrying. If IC (the number of items carried) is 0, a message indicating that the character holds nothing is displayed.

LOOK. This is one of the simplest commands; it just sets the description request flag with a line such as

```
4500 DF=1:GOTO 400
```

QUIT. Even simpler, except that it's nice to make sure the player really means it:

```
4600 PRINT "DO YOU REALLY WANT TO QUIT";
4610 INPUT Y$
4620 IF LEFT$(Y$,1)<>"Y" THEN 1000
4630 END
```

Unusual Commands

Other verbs vary from one adventure game to another. EXAMINE can give you additional information about items. FEEL, SMELL, and TOUCH might serve a similar purpose. The process of examination might also cause other, previously hidden, items to appear. OPEN could be another way to reveal a hidden item. Words like CUT and BURN might have interesting effects on items, but unless an appropriate tool is in the character's inventory, these commands would simply return a message like "You can't do that."

Having a few commands that do nothing but return a standard response is useful, just because you can increase the number of commands that get an interesting answer without adding much to the programming effort. For instance, the verb BREAK with any object might get the response "Vandalism won't help your situation." This will also leave the player wondering whether there's some object that *could* be broken for a useful result.

Commands like CLIMB or ENTER might work on certain objects to provide a way of getting from one room to another, in addition to the directional commands. (Don't use GO for this, please. In spite of what some adventure game programmers think, you don't "go a door.")

Other commands might also surprise the player by transporting the character from one place to another. For instance, taking an item might cause a trap door to open, dropping the character into the room below. Magic words can serve this purpose. A magic word may be restricted in its use to a certain room, so it provides passage only from that room to another.

What Goes into It?

The mechanics of writing an adventure program are only part of the job, just as grammar and spelling are only part of what goes into writing. The other part is what you actually have to say. Creating the content of an adventure can't be reduced to a cookbook approach. Still, some general guidelines are possible.

Quests and hunts. There are two basic types of adventure: the quest and the treasure hunt. In a quest adventure, you're given a particular goal to achieve, such as solving a mystery or obtaining a single treasure. In a treasure hunt, you're trying to find as many treasures as possible to get a high score.

The quest adventure is an all-or-nothing proposition. The program can give you a score to indicate how close you've come to success, but you probably won't be satisfied until you solve it. The treasure hunt offers more satisfaction to the beginning adventurer since, if even a few treasures are found, there's a sense of accomplishment. If a quest is like climbing a mountain, a treasure hunt can be compared with hiking across a series of low hills. Each one has its own kind of satisfaction.

Make the pieces fit. In either case, all the pieces should fit together. This is more obvious for a quest—each step is part of a developing story. Even in a treasure hunt, though, everything should be set against a common background and story line. If your setting is the world of Greek mythology, Wotan and Brunhilde shouldn't appear without good reason. If you've chosen a science-fiction setting, it shouldn't have magical elements that don't fit. Humorous events can certainly liven up an adventure, but they shouldn't be jarringly out of place.

The puzzles should be interrelated. Otherwise, what you'll end up with is a series of small puzzles rather than one complete adventure. Solving one puzzle should provide a tool that's needed for solving the next one. The various items required should be scattered around so that the character has to go back and forth among the rooms rather than having everything too neatly at hand.

Don't cheat. The puzzles should always be logical. The solution should make sense, at least once the player has stumbled upon it. A puzzle that reduces the player to trying actions at random has failed. If the way to summon a genie in your adventure is to kiss a coconut, be sure to provide some clue that will suggest that action. If you don't, you'll have a hard time getting people to play your second adventure.

Traps should not be sprung unexpectedly. It should be possible for the player to get a hint of danger ahead before walking into it, perhaps by requiring the player to examine things carefully. This doesn't mean that everything should be so easy that a player can solve it the first time. It means that at the end of the puzzle or game, the player sees that the program was "playing fair." One adventure game I've played, for example, requires the character to escape from a passage to survive, yet there was no indication that the passage was dangerous. This forces the player to rely on knowledge gained in a "previous life," something not as realistic as many players would like.

Just as when you create any game, the art of text adventure writing is much like the art of storytelling. To keep the player interested, interesting things have to happen. One event should follow reasonably from another and lead to a climax. Because it is a form of storytelling, the text adventure offers you, the author, a chance to express yourself, something not often found in other forms of videogames. When you write an adventure, you're doing more than creating a game; you're creating a world.

3
Recreation and Education

3
Trap 'Em

Jon Rhees

Build fences around your opponent without letting yourself get hemmed in. This simple game includes a variety of options to keep it ever challenging.

Don't Fence Me In

This game puts you in the construction business. Specifically, you're building fences, and the construction code is straightforward: Fences may be built horizontally or vertically; your construction may not touch the outer walls, your previous work, or your opponent's work; nor can it touch any obstacles that may be strewn in your path.

You score points by outlasting your rival. If your rival's fence crashes first, you win the round and a number of points based on the amount of time consumed by the round. The first player to reach 100 points wins the game.

You have these choices available in setting up the game: one or two players; joystick or keyboard input; adding obstacles to the playfield; and increasing or decreasing the speed of the game.

Approximately 30 percent of the program—the game action itself—is written in machine language. The sound, timing, and scoring routines are written in BASIC. Accompanying the article is a line-by-line description of how the program works.

The game is best when played by two people. The one-player option was added so that players could practice if no opponent could be found. You race the clock, trying to survive as long as possible. If you use the practice option, the most challenging level is nine, with obstacles. You have ten rounds to rack up as many points as you can.

Commodore 64 Program Description

Lines	**Description**
100–120	Call the option routines and initialize variables.
130	Checks for winner and jumps to win routine.
140–150	Draw screen border.
160	Checks for barrier option; jumps to subroutine.
190	Positions players and directions. (Locations 251–254 hold low and high bytes of each player's position. Locations 837–838 hold players' directions.)

Lines	Description
200	Initializes time and calls machine language routine, which returns to BASIC when collision occurs. Score is then determined based on amount of elapsed time.
210–230	Check value in location 834 for number of player in collision, then jump to appropriate routine to update winner's score.
245–250	Flash colliding fence.
270–480	Allow player to choose options.
490–510	Randomly place barriers on screen.
520–550	Initialize sound and variables.
560–660	Print scores and totals, then jump to beginning.
670–1350	Load machine language.

Trap 'Em

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
100 CLR:GOSUB670:GOSUB 520:GOSUB540:GOSUB260:GOSUB
    460                                          :rem 19
110 S1=0:S2=0:GOSUB410:IFFLTHEN100               :rem 25
120 PRINT"{CLR}":C=54272                         :rem 181
130 R=R+1:IFS1>=100OR(S2>=100ANDNP=2)OR(R=11ANDNP=
    1)THEN560                                    :rem 25
140 FORA=1104TO1143:POKEA,160:POKEA+C,0:POKEA+880,
    160:POKEA+880+C,0:NEXT                       :rem 253
150 FORA=1144TO1944STEP40:POKEA,160:POKEA+C,0:POKE
    A+39,160:POKEA+39+C,0:NEXT                   :rem 67
160 IFB$="Y"THENGOSUB490                         :rem 178
170 IFNP=2THENPRINT"{HOME}{7 SPACES}{RVS}{RED}RED"
    S1"{OFF}{13 SPACES}{RVS}{BLU}BLUE"S2         :rem 136
180 IFNP=1THENPRINT"{HOME}"TAB(8)"{RVS}{RED}SCORE"
    S2;SPC(8)"ROUND"R                            :rem 226
190 POKE251,194:POKE252,5:POKE253,214:POKE254,5:PO
    KE837,7:POKE838,11                           :rem 193
200 TI$="000000":SYS49152:SC=INT(TI/60):IFNP=1THEN
    SC=SC*LV                                     :rem 230
210 ONPEEK(834)GOTO220,230                       :rem 211
220 SP=PEEK(870)+256*PEEK(871):GOSUB245:S2=S2+SC:G
    OTO120                                       :rem 46
230 SP=PEEK(872)+256*PEEK(873):GOSUB245:S1=S1+SC:G
    OTO120                                       :rem 49
245 FORA=1TO6:POKESP,PEEK(SP)-2*(PEEK(SP)AND128)+1
    28:FORB=1TO400:NEXT                          :rem 3
250 NEXT:RETURN                                  :rem 240
260 REM OPTION ROUTINE                           :rem 123
270 POKE53281,1:PRINT"{CLR}{5 DOWN}"TAB(15)"{RED}T
    RAP 'EM":POKE198,0                           :rem 96
```

```
280 PRINT"{3 DOWN}"TAB(13)"{BLU}{RVS}1{OFF} ONE PL
    AYER":PRINT"{2 DOWN}"TAB(13)"{RVS}2{OFF} TWO P
    LAYER"                                :rem 171
290 PRINTSPC(13)"{2 DOWN}{RVS}3{OFF} QUIT" :rem 67
300 GETA$:IFA$<"1"ORA$>"3"THEN300          :rem 52
310 IFA$="3"THENPRINT"{CLR}":END          :rem 224
320 NP=VAL(A$):POKE836,NP:PRINT"{3 DOWN}{BLK}"TAB(
    14)"{RVS}J{OFF}OYSTICK OR"             :rem 49
330 PRINT"{DOWN}"TAB(8)"{RVS}K{OFF}EYBOARD AND JOY
    STICK 2"                              :rem 134
340 GETA$:IFA$="J"THENRETURN              :rem 228
350 IFA$<>"K"THEN340                       :rem 91
360 PRINT"{CLR}{3 DOWN}{7 SPACES}WHICH KEY TO GO U
    P?":WAIT198,1:A(1)=PEEK(197):POKE198,0:rem 207
370 PRINT"{DOWN}{7 SPACES}WHICH KEY TO GO DOWN?":W
    AIT198,1:A(2)=PEEK(197):POKE198,0     :rem 175
380 PRINT"{DOWN}{7 SPACES}WHICH KEY TO GO LEFT?":W
    AIT198,1:A(3)=PEEK(197):POKE198,0     :rem 164
390 PRINT"{DOWN}{7 SPACES}WHICH KEY TO GO RIGHT?":
    WAIT198,1:A(4)=PEEK(197):POKE198,0    :rem 249
400 FORA=1TO4:POKE829+A,A(A):NEXT:RETURN   :rem 11
410 PRINT"{CLR}{5 DOWN}"SPC(11)"ENTER SPEED (0-9)"
                                            :rem 1
420 PRINTSPC(7)"{2 DOWN}OR (C) TO CHANGE OPTIONS"
                                          :rem 161
430 GETA$:IF(A$<"0"ORA$>"9")ANDA$<>"C"THEN430
                                          :rem 203
440 IFA$="C"THENFL=1                      :rem 127
450 LV=VAL(A$):P=60-LV*6:POKE839,P:POKE840,P:LV=LV
    +1:RETURN                             :rem 168
460 PRINT"{CLR}"SPC(8)"{4 DOWN}DO YOU WANT BARRIER
    S?"                                    :rem 88
470 GETB$:IFB$<>"Y"ANDB$<>"N"THEN470       :rem 54
480 RETURN                                :rem 124
490 FORA=1TO30                             :rem 57
500 Q=RND(1)*870+1104:IFPEEK(Q)<>32OR(Q>1463ANDQ<1
    503)THEN500                           :rem 238
510 POKEQ,160:POKEQ+C,0:NEXT:RETURN       :rem 240
520 REM INITIALIZE                        :rem 109
530 FORA=54272TO54296:POKEA,0:NEXT:RETURN  :rem 71
540 POKE54287,255:POKE54290,129:POKE54273,7:POKE54
    296,15:POKE54277,21                   :rem 166
550 POKE54278,240:RETURN                  :rem 175
560 GOSUB520:IFNP=2THEN600                 :rem 77
570 PRINT"{CLR}{10 DOWN}"SPC(15)"{BLU}SCORE:"S2
                                          :rem 229
580 IFS2>HITHENHI=S2                        :rem 2
590 PRINTSPC(16)"{DOWN}{RED}HIGH:"HI:GOTO640
                                           :rem 50
```

```
600 W=-(S1>=100)-2*(S2>=100):PRINT"{CLR}{6 DOWN}"S
    PC(13)"{RED}PLAYER"W"WINS!"                 :rem 108
610 PRINT"{2 DOWN}{BLU}{4 SPACES}PLAYER1:"S1:PRINT
    SPC(25)"{UP}PLAYER2:"S2                     :rem 93
620 WI(W)=WI(W)+1:PRINT"{2 DOWN}{4 SPACES}WINS
    {3 SPACES}:"WI(1):PRINTSPC(25)"{UP}WINS
    {3 SPACES}:"WI(2)                           :rem 99
630 T1=T1+S1:T2=T2+S2:PRINT"{2 DOWN}{4 SPACES}TOTA
    L{2 SPACES}:"T1:PRINTSPC(25)"{UP}TOTAL
    {2 SPACES}:"T2                              :rem 176
640 PRINTSPC(15)"{2 DOWN}HIT ANY KEY":POKE198,0
                                                :rem 71
650 GETA$:IFA$=""THEN650                        :rem 89
660 ONNPGOTO100,110                             :rem 95
670 I=49152:IFPEEK(I)=32THENRETURN              :rem 97
680 PRINT"{CLR}{5 DOWN}"SPC(13)"PLEASE WAIT"
                                                :rem 37
690 READ A:IF A=256 THEN RETURN                 :rem 239
700 POKE I,A:I=I+1:GOTO 690                     :rem 243
710 DATA 32,22,192,32,229,192                   :rem 145
720 DATA 173,66,3,240,1,96                      :rem 255
730 DATA 32,72,193,165,197,208                  :rem 210
740 DATA 237,76,15,192,169,33                   :rem 162
750 DATA 141,4,212,162,3,181                    :rem 87
760 DATA 251,157,102,3,202,16                   :rem 137
770 DATA 248,160,100,173,0,220                  :rem 185
780 DATA 41,15,201,15,208,3                     :rem 38
790 DATA 173,70,3,141,61,3                      :rem 251
800 DATA 141,70,3,173,1,220                     :rem 30
810 DATA 141,60,3,165,197,205                   :rem 146
820 DATA 62,3,208,4,162,254                     :rem 45
830 DATA 208,33,205,63,3,208                    :rem 94
840 DATA 4,162,253,208,24,205                   :rem 144
850 DATA 64,3,208,4,162,251                     :rem 47
860 DATA 208,15,205,65,3,208                    :rem 99
870 DATA 4,162,247,208,6,173                    :rem 106
880 DATA 60,3,76,111,192,138                    :rem 103
890 DATA 45,60,3,41,15,201                      :rem 247
900 DATA 15,208,3,173,69,3                      :rem 255
910 DATA 141,60,3,141,69,3                      :rem 247
920 DATA 136,208,166,169,32,141                 :rem 253
930 DATA 4,212,206,71,3,208                     :rem 40
940 DATA 154,173,72,3,141,71                    :rem 98
950 DATA 3,160,0,162,0,185                      :rem 245
960 DATA 60,3,74,176,8,169                      :rem 15
970 DATA 40,32,199,192,76,190                   :rem 165
980 DATA 192,74,176,8,169,40                    :rem 120
990 DATA 32,217,192,76,190,192                  :rem 214
1000 DATA 74,176,8,169,1,32                     :rem 46
1010 DATA 199,192,76,190,192,169                :rem 58
```

```
1020 DATA 1,32,217,192,232,232              :rem 179
1030 DATA 200,204,68,3,208,207              :rem 182
1040 DATA 96,141,67,3,181,251               :rem 146
1050 DATA 56,237,67,3,149,251               :rem 153
1060 DATA 181,252,233,0,149,252             :rem 239
1070 DATA 96,24,117,251,149,251             :rem 250
1080 DATA 181,252,105,0,149,252             :rem 239
1090 DATA 96,160,0,173,68,3                  :rem 50
1100 DATA 201,1,240,35,165,251              :rem 174
1110 DATA 197,253,208,29,165,252             :rem 46
1120 DATA 197,254,208,23,173,27             :rem 249
1130 DATA 212,16,9,169,1,141                 :rem 87
1140 DATA 66,3,32,83,193,96                  :rem 55
1150 DATA 169,2,141,66,3,32                  :rem 41
1160 DATA 105,193,96,160,0,140              :rem 188
1170 DATA 66,3,173,27,212,16                 :rem 94
1180 DATA 7,32,41,193,32,53                  :rem 43
1190 DATA 193,96,32,53,193,32               :rem 156
1200 DATA 41,193,96,177,251,201             :rem 244
1210 DATA 32,240,5,169,1,141                 :rem 81
1220 DATA 66,3,96,173,68,3                    :rem 5
1230 DATA 201,1,240,11,177,253              :rem 177
1240 DATA 201,32,240,5,169,2                 :rem 82
1250 DATA 141,66,3,96,32,105                 :rem 93
1260 DATA 193,173,68,3,201,2                 :rem 93
1270 DATA 240,1,96,169,219,145              :rem 204
1280 DATA 253,165,253,24,105,0              :rem 189
1290 DATA 133,106,165,254,105,212            :rem 82
1300 DATA 133,107,169,6,145,106             :rem 240
1310 DATA 96,160,0,169,214,145              :rem 194
1320 DATA 251,165,251,24,105,0              :rem 180
1330 DATA 133,106,165,252,105,212            :rem 75
1340 DATA 133,107,169,2,145,106             :rem 240
1350 DATA 96,256                             :rem 27
```

Mindbusters

Ned W. Schultz

Here's a graphics puzzle game that is both challenging and unusually fascinating. Try it on three levels.

Rack Your Brain

Are you ready to pit your brain against the computer's? "Mindbusters" presents you with three graphics puzzles that are guaranteed to keep your mind's microprocessors and memory chips whirring for hours.

After you type, save, and run your copy of Mindbusters, you can choose to solve one of three puzzles: a mind bender, a mind bruiser, or a mind blower. Warm up with the mind bender—it's the easiest. When you're prepared to press your brain to its limits, you're ready for the mind blower.

Following your selection, the program constructs a puzzle and displays it at the upper-left corner of the screen. Your job is to match that puzzle in the workspace at the lower-right corner of the screen. What's more, you try to solve the puzzle in as little time as possible. A timer ticks away as you work. There's no limit to how much time you can take, but the timer lets you compare your progress with a previous performance or against another player if you wish. Your fastest time during the current session will be displayed on the screen.

Each puzzle is composed of several horizontal rows of odd shapes. A tiny arrow to the right of the workspace points to the row you're currently working on. To work on different rows, you can move the arrow up and down with the I and M keys. To move the row of shapes next to the arrow left or right, press the J or K key. When you think you've matched a row to the puzzle pattern, start working on another row.

When you succeed in correctly matching all the rows, the program automatically signals that you've solved the puzzle. Then you can play again if you like.

Helpful Hints

Because Mindbusters can generate a tremendous number of different puzzles, there are very few tricks to mastering it. You probably should work from top to bottom or vice versa. The best tip is to concentrate, concentrate, concentrate.

Important: When typing in the program, be extra careful with the long strings of characters at the beginning of the listing. These strings become the puzzle shapes. If you mistype or transpose a couple of characters when typing these strings, the program may still run, but it won't know when you've solved the puzzle. If you're using COMPUTE!'s "Automatic Proofreader" (Appendix C) to enter the listing, remember that the Proofreader does not catch character-transposition errors.

Mindbusters

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
10 S=54272:R$="000000":FORI=1TO4:READKE(I):NEXT
                                            :rem 238
20 PRINTCHR$(14)CHR$(8)                      :rem 48
30 A$="XVAWVBWWAXNAWVBWWAXAWVBWANANAVVWANNNXAWVWAX
   WAXCVNVNWAWVBNWCCXVNVAWNW"                :rem 57
40 TM$="":FORI=1TO68:TM=ASC(MID$(A$,I,1))+97:TM$=T
   M$+CHR$(TM):NEXT:A$=TM$                   :rem 45
50 B$="12*0Z*Z*,0<2Z/*/00,Z/02ZZ2Z*1,<Z-21,-2*Z<0Z
   -210*,Z*Z*1<122Z<Z1*<Z,*Z"               :rem 167
60 TM$="":FORI=1TO68:TM=ASC(MID$(B$,I,1))+129:TM$=
   TM$+CHR$(TM):NEXT:B$=TM$                  :rem 93
70 C$="ZZ$$$ZZ$$Z$Z$Z$Z$ZZZZZ$Z$$$Z$$$Z$Z$$$$Z$Z$Z
   ZZZ$Z$Z$$$Z$ZZZZZZ$$Z$$Z$"                :rem 11
80 TM$="":FORI=1TO68:TM=ASC(MID$(C$,I,1))+133:TM$=
   TM$+CHR$(TM):NEXT:C$=TM$                  :rem 92
90 POKE53281,1:PRINT"{CLR}{PUR}******************
   **********************";                 :rem 109
100 PRINT"[4]{RVS}{14 SPACES}MINDBUSTERS
    {15 SPACES}{OFF}";                      :rem 168
110 PRINT"{PUR}***********************************
    *****":GOSUB460:POKE53280,7              :rem 33
120 PRINT"{HOME}{3 DOWN}{4 RIGHT}[4][12 P]":PRINT"
    {3 RIGHT}[N]"SPC(12)"[H]{2 RIGHT}USE I, J, K A
    ND M"                                   :rem 132
130 PRINT"{3 RIGHT}[N]"SPC(12)"[H]":PRINT"
    {3 RIGHT}[N]"SPC(12)"[H]{2 RIGHT}KEYS TO MATCH
     THIS"                                   :rem 84
140 PRINT"{3 RIGHT}[N]"SPC(12)"[H]":PRINT"
    {3 RIGHT}[N]"SPC(12)"[H]{2 RIGHT}PATTERN AS FA
    ST"                                     :rem 177
150 PRINT"{3 RIGHT}[N]"SPC(12)"[H]":PRINT"
    {3 RIGHT}[N]"SPC(12)"[H]{2 RIGHT}AS YOU CAN!!!
    !"                                      :rem 185
160 PRINT"{3 RIGHT}[N]"SPC(12)"[H]":PRINT"
    {4 RIGHT}[12 Y]"                         :rem 14
```

```
170 POKE214,3:PRINT                               :rem 132
180 FORN=1TO8:PP(N)=INT(RND(1)*56)+1:PRINT"
    {4 RIGHT}"CHR$(Z)MID$(D$,PP(N),12)            :rem 60
190 NEXT:PRINT:PRINTTAB(19)"{BLK}E12 P3"          :rem 1
200 FORN=1TO8:PRINTTAB(18)"EN3"SPC(12)"EH3":NEXT:P
    RINTTAB(19)"E12 Y3"                           :rem 146
210 POKE214,13:PRINT                              :rem 176
220 FORN=1TO8:P(N)=INT(RND(1)*56)+1:PRINTTAB(19)CH
    R$(Z)MID$(D$,P(N),12):NEXT                    :rem 234
230 AL=1616:POKEAL,31:POKEAL+S,0:AC=1:TI$="000000"
                                                   :rem 75
240 POKE198,0:KE=PEEK(197):J=0:FORI=1TO4:IFKE=KE(I
    )THENJ=I:I=4                                   :rem 52
250 NEXT:ONJGOTO280,320,300,340                    :rem 13
260 POKE214,13:PRINT:PRINT"{4 RIGHT}{RED}{RVS}RECO
    RD{OFF}{RIGHT}{BLK}"MID$(R$,3,2)+":"+MID$(R$,5
    ,2)                                           :rem 186
270 PRINT"{DOWN}{4 RIGHT}{RVS}TIME{OFF}{3 RIGHT}"M
    ID$(TI$,3,2)":"MID$(TI$,5,2):GOTO240          :rem 188
280 POKEAL,32:AL=AL-40:AC=AC-1:IFAL<1616THENAL=161
    6:AC=1                                         :rem 57
290 POKEAL,31:POKEAL+S,0:GOTO240                  :rem 192
300 POKEAL,32:AL=AL+40:AC=AC+1:IFAL>1896THENAL=189
    6:AC=8                                         :rem 75
310 GOTO290                                       :rem 104
320 POKE214,12+AC:PRINT:P(AC)=P(AC)-1:IFP(AC)<1THE
    NP(AC)=1                                      :rem 156
330 GOTO350                                       :rem 103
340 POKE214,12+AC:PRINT:P(AC)=P(AC)+1:IFP(AC)>56TH
    ENP(AC)=56                                     :rem 18
350 PRINTTAB(19)CHR$(Z)MID$(D$,P(AC),12)          :rem 250
360 FORX=1TO8:IFPP(X)<>P(X)THEN240                :rem 107
370 NEXT:SC$=TI$                                  :rem 203
380 POKE214,15:PRINT:PRINT"{4 RIGHT}{BLK}{RVS}TIME
    {OFF}{3 RIGHT}"MID$(SC$,3,2)+":"+MID$(SC$,5,2)
                                                  :rem 213
390 PRINT"{DOWN}{3 RIGHT}{PUR}PUZZLE SOLVED!":GOSU
    B570:PRINT"{DOWN}{BLK}{4 RIGHT}PLAY AGAIN?"
                                                  :rem 148
400 PRINTSPC(7)"{DOWN}{RVS}Y{OFF}/{RVS}N{OFF}"
                                                    :rem 2
410 POKE53280,4:GETK$:IFK$=""THENPOKE53280,3:GOTO4
    10                                             :rem 47
420 IFK$="N"THENSYS2048                            :rem 95
430 IFR$="000000"ORSC$<R$THENR$=SC$               :rem 230
440 IFK$="Y"THEN90                                  :rem 8
450 GOTO410                                       :rem 103
460 PRINTSPC(10)"{3 DOWN}{BLK}DO YOU WANT TO:":PRI
    NTSPC(11)"{DOWN}{RVS}1{OFF} BEND YOUR MIND?"
                                                  :rem 198
```

```
470 PRINTSPC(11)"{DOWN}{RVS}2{OFF} BRUISE YOUR MIN
    D?"                                     :rem 236
480 PRINTSPC(11)"{DOWN}{RVS}3{OFF} BLOW YOUR MIND?
    "                                        :rem 88
490 POKE53280,3:GETK$:IFK$=""THENPOKE53280,4:GOTO4
    90                                       :rem 63
500 K=VAL(K$):IFK<1ORK>3THEN490             :rem 106
510 IFK=1THEND$=A$:Z=31:GOTO540              :rem 88
520 IFK=2THEND$=B$:Z=28:GOTO540              :rem 97
530 D$=C$:Z=144                              :rem 14
540 PRINT"{HOME}{3 DOWN}":FORN=1TO10:PRINT"
    {39 SPACES}":NEXT                        :rem 21
550 RETURN                                  :rem 122
560 DATA 33,37,36,34                        :rem 217
570 FORI=STOS+24:POKEI,0:NEXT:POKES+24,15:POKES+5,
    48:POKES+6,48                           :rem 178
580 POKES+4,33:FORI=20TO80STEP3:POKES+1,I:FORJ=1TO
    50:NEXT:NEXT:POKES+4,32                 :rem 159
590 POKES+24,0:RETURN                        :rem 39
```

Squares

Douglas Fish

Teach your snake well: It will remember each move you make as you try to conquer the board with your squares. A strategy game for one to four players.

Reptilian Intelligence

At first glance, "Squares" looks a lot like Dots, the paper and pencil game where opponents take turns connecting dots to try and complete squares. And, as in the paper game, the basic objective is to complete more squares than your opponents. But the similarities end there—in Squares, the dots are connected by an intelligent "snake," which you control.

After loading and running Squares, you are asked if each of the four snakes will be controlled by a player or by the computer. Moves for the player-controlled snakes are entered via the keyboard; the computer snakes move around somewhat randomly.

You can move your snake up, down, left, or right by pressing the I, M, J, or K keys respectively (as a reminder, the directions are printed on the screen during the game). When you move your snake between two dots, it leaves a trail in your player's color.

With each move you make, you train your snake to move in a certain way, depending on the pattern of trails around it. For example, say, there are trails to the left of and below your snake, and you move it up. From then on, whenever your snake encounters a pattern in which there are trails to the left and below it, it will move up.

If the snake encounters a pattern it hasn't learned yet, as when you first start the game, it will ask you for a direction. Again, the direction you choose will train the snake for that pattern.

Trapped Snakes

A snake can become trapped, though, if you give it an instruction which forms a loop with a previous instruction. For instance, you tell it to go right, but when it moves right, it enters a pattern where it has been instructed to move left. It then becomes trapped between those two instructions. A trapped snake can be released later, however, if the pattern it's in is changed by another snake.

When your snake completes a square, it fills in with your color, and you earn a point. The game is over when all the squares are filled or all the snakes are trapped. Whoever completes the most squares wins the game.

There are a number of strategies you can develop for conquering long rows of squares or avoiding getting trapped. You may find, though, that it's difficult to remember how your snake has been trained for each possible pattern of trails. Also, each game that you play will be unique, so what works for one game may get you trapped early on in the next. It's usually a combination of strategy and chance that wins the game.

Squares

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
10 POKE53281,Ø:POKE53280,Ø:PRINT"{WHT}"     :rem 198
20 DIMIN(15,4):FORA=984TO1Ø23:POKEA,32:NEXT:rem 75
30 DR(Ø)=-4Ø:DR(1)=1:DR(2)=4Ø:DR(3)=-1      :rem 235
40 CL(1)=3:CL(2)=4:CL(3)=11:CL(4)=9          :rem 82
50 P(1)=136Ø:P(2)=1358:P(3)=144Ø:P(4)=1438:CO=5427
   2                                         :rem 15Ø
60 FORX=1TO4:P(X)=1Ø24+INT(RND(1)*15)*2+INT(RND(1)
   *1Ø)*8Ø:NEXT                              :rem 61
70 GOTO56Ø                                   :rem 59
80 PRINT"{CLR}";:FORX=1TO1Ø:FORY=1TO15:PRINT"Q ";:
   NEXT:PRINT:PRINT:NEXT                     :rem 122
90 GOSUB63Ø:PRINTTAB(22);"{DOWN} I{DOWN}{2 LEFT}J+
   K{DOWN}{2 LEFT}M"                          :rem 1
100 QF=1:FORPL=1TO4:P=P(PL):CL=CL(PL):Q=Ø:FL=Ø
                                             :rem 79
110 P1=P:GOSUB38Ø:P2=SI:LF=Ø                 :rem 195
120 GOSUB47Ø:GOSUB63Ø:PRINT"{2 DOWN}PLAYER"PL"'S T
    URN";:POKE646,CL:PRINT"{2 SPACES}Q{WHT}"
                                             :rem 126
130 PRINT "{21 SPACES}"                      :rem 1Ø1
140 GOSUB38Ø                                 :rem 176
150 IN=IN(SI,PL):IFIN=ØTHENGOSUB29Ø          :rem 2Ø5
160 GOTO5ØØ                                  :rem 1Ø1
170 IFABS(IN)=1THENPOKEP+IN,67:GOTO19Ø       :rem 1Ø5
180 POKEP+IN,66                              :rem 114
190 POKEP+CO,1:POKEP+CO+IN*2,CL:POKEP+IN+CO,CL
                                             :rem 117
200 P(PL)=P+IN*2:GOSUB41Ø                    :rem 2Ø1
210 IF(S1=15)AND(PEEK(X+D)=32)THENPOKEX+D,16Ø:POKE
    X+D+CO,CL:B(PL)=B(PL)+1                  :rem 94
```

```
220 IF(S2=15)AND(PEEK(X-D)=32)THENPOKEX-D,160:POKE
    X-D+CO,CL:B(PL)=B(PL)+1                    :rem 102
230 P=P(PL):GOSUB520                           :rem 176
240 GOSUB380:IFP1=PANDP2=SITHENLF=1:Q=9        :rem 230
250 IFQ=>9THEN270                              :rem 243
260 Q=Q+1:GOTO140                              :rem 219
270 IFLF=0THENQF=0                              :rem 90
280 NEXTPL:GOTO640                             :rem 130
290 GOSUB630:PRINT"{3 DOWN}WHAT DIRECTION":POKEP+C
    O,CL:POKE198,0                              :rem 95
300 IFTY(PL)=2THENGOSUB340:Q=10:GOTO330        :rem 163
310 GETA$:IFA$<>"I"ANDA$<>"M"ANDA$<>"J"ANDA$<>"K"T
    HEN310                                     :rem 149
320 Q=10:D=(A$="J")*-3+(A$="K")*-1+(A$="M")*-2
                                               :rem 183
330 IN(SI,PL)=DR(D):IN=IN(SI,PL):RETURN        :rem 190
340 IFSI=15THEND=INT(RND(1)*4):RETURN           :rem 47
350 IFFL=>4THEND=INT(RND(1)*4):RETURN           :rem 50
360 D=INT(RND(1)*4):IF(SIAND2↑D)=2↑DTHEN360:rem 80
370 FL=FL+1:RETURN                             :rem 113
380 SI=0:FORX=0TO3:I=PEEK(DR(X)+P)              :rem 80
390 IFI<>32THENSI=SI+2↑X                        :rem 10
400 NEXT:RETURN                                :rem 237
410 S1=0:S2=0:X=(P(PL)+P)/2:IFABS(X-P)=1THEND=40:G
    OTO430                                      :rem 60
420 D=1                                         :rem 72
430 FORY=0TO3:Z=PEEK(X+DR(Y)+D):IF(Z=66)OR(Z=67)TH
    ENS1=S1+2↑Y                                 :rem 46
440 NEXT                                       :rem 215
450 FORY=0TO3:Z=PEEK(X+DR(Y)-D):IFZ=(66)OR(Z=67)TH
    ENS2=S2+2↑Y                                 :rem 52
460 NEXT:RETURN                                :rem 243
470 GOSUB630:PRINT"{19 SPACES}"                :rem 191
480 PRINT"{18 SPACES}"                         :rem 109
490 PRINT"{18 SPACES}":GOTO520                 :rem 120
500 IFPEEK(P+IN*2)=81THEN170                   :rem 117
510 GOSUB630:PRINT:GOSUB480:GOSUB630:PRINT"{DOWN}I
    LLEGAL MOVE":GOSUB290:GOTO140              :rem 201
520 PRINT"{HOME}":FORX=1TO4                     :rem 57
530 PRINT TAB(29);" PLR."X;:POKE646,CL(X):PRINT"Q
    {WHT}"                                      :rem 52
540 PRINTTAB(30);B(X):NEXT                      :rem 80
550 RETURN                                     :rem 122
560 PRINT"{CLR}{6 DOWN}{RVS}E1}"SPC(16)"SQUARES
    {WHT}"                                       :rem 3
570 PRINT"{7 DOWN}{10 SPACES}{CYN}1. PLAYER CONTRO
    LLED                                       :rem 131
580 PRINT"{WHT}{DOWN}{10 SPACES}{GRN}2. COMPUTER C
    ONTROLLED                                   :rem 69
```

```
590 FORX=1TO4                                    :rem 34
600 PRINT"{YEL}{HOME}{1Ø DOWN}{8 SPACES}SNAKE "X"
    {SPACE}(CHOOSE 1 OR 2){WHT}"                :rem 235
61Ø GETA$:IFVAL(A$)>2ORVAL(A$)=ØTHEN61Ø         :rem 27
62Ø TY(X)=VAL(A$):NEXT:GOTO8Ø                   :rem 24
63Ø PRINT"{HOME}":FORQQ=1TO18:PRINT:NEXT:RETURN
                                                 :rem 2Ø
64Ø IFQF=ØTHEN1ØØ                                :rem 237
65Ø PRINT"{CLR}{6 DOWN}"SPC(14)"{RVS}E1}GAME OVER!
    {3 DOWN}"                                    :rem 13Ø
66Ø FORX=1TO4:POKE 646,CL(X):PRINTTAB(7)"{DOWN}PLA
    YER"X;"...."B(X)" SQUARES"                  :rem 183
67Ø NEXT                                         :rem 22Ø
68Ø PRINT"{3 DOWN}"SPC(1Ø)"{WHT}ANOTHER GAME? (Y/N
    )":POKE198,Ø                                 :rem 123
69Ø GETA$:IFA$="Y"THENRUN                        :rem 16
7ØØ IFA$="N"THENPRINT"{CLR}":END                 :rem 254
71Ø GOTO69Ø                                      :rem 112
```

Quiz Master

George W. Miller

This two-program package offers an effective and uncomplicated way to set up and administer multiple-choice quizzes. It's menu-driven for ease of use, and ideal for school or home study.

"Quiz Master," a package of two programs, includes "Quiz Generator" and "Student Quiz." Together, they can be used to create and administer quizzes. The first program allows parents or teachers to create multiple-choice tests, while the second presents the tests to the student. The only thing the student has to do is answer the questions.

Abbreviations Required

Type in Program 1, Quiz Generator, and save it on a new disk. You'll be using Quiz Generator to generate sequential files, which can use up disk space rather quickly, so it's best to start with a fresh disk.

Typing in the Quiz Master package is simplified when you use "Automatic Proofreader," the error-checking program you'll find in Appendix C. Make sure you read the explanation and have a copy of the Proofreader on disk before you begin typing in either Quiz Generator or Student Quiz.

Menu Options

Once you've typed in Program 1, Quiz Generator, load and run it. It begins with a display of the main menu, which includes these categories: Enter New Questions, Review Questions, Change a Question, Load Previous Data, Add to Test in File, Initialize Disk, and End.

Press the 1 key to enter new questions and create a quiz. You'll then be asked if a file of quiz names exists. If this is the first time you've used the program or if you're starting a new group of tests on a new disk, answer by hitting N. Next, provide a name for your quiz. The quiz name is stored in a SEQuential file called TEST TITLES. Quiz Generator accepts up to 15 quiz files for each disk (a limitation because of the menu's screen formatting). If you're covering more than one subject, you may want to have a separate disk for each—for instance, a disk for history quizzes, another disk for math quizzes, and so on.

Just follow the prompts to enter your quiz. You have full use of all screen editing functions, including the cursor control keys and the INST/DEL key. Be careful to make changes only where you intend to, and don't move the cursor to areas where other text appears.

You shouldn't be concerned about word wraparound, the breaking of words at the end of the 40-column line. Just type each sentence, putting spaces where they normally occur, and use standard punctuation, including commas and colons. Quiz Generator looks at your sentences and finds the proper place to break each line. Each question can contain up to 80 characters, counting spaces.

Type in the four answer choices to the question, and give the correct letter choice when prompted. Each quiz can contain up to 100 questions and their answers. To store the quiz, type the pound symbol (£). The program opens a file with the quiz name you specified and stores your information. A file to store the student's grades is also created.

When you return to the menu, type 2 to review the questions. The screen formatting section of the program right justifies your questions, and the screen display ends each line with the last word which fits without hyphenation.

Follow the screen prompts to review each question. You'll be shown the questions, answer choices, and the letter of the correct answer to make sure that you made no errors when you entered the quiz. If you notice any mistakes, jot down the number of the question so that you can change it later.

If you want to change any of the questions, enter 3 and answer the prompts. You'll have to enter the number of the question you want to change—that's why you jotted them down when you reviewed the quiz (option 2). The computer displays the question and answer choices, and you can enter the correct question and answer choices.

Option 4, Load Previous Data, loads a quiz previously stored. You can then review this quiz.

Select option 5 if you want to add questions to a quiz already stored on your disk. You'll start entering questions at the first unused question number in the file.

The Initialize Disk routine, option 6, formats, or NEWs, a disk and gives you several chances to abort the routine. Make certain the disk in the drive is the one you want formatted since all information on it will be destroyed by the routine. You can't enter this routine by accident, because you're actually taken out of the program before you can run it.

Exiting Quiz Generator is simple; just select option 7, End.

Student Quiz

Next, type in Program 2, Student Quiz, and save it. (Be sure to save this program before typing RUN, as any mistakes will give you a scrambled, tokenized BASIC listing.) If you plan to use Quiz Generator to give tests to groups of students, save Student Quiz on a second disk for use by the students. This will safeguard Quiz Generator from accidental erasure.

When a student loads and runs Student Quiz, RUN/STOP–RESTORE and LIST are disabled, as are all cursor controls. The student can answer only the prompts from the computer. The student is asked which quiz has been assigned, and that quiz is then loaded and run. With the checks built into the program, all the student can do is enter A, B, C, or D as answers.

Because a random number routine is used to scramble the order of the questions, the quiz will be different each time. Quiz Generator also uses one question fewer than you've placed in memory. In effect, each student takes a slightly different quiz each time the quiz is given. The more questions you store in the file, the more variations Quiz Generator has to work with.

Since the random number generator searches for new numbers every time, it can take several minutes to generate a quiz, especially if you have numerous questions in the file. The screen will be blank during this process, and all keys will be disabled. Everything will return to normal when the quiz is ready.

Program 1. Quiz Generator

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
10 REM QUIZ MASTER                                  :rem 90
20 DIMQ$(100),A$(100),B$(100),C$(100),D$(100),E$(1
   00),M$(15)                                       :rem 48
30 PRINT"{WHT}":POKE53280,13:POKE53281,5:GOTO50
                                                    :rem 217
```

```
40 POKE198,6:POKE631,30:POKE632,34:POKE633,34:POKE
   634,20:POKE635,5:RETURN                        :rem 110
50 POKE53272,23:GOTO990                            :rem 61
60 PRINT"{CLR}":CLR                               :rem 229
70 DIMQ$(100),A$(100),B$(100),C$(100),D$(100),E$(1
   00),M$(15),SN$(400),G(400)                      :rem 99
80 GOSUB1840:GOSUB1380:GOSUB2060                  :rem 189
90 PRINT"ENTER NUMBER OF TEST TO LOAD:":INPUTN
                                                  :rem 235
100 IFN<0ORN>XTHENPRINT"INVALID RANGE":GOTO90
                                                  :rem 175
110 N$=M$(N):PRINTSPC(12)"{CLR}{RVS}{6 DOWN}
    {9 RIGHT}{3 SPACES}LOADING DATA{3 SPACES}"
                                                  :rem 158
120 GOSUB1840:GOSUB2140:OPEN2,8,2,+N$+" FILE,S,R":
    X=0                                            :rem 21
130 X=X+1                                         :rem 221
140 INPUT#2,Q$(X):INPUT#2,A$(X):INPUT#2,B$(X)
                                                  :rem 119
150 INPUT#2,C$(X):INPUT#2,D$(X):INPUT#2,E$(X)
                                                  :rem 112
160 IFST AND64THEN180                             :rem 210
170 GOTO130                                       :rem 101
180 CLOSE2:POKE198,0:L=X:N=X:T=0                  :rem 219
190 IFR=1THEN250                                  :rem 175
200 GOSUB2140:GOSUB2060:GOSUB1690                 :rem 229
210 IFH=0THEN990                                  :rem 168
220 IFH=1THEN250                                  :rem 159
230 REM INPUT QUESTIONS                           :rem 212
240 PRINT"{CLR}{DOWN}ENTER NAME FOR QUIZ":INPUTN$:
    GOSUB1480:N=0                                  :rem 18
250 N=N+1:PRINT"{CLR}":PRINTSPC(13)"{RVS} QUIZ MAS
    TER {OFF}"                                    :rem 201
260 PRINT:PRINT"{RVS} WARNING!{2 SPACES}DO NOT EXC
    EED 80 CHARACTERS{2 SPACES}"                  :rem 115
270 PRINT:PRINT"{RVS}{7 SPACES}ENTER £ TO EXIT RO
    UTINE{10 SPACES}"                             :rem 160
280 IFN>=100THENPRINT"{CLR}{5 DOWN}{14 SPACES}FILE
     FULL":FORT=1TO2000:NEXT:GOTO1000             :rem 179
290 GOSUB40                                       :rem 127
300 H=0                                            :rem 72
310 PRINT"ENTER QUESTION #";N:PRINT               :rem 206
320 INPUTQ$(N)                                     :rem 57
330 IFQ$(N)=""THEN320                             :rem 126
340 IFQ$(N)=CHR$(92)THENN=N-1:GOTO780             :rem 172
350 IFLEN(Q$(N))>80THENGOSUB1310                  :rem 133
360 IFH=1THEN250                                  :rem 164
370 PRINT"ENTER FIRST ANSWER:":PRINT:GOSUB40:H=0
                                                  :rem 207
380 INPUT"A. ";A$(N):IFA$(N)=""THEN380             :rem 53
```

```
390 IFASC(A$(N))=92THENN=N-1:GOTO780          :rem 119
400 A$(N)="A. "+A$(N)                         :rem 183
410 IFLEN(A$(N))>80THENGOSUB1310              :rem 114
420 IFH=1THENGOTO370                          :rem 221
430 PRINT"ENTER SECOND ANSWER:":PRINT:GOSUB40:H=0
                                                :rem 0
440 INPUT"B. ";B$(N):IFB$(N)=""THEN440         :rem 50
450 IFASC(B$(N))=92THENN=N-1:GOTO780          :rem 117
460 B$(N)="B. "+B$(N)                         :rem 192
470 IFLEN(B$(N))>80THENGOSUB1310              :rem 121
480 IFH=1THENGOTO430                          :rem 224
490 PRINT"ENTER THIRD ANSWER:":PRINT:GOSUB40:H=0
                                              :rem 197
500 INPUT"C. ";C$(N):IFC$(N)=""THEN500         :rem 47
510 IFASC(C$(N))=92THENN=N-1:GOTO780          :rem 115
520 C$(N)="C. "+C$(N)                         :rem 192
530 IFLEN(C$(N))>80THENGOSUB1310              :rem 119
540 IFH=1THENGOTO490                          :rem 227
550 PRINT"ENTER FOURTH ANSWER:":PRINT:GOSUB40:H=0
                                               :rem 31
560 INPUT"D. ";D$(N):IFD$(N)=""THEN560         :rem 62
570 IFASC(D$(N))=92THENN=N-1:GOTO780          :rem 122
580 D$(N)="D. "+D$(N)                         :rem 201
590 IFLEN(D$(N))>80THENGOSUB1310              :rem 126
600 IFH=1THENGOTO550                          :rem 221
610 PRINT"ENTER LETTER OF CORRECT ANSWER:":PRINT:H
    =0                                         :rem 29
620 INPUTE$(N):IFE$(N)=""THEN620               :rem 73
630 IFASC(E$(N))=92THENN=N-1:GOSUB780:GOTO1010
                                              :rem 244
640 IFLEN(E$(N))<>1THENGOSUB1310              :rem 128
650 IFE$(N)="A"THEN700                        :rem 186
660 IFE$(N)="B"THEN700                        :rem 188
670 IFE$(N)="C"THEN700                        :rem 190
680 IFE$(N)="D"THEN700                        :rem 192
690 PRINT"{RVS} ERROR: RE-ENTER":GOTO620       :rem 19
700 IFH=1THEN610                              :rem 162
710 IFP=1THENRETURN                           :rem 244
720 L=N:GOSUB2060:IFN=100THEN740              :rem 161
730 GOTO250                                   :rem 106
740 PRINT:PRINTSPC(7)"FILE CONTAINS 100 ENTRIES."
                                              :rem 249
750 PRINT:PRINT"DATA WILL BE STORED. OPEN NEW TEXT
     FILE"                                    :rem 141
760 GOSUB1740:GOSUB780:GOTO1010               :rem 119
770 REM STORE DATA                             :rem 41
780 GOSUB2060:PRINTSPC(10)"{RVS} WAIT, STORING DAT
    A ":GOSUB2140                              :rem 62
790 GOSUB1840:OPEN2,8,2,"@0:"+N$+" FILE,S,W"
                                              :rem 149
```

```
800 FORX=1TOL:PRINT#2,Q$(X):PRINT#2,A$(X):PRINT#2,
    B$(X)                                     :rem 71
810 PRINT#2,C$(X):PRINT#2,D$(X):PRINT#2,E$(X):NEXT
                                              :rem 227
820 CLOSE2:POKE198,0:GOSUB2140:GOSUB2060:GOSUB1690
    :RETURN                                   :rem 80
830 REM CHANGE ANSWER                         :rem 245
840 GOSUB2060:P=1:PRINT"{CLR}{3 DOWN}{RVS} ENTER N
    UMBER OF QUESTION":INPUTW                 :rem 180
850 PRINT"{CLR}{2 DOWN}":S$=Q$(W):GOSUB1210:S$=A$(
    W):GOSUB1210:S$=B$(W):GOSUB1210           :rem 159
860 S$=C$(W):GOSUB1210:S$=D$(W):GOSUB1210     :rem 91
870 PRINT"CORRECT ANSWER IS:":PRINTE$(W)      :rem 128
880 GOSUB1690:N=W-1:GOSUB250:GOSUB780:RETURN
                                              :rem 46
890 REM REVIEW ROUTINE                        :rem 125
900 GOSUB2060:Y=1:PRINT"{CLR}{DOWN}"          :rem 145
910 PRINT:PRINTTAB(20-LEN(N$)/2);N$:GOSUB1690:PRIN
    T"{2 DOWN}"                               :rem 12
920 FORN=1TOL:PRINT"{CLR}"                    :rem 203
930 IFQ$(N)=""THENGOTO980                     :rem 201
940 S$=STR$(N)+". "+Q$(N):PRINT:GOSUB1210     :rem 46
950 REM ANSWER CHOICES                        :rem 80
960 S$=A$(N):GOSUB1210:S$=B$(N):GOSUB1210:S$=C$(N)
    :GOSUB1210:S$=D$(N):GOSUB1210             :rem 43
970 PRINT:PRINT"CORRECT ANSWER IS: ":PRINTTAB(7)E$
    (N):GOSUB2060                             :rem 32
980 GOSUB1690:NEXT:RETURN                     :rem 132
990 REM PROGRAM MENU                          :rem 211
1000 H=0                                      :rem 118
1010 PRINT"{CLR}":POKE53280,13:POKE53281,5:rem 238
1020 GOSUB2060                                :rem 11
1030 P=0                                      :rem 129
1040 PRINTSPC(13)"{DOWN}{RVS} QUIZ MASTER "
                                              :rem 105
1050 PRINT:PRINTSPC(5)"ENTER NUMBER OF FUNCTION:"
                                              :rem 198
1060 PRINT:PRINTSPC(8)"1. ENTER NEW QUESTIONS"
                                              :rem 224
1070 PRINT:PRINTSPC(8)"2. REVIEW QUESTIONS"
                                              :rem 204
1080 PRINT:PRINTSPC(8)"3. CHANGE A QUESTION"
                                              :rem 144
1090 PRINT:PRINTSPC(8)"4. LOAD PREVIOUS DATA"
                                              :rem 106
1100 PRINT:PRINTSPC(8)"5. ADD TO TEST IN FILE"
                                              :rem 15
1110 PRINT:PRINTSPC(8)"6. INITIALIZE DISK":rem 203
1120 PRINT:PRINTSPC(8)"7. END"                :rem 135
1130 PRINT:PRINTSPC(5)"NUMBER?"               :rem 81
```

```
1140 GETG$:IFG$=""THEN1140                    :rem 187
1150 G=ASC(G$)-48:IFG<1ORG>8THEN1140           :rem 71
1160 ONGGOSUB240,900,840,60,1660,1750,1190:rem 202
1170 GOTO1010                                 :rem 196
1180 GOSUB2060                                 :rem 18
1190 POKE198,0:SYS198                         :rem 211
1200 REM PRINT JUSTIFY                         :rem 98
1210 PRINT                                     :rem 81
1220 IFLEN(S$)<40THENPRINTS$:GOTO1300           :rem 5
1230 X=40:Y=1                                 :rem 192
1240 X=X-1                                     :rem 18
1250 IFASC(MID$(S$,X,Y)+CHR$(0))<>32THEN1240
                                              :rem 208
1260 PRINTLEFT$(S$,X)                         :rem 241
1270 Z=LEN(S$)                                  :rem 8
1280 Z=Z-X                                     :rem 65
1290 PRINTRIGHT$(S$,Z)                         :rem 73
1300 RETURN                                   :rem 164
1310 PRINT"ENTRY TOO LONG: RE-PHRASE"          :rem 11
1320 H=1:FORT=1TO2000:NEXT:RETURN              :rem 87
1330 REM TEST TITLE FILE                      :rem 141
1340 PRINT:PRINT"HAS TEST TITLE FILE BEEN INITIATE
     D?(Y/N)":GOSUB2060                       :rem 187
1350 GETG$:IFG$=""THEN1350                    :rem 193
1360 IF G$="N"THEN1480                        :rem 143
1370 IFG$<>"Y"THEN1350                        :rem 212
1380 PRINT:PRINTSPC(17)"{RVS} WAIT "           :rem 74
1390 GOSUB2140:GOSUB1840:OPEN3,8,3,"TEST TITLES,S,
     R"                                       :rem 127
1400 X=0                                      :rem 138
1410 X=X+1:INPUT#3,M$(X)                      :rem 117
1420 IFST AND64THEN1440                        :rem 50
1430 GOTO1410                                 :rem 199
1440 CLOSE3:POKE198,0:GOSUB2140                :rem 90
1450 IFS1<>0THEN2100                          :rem 121
1460 PRINT"{CLR}":PRINTSPC(14)"TEST TITLES":PRINT
                                              :rem 178
1470 FORA=1TOX:PRINTA;". ";M$(A):NEXT:RETURN
                                              :rem 227
1480 REM INITIATE TEST FILE                   :rem 104
1490 IFX=15THENGOSUB1730                      :rem 210
1500 IF X=15THENX=1                            :rem 69
1510 IFX=1THEN1620                             :rem 20
1520 PRINT"{CLR}HAS FILE OF TEST NAMES BEEN STARTE
     D?"                                       :rem 97
1530 GETG$:IFG$=""THEN1530                    :rem 193
1540 IF G$="N"THEN1620                        :rem 139
1550 IFG$="Y"THENGOSUB1390                     :rem 27
1560 PRINTX+1". {RVS}"N$                        :rem 3
1570 PRINT"IS YOUR TITLE ORIGINAL?"            :rem 31
```

```
1580 GETG$:IFG$=""THEN1580                      :rem 203
1590 IF G$="Y"THEN1620                          :rem 155
1600 PRINT"ENTER NEW TITLE FOR TEST:"            :rem 99
1610 INPUTN$                                    :rem 202
1620 M$(X+1)=N$:GOSUB2060                       :rem 112
1630 PRINT:PRINTSPC(13)"{RVS} SAVING TITLE "
                                                 :rem 89
1640 GOSUB1840:OPEN3,8,3,"@0:TEST TITLES,S,W"
                                                :rem 171
1650 FORA=1TOX+1:PRINT#3,M$(A):NEXT:CLOSE3:POKE198
     ,0:GOSUB2140:RETURN                         :rem 99
1660 REM TEST ADDITION ROUTINE                   :rem 99
1670 CLR                                        :rem 175
1680 R=1:GOSUB70:R=0:GOTO1010                   :rem 222
1690 PRINT:PRINTTAB(5)"{RVS} PRESS SPACE BAR TO CO
     NTINUE {OFF}"                              :rem 191
1700 GETG$:IFASC(G$+CHR$(0))<>32THEN1700        :rem 242
1710 RETURN                                     :rem 169
1720 PRINT"{CLR}"                                :rem 46
1730 PRINTSPC(10)"{CLR}{10 DOWN}FILE FULL":rem 196
1740 FORT=1TO2000:NEXT:RETURN                   :rem 109
1750 PRINT"{CLR}{2 DOWN}{RVS}{2 SPACES}DO YOU WANT
      TO INITIALIZE A NEW DISK? ":PRINTTAB(17)"
     {RVS} (Y/N) "                              :rem 173
1760 GETG$:IFG$=""THEN1760                      :rem 203
1770 IFG$="Y"THENPRINT"TYPE GOTO 1790 AND PRESS RE
     TURN":END                                   :rem 64
1780 IFG$<>"N"THEN1760                          :rem 211
1790 IFG$="N"THENRETURN                         :rem 169
1800 END                                        :rem 160
1810 PRINT"{CLR}{5 DOWN}{6 SPACES}INSERT NEW DISK
     {SPACE}INTO DRIVE                          :rem 255
1820 PRINT"{4 DOWN}{2 SPACES}PRESS ANY KEY WHEN RE
     ADY TO PROCEED"                             :rem 10
1830 GETG$:IFG$=""THEN1830                      :rem 199
1840 OPEN15,8,15:PRINT#15,"I0:":CLOSE15         :rem 228
1850 PRINT"{CLR}{2 DOWN}{15 SPACES}{RVS} WARNING!!
     !"                                          :rem 95
1860 PRINT"{2 SPACES}{RVS} DISK IN DRIVE IS ABOUT
     {SPACE}TO BE ERASED!"                        :rem 4
1870 PRINT"{2 SPACES}{RVS}{9 SPACES}ARE YOU SURE?
     {SPACE}(Y/N){9 SPACES}"                     :rem 45
1880 GETG$:IFG$=""THEN1880                      :rem 209
1890 IFG$="Y"THEN1920                           :rem 161
1900 IFG$="N"THEN1010                           :rem 132
1910 GOTO1880                                   :rem 213
1920 PRINT"{CLR}{2 DOWN}ENTER DISKNAME";:INPUTDN$
                                                 :rem 87
1930 IFLEN(DN$)>15THENPRINT"{2 DOWN}NAME TOO LONG"
     :FORT=1TO1000:NEXT:GOTO1920                :rem 202
```

```
1940 PRINT"{2 DOWN}ENTER 2 CHARACTER DISK I.D.":IN
     PUTID$                                    :rem 141
1950 POKE53281,2:POKE53280,2:PRINT"{CLR}{5 DOWN}
     {10 SPACES}LAST CHANCE TO STOP!!!"     :rem 61
1960 PRINT:PRINT"{9 SPACES}PRESS ANY KEY TO STOP!!
     !":FORT=1TO1000                           :rem 217
1970 GETG$:IFG$<>""THEN1010                    :rem 255
1980 NEXT                                      :rem 17
1990 PRINT"{CLR}{4 DOWN}DISK IS BEING FORMATTED--W
     AIT"                                      :rem 220
2000 OPEN15,8,15:PRINT#15,"N0:"+DN$+","+ID$:rem 32
2010 INPUT#15,S1,S$,S2,S3:CLOSE15:IFS1<>0THEN2100:
     GOSUB2040                                 :rem 17
2020 PRINT"{CLR}{10 DOWN}{9 SPACES}DISK FORMATTED
     {SPACE}":FORT=1TO2000:NEXT                :rem 228
2030 POKE53280,13:POKE53281,5:GOTO1010     :rem 136
2040 FORT=1TO1000:NEXT:POKE53280,13:POKE53281,5:RE
     TURN                                      :rem 46
2050 REM SOUND ROUTINE                         :rem 90
2060 S=54272                                   :rem 92
2070 POKES,100:POKES+1,125:POKES+5,0:POKES+6,240:P
     OKES+24,15:POKES+4,17                     :rem 201
2080 FORT=0TO100:NEXT                          :rem 31
2090 POKES+4,0:RETURN                          :rem 34
2100 PRINT"DISK ERROR ";S1,S$,S2,S3        :rem 142
2110 PRINT:PRINT"CORRECT ERROR CONDITION AND TRY A
     GAIN"                                     :rem 209
2120 GOSUB1690                                 :rem 21
2130 GOTO1000                                  :rem 192
2140 OPEN15,8,15:INPUT#15,S1,S$,S2,S3:CLOSE15:IFS1
     <>0THEN2100                               :rem 93
2150 PRINT"DISK STATUS: "S$                    :rem 89
2160 RETURN                                    :rem 169
```

Program 2. Student Quiz

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
10 REM STUDENT QUIZ GENERATOR                  :rem 92
20 PRINT"{CLR}{WHT}":CLR:POKE53280,16:POKE53281,16
   :POKE808,225:POKE649,0:S=54727              :rem 236
30 DIMQ$(100),A$(100),B$(100),C$(100),D$(100),E$(1
   00),M$(15),A(100)                           :rem 128
40 GOSUB840:PRINT"{CLR}{N}":PRINTSPC(12)"{RVS}
   {2 SPACES}LOADING DATA{3 SPACES}":PRINT"{BLK}"
                                               :rem 22
45 GOSUB390:PRINT"{WHT}"                       :rem 149
50 FORX=1TOA:PRINTX". "M$(X):NEXT              :rem 26
```

```
60 PRINT"{DOWN}ENTER NUMBER OF TEST":POKE649,10:IN
   PUTX                                       :rem 159
70 IFX<1ORX>ATHENPRINT"INVALID RANGE":GOTO60
                                              :rem 128
80 N$=M$(X):POKE649,0:OPEN15,8,15:PRINT"{CLR}":OPE
   N2,8,2,+N$+" FILE,S,R"                     :rem 180
90 PRINT"{9 DOWN}{5 SPACES}LOADING ";N$;" QUIZ":PR
   INT"{BLK}"                                 :rem 97
100 X=0                                       :rem 86
110 X=X+1                                     :rem 219
120 INPUT#2,Q$(X):INPUT#2,A$(X):INPUT#2,B$(X)
                                              :rem 117
125 INPUT#2,C$(X):INPUT#2,D$(X):INPUT#2,E$(X)
                                              :rem 114
130 IFST AND64THEN150                         :rem 204
140 GOTO110                                   :rem 96
150 CLOSE2:POKE198,0:L=X:CLOSE15:GOSUB920:PRINT"
    {CLR}{WHT}"                               :rem 206
160 REM TEST ROUTINE                          :rem 225
170 Y=1:POKE649,10:GOSUB540                   :rem 75
180 FORN=1TOL-1:PRINT"{CLR}{DOWN}":PRINTTAB(20-LEN
    (N$)/2);N$                                :rem 102
190 S$=STR$(N)+". "+Q$(A(N)):PRINT:GOSUB450
                                              :rem 146
200 REM ANSWER CHOICES                        :rem 68
210 S$=A$(A(N)):GOSUB450:S$=B$(A(N)):GOSUB450:S$=C
    $(A(N)):GOSUB450                          :rem 225
220 S$=D$(A(N)):GOSUB450:S$=E$(A(N))          :rem 188
230 PRINT"{DOWN}ENTER LETTER OF MOST CORRECT ANSWE
    R:":POKE198,0                             :rem 160
240 INPUTF$                                   :rem 144
250 IFLEN(F$)<>1THENPRINT"ENTER ONE LETTER ONLY":G
    OTO240                                    :rem 102
260 IFASC(F$)<65ORASC(F$)>68THENPRINT"ANSWER MUST
    {SPACE}BE A,B,C, OR D":GOTO240            :rem 151
270 IFASC(F$)=ASC(S$)THENP=P+1                :rem 254
280 IFASC(F$)=ASC(S$)THENPRINTSPC(9)"{RVS}
    {2 SPACES}ANSWER IS CORRECT!! ":GOSUB1030
                                              :rem 215
290 IFASC(F$)<>ASC(S$)THEN:GOSUB1060:GOSUB820
                                              :rem 217
300 FORT=1TO4000:NEXT:NEXT                    :rem 149
310 N=N-1                                     :rem 203
320 S=INT(P/N*100+.5):PRINT"{CLR}{DOWN}YOU SCORED
    {SPACE}";S;" %"                           :rem 149
330 IFS>80ANDS<90THENPRINT"STUDY THIS SECTION AGAI
    N"                                        :rem 175
340 IFS>90ANDS<100THENPRINT"VERY GOOD, BUT MORE ST
    UDY WOULD HELP"                           :rem 153
```

```
350 IFS=100THENPRINT"EXCELLENT!!{2 SPACES}PERFECT
    {SPACE}SCORE!!"                              :rem 245
360 FORT=1TO3000:NEXT                            :rem 33
370 PRINT"{4 DOWN}ENTER RUN TO RE-START PROGRAM":P
    OKE808,237:END                               :rem 17
380 REM PRINT JUSTIFY                            :rem 58
390 OPEN15,8,15:OPEN3,8,3,"TEST TITLES,S,R":PRINT"
    {BLK}"                                       :rem 169
400 X=X+1                                        :rem 221
410 INPUT#3,M$(X)                                :rem 193
420 IFSTATUSAND64THEN440                         :rem 13
430 GOTO400                                      :rem 100
440 CLOSE3:POKE198,0:A=X:CLOSE15:PRINT"{CLR}{WHT}"
    :RETURN                                      :rem 139
450 IFLEN(S$)<40THENPRINTS$:GOTO510              :rem 171
460 X=40:Y=1                                     :rem 148
470 X=X-1                                        :rem 230
480 IFASC(MID$(S$,X,Y)+CHR$(0))<>32THEN470:rem 120
490 PRINTLEFT$(S$,X)                             :rem 197
500 Z=LEN(S$):Z=Z-X:PRINTRIGHT$(S$,Z)            :rem 58
510 RETURN                                       :rem 118
520 PRINT:PRINTSPC(14)"TEST TITLES":PRINT:FORA=1TO
    X:PRINTA;". ";M$(A)                          :rem 101
525 NEXT:RETURN                                  :rem 245
530 REM DISABLE CURSOR CONTROLS                  :rem 194
540 IFPEEK(830)=133THEN560                       :rem 215
550 FORI=828TO977:READA:POKEI,A:NEXT             :rem 34
560 SYS828:RETURN                                :rem 86
570 DATA169,000,133,252,169,080                  :rem 42
580 DATA133,251,169,164,133,002                  :rem 38
590 DATA169,083,141,036,003,169                  :rem 49
600 DATA003,141,037,003,096,152                  :rem 25
610 DATA072,138,072,165,252,208                  :rem 42
620 DATA007,032,116,003,169,000                  :rem 21
630 DATA133,253,166,253,189,000                  :rem 41
640 DATA002,133,254,198,252,230                  :rem 36
650 DATA253,104,170,104,168,165                  :rem 40
660 DATA254,096,160,000,132,252                  :rem 34
670 DATA165,002,032,210,255,169                  :rem 37
680 DATA157,032,210,255,032,228                  :rem 38
690 DATA255,240,251,164,252,133                  :rem 42
700 DATA254,169,032,032,210,255                  :rem 33
710 DATA169,157,032,210,255,165                  :rem 43
720 DATA254,201,013,240,043,201                  :rem 17
730 DATA020,208,013,192,000,240                  :rem 18
740 DATA211,136,169,157,032,210                  :rem 36
750 DATA255,076,118,003,041,127                  :rem 39
760 DATA201,032,144,196,196,251                  :rem 44
770 DATA240,192,165,254,153,000                  :rem 38
780 DATA002,032,210,255,169,000                  :rem 27
```

```
790 DATA133,212,200,076,118,003                :rem 30
800 DATA230,252,153,000,002,169                :rem 23
810 DATA032,032,210,255,096,013                :rem 27
820 PRINTSPC(10)"{RVS} SORRY ANSWER IS WRONG "
                                               :rem 45
830 PRINT"{DOWN}CORRECT CHOICE IS: ";S$:RETURN
                                               :rem 92
840 PRINT"{CLR}{5 DOWN}":PRINTSPC(13)"{RVS} QUIZ M
    ASTER ":POKE53272,23                       :rem 31
850 PRINT"{DOWN}{4 SPACES}THESE TESTS ARE MULTIPLE
     CHOICE."                                  :rem 40
860 PRINT"ENTER THE BEST ANSWER FROM THE CHOICES"
                                               :rem 95
870 PRINT"GIVEN."                              :rem 23
880 PRINT"{DOWN}{4 SPACES}ENTER THE NUMBER OF THE
    {SPACE}TEST YOU "                         :rem 221
890 PRINT"HAVE BEEN ASSIGNED WHEN THE PROGRAM "
                                               :rem 41
900 PRINT"CALLS FOR IT."                      :rem 139
910 FORT=1TO6000:NEXT:RETURN                   :rem 63
920 REM RANDOM GEN.                            :rem 72
930 PRINT"{CLR}{DOWN}WAIT-- PREPARING QUIZ":PRINT"
    {BLK}"                                     :rem 44
940 FORX=1TOL                                  :rem 57
950 A(X)=INT(RND(.)*L)+1                       :rem 54
960 IFX=1THEN1000                             :rem 228
970 FORY=1TOX-1                               :rem 167
980 IFA(Y)=A(X)THEN950                         :rem 15
990 NEXTY                                      :rem 58
1000 NEXTX                                     :rem 88
1010 PRINT"{WHT}":RETURN                      :rem 178
1020 REM CORRECT ANSWER SOUND                  :rem 18
1030 S=54272:POKES,150:POKES+1,100:POKES+5,0:POKES
     +6,240:POKES+24,15:POKES+4,17            :rem 144
1040 FORT=0TO200:NEXT:POKES+4,0:RETURN        :rem 172
1050 REM WRONG ANSWER SOUND                   :rem 144
1060 S=54272:POKES,150:POKES+1,5:POKES+5,0:POKES+6
     ,240:POKES+24,15:POKES+4,17               :rem 55
1070 FORT=0TO200:NEXT:POKES+4,0:RETURN        :rem 175
```

Making Calendars

Paul C. Liu

Put your printer to good use by making a full set of calendars. These three programs will give you a wall calendar, an appointment calendar, and one for the year at a glance. For a 1515, 1525, 1526, or MPS-801 printer.

Your Days Are Numbered

A practical use for a computer is making your own calendars. Here are three calendar-making programs which require the use of a printer. The programs are written entirely in BASIC without PEEKs or POKEs, so they can be easily adapted for other computers or non-Commodore printers.

In calendar making, it is essential to know the correct day of the week for any given date. If we let D1 be the day of the week (for Sunday D1=1, for Monday D1=2, and so on), and let M, D, and Y be the month, day, and year, respectively, D1 can be calculated by:

```
D1=INT(2.6*(M-2)-0.2)+D+Y-1900+INT((Y-1900)/4)
D1=D1+INT(19/4)-2*19
D1=D1-INT(D1/7)*7+1
```

Two modifications have to be used with the above formula. For M equal to 1 or 2, we have to add 12 and subtract 1 from Y. In other words, we consider the months of January and February as the thirteenth and fourteenth months of the previous year. In addition, for M equal to 4 or 9, the calculated D1 has to be increased by 1.

Good for More Than 100 Years

This algorithm performs flawlessly for the twentieth and twenty-first centuries, up to the year 2100. If you really want to be meticulous beyond that, you can make further modifications by reducing D1 by 1 after March 2100, and repeating that every 100 years. You must do this because the century years like 2100 and 2200 which are not divisible by 400 are not leap years, but the algorithm treats them as if they were.

The programs contain modifications like the one above to make them accurate for the next five centuries, provided, of course, that the current calendar system is not reformed. (The last calendar reform was 1752.)

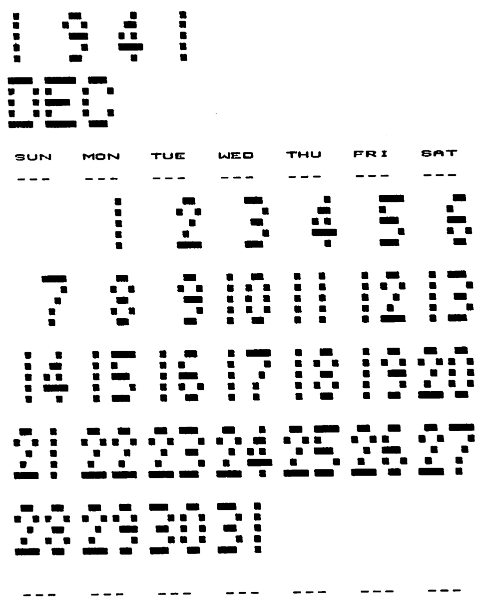
Once we know the day of the week for the given date, especially the first day of the month, the rest of the calendar-making task is just a matter of setting up and getting the proper format and display.

A Monthly Calendar

After you load one of the programs, type RUN, and press RETURN, the computer will briefly explain what the program is for and will then ask you to input the month and year of the calendar you wish to see. The numbers should be separated by a comma, and the year should be the full four digits (1985, not 85). Then the monthly calendar of your choice will be displayed on the screen.

Program 1 will give you a copy of a monthly calendar by printing it on your printer. This is a long program because it contains a set of enlarged numbers and characters, together with a blank subroutine to use them. The result is a calendar that you can hang on the wall. *If you have a 1526 printer and would like a neater printout, try executing the following commands before running Program 1:*

OPEN 6,4,6:PRINT#6, CHR$(18)
CLOSE6



Program 2 also gives you a printed monthly calendar, but in a different format. The program tabulates the days of the month as a list. It can serve as an appointment calendar for your desk, with room for short notes each day. Along with the regular date, you are told what day of the year it is.

```
DECEMBER   1941
MONDAY      1  ( 335 )
TUESDAY     2  ( 336 )
WEDNESDAY   3  ( 337 )
THURSDAY    4  ( 338 )
FRIDAY      5  ( 339 )
SATURDAY    6  ( 340 )
[redacted] ( 341 )
MONDAY      8  ( 342 )
TUESDAY     9  ( 343 )
WEDNESDAY  10  ( 344 )
THURSDAY   11  ( 345 )
FRIDAY     12  ( 346 )
SATURDAY   13  ( 347 )
[redacted] ( 348 )
MONDAY     15  ( 349 )
TUESDAY    16  ( 350 )
WEDNESDAY  17  ( 351 )
THURSDAY   18  ( 352 )
FRIDAY     19  ( 353 )
SATURDAY   20  ( 354 )
[redacted] ( 355 )
MONDAY     22  ( 356 )
TUESDAY    23  ( 357 )
WEDNESDAY  24  ( 358 )
THURSDAY   25  ( 359 )
FRIDAY     26  ( 360 )
SATURDAY   27  ( 361 )
[redacted] ( 362 )
MONDAY     29  ( 363 )
TUESDAY    30  ( 364 )
WEDNESDAY  31  ( 365 )
```

A Year on One Sheet

Program 3 will give you all 12 months of the year printed on one sheet. The message HAPPY NEW YEAR is at the top of the calendar, but you can put a different short message there by modifying the text in line 7.

HAPPY NEW YEAR 1941

JANUARY

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

FEBRUARY

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	

MARCH

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

APRIL

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

MAY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

JUNE

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

JULY

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

AUGUST

S	M	T	W	T	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

SEPTEMBER

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

OCTOBER

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

NOVEMBER

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

DECEMBER

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

In all three programs, after you input the month and year as requested, the computer prompts you to turn on the printer. Before you do this, you should set the perforation of the printing paper over the starting postion of the printhead so that the calendar will appear entirely on one sheet of paper. The programs are written for the Commodore 1515, 1525, 1526, and MPS-801 printers. Other printers may require modifications to the programs.

Program 1. Monthly Calendar

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
1 GOTO10                                        :rem 203
5 E1=1:E2=1:E3=1:E4=1:E5=1:E6=1:E7=1            :rem 226
6 GOSUB1109:D8=D7-1:RETURN                      :rem 103
10 OPEN1,4:SYS65517:A=PEEK(781):IFA=40THENPOKE5328
   1,1                                          :rem 156
20 GOSUB4000:GOSUB3200:PRINT#1,""               :rem 176
30 ONM0GOSUB3010,3020,3030,3040,3050,3060,3070,308
   0,3090,3100,3110,3120                         :rem 56
40 PRINT#1,"":PRINT#1,"":GOSUB1610:GOSUB1650:GOSUB
   1660                                         :rem 207
80 OND9GOSUB1811,1821,1831,1841,1851,1861,1871
                                                :rem 172
99 PRINT#1,"":PRINT#1,""                         :rem 78
100 G1=D8                                       :rem 194
105 G=G1:GOSUB1720:D1=D:E1=E                    :rem 120
110 G2=G+1:G=G2:GOSUB1720:D2=D:E2=E              :rem 10
115 G3=G+1:G=G3:GOSUB1720:D3=D:E3=E              :rem 19
120 G4=G+1:G=G4:GOSUB1720:D4=D:E4=E              :rem 19
125 G5=G+1:G=G5:GOSUB1720:D5=D:E5=E              :rem 28
130 G6=G+1:G=G6:GOSUB1720:D6=D:E6=E              :rem 28
135 G7=G+1:G=G7:GOSUB1720:D7=D:E7=E              :rem 37
140 G1=G7+1:GOSUB1109:PRINT#1,"":PRINT#1,"":IFG1<=
    E9THEN105                                   :rem 188
155 PRINT#1,""                                  :rem 236
1000 GOTO5000                                   :rem 191
1109 GOSUB2000:X=E1:X1=D1:GOSUB11000            :rem 115
1120 X=E2:X1=D2:GOSUB11000                      :rem 242
1130 X=E3:X1=D3:GOSUB11000                      :rem 245
1140 X=E4:X1=D4:GOSUB11000                      :rem 248
1150 X=E5:X1=D5:GOSUB11000                      :rem 251
1160 X=E6:X1=D6:GOSUB11000                      :rem 254
1170 X=E7:X1=D7:FL=1:GOSUB11000                  :rem 59
1209 GOSUB2000:X=E1:X1=D1:GOSUB12000            :rem 117
1220 X=E2:X1=D2:GOSUB12000                      :rem 244
1230 X=E3:X1=D3:GOSUB12000                      :rem 247
1240 X=E4:X1=D4:GOSUB12000                      :rem 250
1250 X=E5:X1=D5:GOSUB12000                      :rem 253
1260 X=E6:X1=D6:GOSUB12000                        :rem 0
1270 X=E7:X1=D7:FL=1:GOSUB12000                  :rem 61
1309 GOSUB2000:X=E1:X1=D1:GOSUB13000            :rem 119
```

```
1320 X=E2:X1=D2:GOSUB13000                          :rem 246
1330 X=E3:X1=D3:GOSUB13000                          :rem 249
1340 X=E4:X1=D4:GOSUB13000                          :rem 252
1350 X=E5:X1=D5:GOSUB13000                          :rem 255
1360 X=E6:X1=D6:GOSUB13000                            :rem 2
1370 X=E7:X1=D7:FL=1:GOSUB13000                      :rem 63
1409 GOSUB2000:X=E1:X1=D1:GOSUB14000                :rem 121
1420 X=E2:X1=D2:GOSUB14000                          :rem 248
1430 X=E3:X1=D3:GOSUB14000                          :rem 251
1440 X=E4:X1=D4:GOSUB14000                          :rem 254
1450 X=E5:X1=D5:GOSUB14000                            :rem 1
1460 X=E6:X1=D6:GOSUB14000                            :rem 4
1470 X=E7:X1=D7:FL=1:GOSUB14000                      :rem 65
1509 GOSUB2000:X=E1:X1=D1:GOSUB15000                :rem 123
1520 X=E2:X1=D2:GOSUB15000                          :rem 250
1530 X=E3:X1=D3:GOSUB15000                          :rem 253
1540 X=E4:X1=D4:GOSUB15000                            :rem 0
1550 X=E5:X1=D5:GOSUB15000                            :rem 3
1560 X=E6:X1=D6:GOSUB15000                            :rem 6
1570 X=E7:X1=D7:FL=1:GOSUB15000                      :rem 67
1600 RETURN                                         :rem 167
1610 PRINT#1,"{5 SPACES}";:PRINT#1,CHR$(14)"SUN";:
     PRINT#1,CHR$(15)"{5 SPACES}";                   :rem 69
1611 PRINT#1,CHR$(14)"MON";:PRINT#1,CHR$(15)"
     {5 SPACES}";                                   :rem 116
1612 PRINT#1,CHR$(14)"TUE";:PRINT#1,CHR$(15)"
     {5 SPACES}";                                   :rem 121
1613 PRINT#1,CHR$(14)"WED";:PRINT#1,CHR$(15)"
     {5 SPACES}";                                   :rem 108
1614 PRINT#1,CHR$(14)"THU";:PRINT#1,CHR$(15)"
     {5 SPACES}";                                   :rem 126
1615 PRINT#1,CHR$(14)"FRI";:PRINT#1,CHR$(15)"
     {5 SPACES}";                                   :rem 111
1616 PRINT#1,CHR$(14)"SAT":PRINT#1,CHR$(15)" "
                                                      :rem 1
1620 PRINT#1,"{5 SPACES}";:PRINT#1,CHR$(14)"---";:
     PRINT#1,CHR$(15)"{5 SPACES}";                  :rem 215
1621 PRINT#1,CHR$(14)"---";:PRINT#1,CHR$(15)"
     {5 SPACES}";                                    :rem 18
1622 PRINT#1,CHR$(14)"---";:PRINT#1,CHR$(15)"
     {5 SPACES}";                                    :rem 19
1623 PRINT#1,CHR$(14)"---";:PRINT#1,CHR$(15)"
     {5 SPACES}";                                    :rem 20
1624 PRINT#1,CHR$(14)"---";:PRINT#1,CHR$(15)"
     {5 SPACES}";                                    :rem 21
1625 PRINT#1,CHR$(14)"---";:PRINT#1,CHR$(15)"
     {5 SPACES}";                                    :rem 22
1626 PRINT#1,CHR$(14)"---":PRINT#1,CHR$(15)" ":RET
     URN                                            :rem 187
```

```
1650 IFMØ=1ORMØ=3ORMØ=5ORMØ=7ORMØ=8ORMØ=1ØORMØ=12T
     HENE9=31                                   :rem 81
1652 IFMØ=4ORMØ=6ORMØ=9ORMØ=11THENE9=3Ø         :rem 122
1654 IFMØ=2ANDY/4<>INT(Y/4)THENE9=28            :rem 16Ø
1656 IFMØ=2ANDY/4=INT(Y/4)THENE9=29             :rem 1Ø2
1658 RETURN                                     :rem 18Ø
166Ø IFMØ=1THENMØ=13:Y=Y-1:GOTO167Ø             :rem 92
1665 IFMØ=2THENMØ=14:Y=Y-1                      :rem 34
167Ø M=MØ-2                                     :rem 52
1675 D9=INT(2.6*M-Ø.2)+D+Y-19ØØ+INT((Y-19ØØ)/4)
                                                :rem 232
168Ø D9=D9+INT(19/4)-2*19                       :rem 45
1685 D9=D9-INT(D9/7)*7+1                        :rem 2Ø
169Ø IFMØ=4ORMØ=9THEND9=D9+1                    :rem 163
1695 IFMØ=13THENMØ=1:Y=Y+1:GOTO171Ø             :rem 93
17ØØ IFMØ=14THENMØ=2:Y=Y+1:D9=D9+1              :rem 227
17Ø5 IFD9=8THEND9=1                             :rem 1Ø4
171Ø IF(Y=21ØØANDMØ>=3)OR(Y>21ØØ)THEND9=D9-1:IFD9=
     ØTHEND9=7                                  :rem 227
1711 IF(Y=22ØØANDMØ>=3)OR(Y>22ØØ)THEND9=D9-1:IFD9=
     ØTHEND9=7                                  :rem 23Ø
1712 IF(Y=23ØØANDMØ>=3)OR(Y>23ØØ)THEND9=D9-1:IFD9=
     ØTHEND9=7                                  :rem 233
1715 RETURN                                     :rem 174
172Ø IFG>E9THENGOTO174Ø                         :rem 144
1722 IFG<1ØTHENGOTO1742                         :rem 117
1726 IFG>=1ØANDG<2ØTHENGOTO1746                 :rem 116
1728 IFG>=2ØANDG<3ØTHENGOTO1748                 :rem 122
173Ø IFG>=3ØTHENGOTO175Ø                        :rem 18Ø
174Ø D=1:E=1:GOTO1755                           :rem 176
1742 D=G+2:E=1:GOTO1755                         :rem 37
1746 D=G-1Ø+2:E=2:GOTO1755                      :rem 184
1748 D=G-2Ø+2:E=3:GOTO1755                      :rem 188
175Ø D=G-3Ø+2:E=4                               :rem 114
1755 RETURN                                     :rem 178
1811 D1=1:D2=3:D3=4:D4=5:D5=6:D6=7:D7=8:GOSUB5:RET
     URN                                        :rem 149
1821 D1=1:D2=1:D3=3:D4=4:D5=5:D6=6:D7=7:GOSUB5:RET
     URN                                        :rem 143
1831 D1=1:D2=1:D3=1:D4=3:D5=4:D6=5:D7=6:GOSUB5:RET
     URN                                        :rem 138
1841 D1=1:D2=1:D3=1:D4=1:D5=3:D6=4:D7=5:GOSUB5:RET
     URN                                        :rem 134
1851 D1=1:D2=1:D3=1:D4=1:D5=1:D6=3:D7=4:GOSUB5:RET
     URN                                        :rem 131
1861 D1=1:D2=1:D3=1:D4=1:D5=1:D6=1:D7=3:GOSUB5:RET
     URN                                        :rem 129
1871 D1=3:D2=4:D3=5:D4=6:D5=7:D6=8:D7=9:GOSUB5:RET
     URN                                        :rem 163
2ØØØ PRINT#1,"{4 SPACES}";:RETURN               :rem 1Ø4
```

```
2001 PRINT#1," {2 +} ";:RETURN                :rem 181
2002 PRINT#1,"{+}{2 SPACES}{+}";:RETURN     :rem 182
2003 PRINT#1,"{+}{2 SPACES}{+}";:RETURN     :rem 183
2004 PRINT#1,"{+}{2 SPACES}{+}";:RETURN     :rem 184
2005 PRINT#1," {2 +} ";:RETURN                :rem 185
2011 PRINT#1," {+}{2 SPACES}";:RETURN         :rem 16
2012 PRINT#1," {+}{2 SPACES}";:RETURN         :rem 17
2013 PRINT#1," {+}{2 SPACES}";:RETURN         :rem 18
2014 PRINT#1," {+}{2 SPACES}";:RETURN         :rem 19
2015 PRINT#1," {+}{2 SPACES}";:RETURN         :rem 20
2021 PRINT#1," {2 +} ";:RETURN                :rem 183
2022 PRINT#1,"{+}{2 SPACES}{+}";:RETURN     :rem 184
2023 PRINT#1,"{2 SPACES}{+} ";:RETURN         :rem 19
2024 PRINT#1," {+}{2 SPACES}";:RETURN         :rem 20
2025 PRINT#1,"{4 +}";:RETURN                   :rem 7
2031 PRINT#1,"{3 +} ";:RETURN                  :rem 94
2032 PRINT#1,"{3 SPACES}{+}";:RETURN          :rem 19
2033 PRINT#1," {2 +} ";:RETURN                :rem 186
2034 PRINT#1,"{3 SPACES}{+}";:RETURN          :rem 21
2035 PRINT#1,"{3 +} ";:RETURN                  :rem 98
2041 PRINT#1,"{2 SPACES}{+} ";:RETURN         :rem 19
2042 PRINT#1," {2 +} ";:RETURN                :rem 186
2043 PRINT#1,"{+} {+} ";:RETURN               :rem 187
2044 PRINT#1,"{4 +}";:RETURN                   :rem 8
2045 PRINT#1,"{2 SPACES}{+} ";:RETURN         :rem 23
2051 PRINT#1,"{4 +}";:RETURN                   :rem 6
2052 PRINT#1,"{+}{3 SPACES}";:RETURN          :rem 21
2053 PRINT#1,"{3 +} ";:RETURN                  :rem 98
2054 PRINT#1,"{3 SPACES}{+}";:RETURN          :rem 23
2055 PRINT#1,"{3 +} ";:RETURN                  :rem 100
2061 PRINT#1," {2 +} ";:RETURN                :rem 187
2062 PRINT#1,"{+}{3 SPACES}";:RETURN          :rem 22
2063 PRINT#1,"{3 +} ";:RETURN                  :rem 99
2064 PRINT#1,"{+}{2 SPACES}{+}";:RETURN     :rem 190
2065 PRINT#1," {2 +} ";:RETURN                :rem 191
2071 PRINT#1,"{4 +}";:RETURN                   :rem 8
2072 PRINT#1,"{3 SPACES}{+}";:RETURN          :rem 23
2073 PRINT#1,"{2 SPACES}{+} ";:RETURN         :rem 24
2074 PRINT#1," {+}{2 SPACES}";:RETURN         :rem 25
2075 PRINT#1," {+}{2 SPACES}";:RETURN         :rem 26
2081 PRINT#1," {2 +} ";:RETURN                :rem 189
2082 PRINT#1,"{+}{2 SPACES}{+}";:RETURN     :rem 190
2083 PRINT#1," {2 +} ";:RETURN                :rem 191
2084 PRINT#1,"{+}{2 SPACES}{+}";:RETURN     :rem 192
2085 PRINT#1," {2 +} ";:RETURN                :rem 193
2091 PRINT#1," {2 +} ";:RETURN                :rem 190
2092 PRINT#1,"{+}{2 SPACES}{+}";:RETURN     :rem 191
2093 PRINT#1," {3 +}";:RETURN                  :rem 102
2094 PRINT#1,"{3 SPACES}{+}";:RETURN          :rem 27
2095 PRINT#1," {2 +} ";:RETURN                :rem 194
```

```
2111 PRINT#1,"{2 SPACES}[+] ";:RETURN          :rem 17
2112 PRINT#1,"{2 SPACES}[+] ";:RETURN          :rem 18
2113 PRINT#1,"{2 SPACES}[+] ";:RETURN          :rem 19
2114 PRINT#1,"{2 SPACES}[+] ";:RETURN          :rem 20
2115 PRINT#1,"{2 SPACES}[+] ";:RETURN          :rem 21
3010 GOSUB2000:PRINT#1," [3 +]{3 SPACES}[3 +]
     {2 SPACES}[+]{3 SPACES}[+]"                :rem 193
3011 GOSUB2000:PRINT#1,"{2 SPACES}[+]{3 SPACES}[+]
     {3 SPACES}[+] [2 +]{2 SPACES}[+]"         :rem 118
3012 GOSUB2000:PRINT#1,"{2 SPACES}[+]{3 SPACES}[+]
     {3 SPACES}[+] [+] [+] [+]"                :rem 119
3013 GOSUB2000:PRINT#1,"[+] [+]{3 SPACES}[5 +] [+]
     {2 SPACES}[2 +]"                           :rem 16
3014 GOSUB2000:PRINT#1,"[3 +]{3 SPACES}[+]
     {3 SPACES}[+] [+]{3 SPACES}[+]"            :rem 31
3015 RETURN                                     :rem 169
3020 GOSUB2000:PRINT#1,"[5 +] [5 +] [4 +] "
                                                :rem 166
3021 GOSUB2000:PRINT#1,"[+]{5 SPACES}[+]{5 SPACES}
     [+]{3 SPACES}[+]"                          :rem 43
3022 GOSUB2000:PRINT#1,"[3 +]{3 SPACES}[4 +]
     {2 SPACES}[4 +] "                          :rem 182
3023 GOSUB2000:PRINT#1,"[+]{5 SPACES}[+]{5 SPACES}
     [+]{3 SPACES}[+]"                          :rem 45
3024 GOSUB2000:PRINT#1,"[+]{5 SPACES}[5 +] [4 +] "
                                                :rem 18
3025 RETURN                                     :rem 170
3030 GOSUB2000:PRINT#1,"[+]{3 SPACES}[+]{2 SPACES}
     [3 +]{2 SPACES}[4 +] "                     :rem 105
3031 GOSUB2000:PRINT#1,"[2 +] [2 +] [+]{3 SPACES}
     [+] [+]{3 SPACES}[+]"                      :rem 196
3032 GOSUB2000:PRINT#1,"[+] [+] [+] [+]{3 SPACES}
     [+] [4 +] "                                :rem 107
3033 GOSUB2000:PRINT#1,"[+] [+] [+] [5 +] [+]
     {2 SPACES}[+] "                            :rem 18
3034 GOSUB2000:PRINT#1,"[+]{3 SPACES}[+] [+]
     {3 SPACES}[+] [+]{3 SPACES}[+]"            :rem 123
3035 RETURN                                     :rem 171
3040 GOSUB2000:PRINT#1," [3 +]{2 SPACES}[4 +]
     {2 SPACES}[4 +] "                          :rem 182
3041 GOSUB2000:PRINT#1,"[+]{3 SPACES}[+] [+]
     {3 SPACES}[+] [+]{3 SPACES}[+]"            :rem 121
3042 GOSUB2000:PRINT#1,"[+]{3 SPACES}[+] [4 +]
     {2 SPACES}[4 +] "                          :rem 18
3043 GOSUB2000:PRINT#1,"[5 +] [+]{5 SPACES}[+]
     {2 SPACES}[+] "                            :rem 199
3044 GOSUB2000:PRINT#1,"[+]{3 SPACES}[+] [+]
     {5 SPACES}[+]{3 SPACES}[+]"                :rem 214
3045 RETURN                                     :rem 172
```

```
3050 GOSUB2000:PRINT#1,"{+}{3 SPACES}{+}{2 SPACES}
     {3 +}{2 SPACES}{+}{3 SPACES}{+}"          :rem 31
3051 GOSUB2000:PRINT#1,"{2 +} {2 +} {+}{3 SPACES}
     {+} {+}{3 SPACES}{+}"                     :rem 198
3052 GOSUB2000:PRINT#1,"{+} {+} {+} {+}{3 SPACES}
     {+}{2 SPACES}{+} {+} "                    :rem 33
3053 GOSUB2000:PRINT#1,"{+} {+} {+} {5 +}
     {3 SPACES}{+}{2 SPACES}"                  :rem 110
3054 GOSUB2000:PRINT#1,"{+}{3 SPACES}{+} {+}
     {3 SPACES}{+}{3 SPACES}{+}{2 SPACES}":rem 215
3055 RETURN                                    :rem 173
3060 GOSUB2000:PRINT#1," {3 +}{2 SPACES}{+}
     {3 SPACES}{+} {+}{3 SPACES}{+}"           :rem 32
3061 GOSUB2000:PRINT#1,"{2 SPACES}{+}{3 SPACES}{+}
     {3 SPACES}{+} {2 +}{2 SPACES}{+}"         :rem 123
3062 GOSUB2000:PRINT#1,"{2 SPACES}{+}{3 SPACES}{+}
     {3 SPACES}{+} {+} {+} {+}"                :rem 124
3063 GOSUB2000:PRINT#1,"{+} {+}{3 SPACES}{+}
     {3 SPACES}{+} {+}{2 SPACES}{2 +}"         :rem 35
3064 GOSUB2000:PRINT#1,"{3 +}{4 SPACES}{3 +}
     {2 SPACES}{+}{3 SPACES}{+}"               :rem 202
3065 RETURN                                    :rem 174
3070 GOSUB2000:PRINT#1," {3 +}{2 SPACES}{+}
     {3 SPACES}{+} {+}{4 SPACES}"              :rem 123
3071 GOSUB2000:PRINT#1,"{2 SPACES}{+}{3 SPACES}{+}
     {3 SPACES}{+} {+}{4 SPACES}"              :rem 48
3072 GOSUB2000:PRINT#1,"{2 SPACES}{+}{3 SPACES}{+}
     {3 SPACES}{+} {+}{4 SPACES}"              :rem 49
3073 GOSUB2000:PRINT#1,"{+} {+}{3 SPACES}{+}
     {3 SPACES}{+} {+}{4 SPACES}"              :rem 216
3074 GOSUB2000:PRINT#1,"{3 +}{4 SPACES}{3 +}
     {2 SPACES}{5 +}"                          :rem 189
3075 RETURN                                    :rem 175
3080 GOSUB2000:PRINT#1," {3 +}{2 SPACES}{+}
     {3 SPACES}{+}{2 SPACES}{3 +} "            :rem 200
3081 GOSUB2000:PRINT#1,"{+}{3 SPACES}{+} {+}
     {3 SPACES}{+} {+}{4 SPACES}"              :rem 215
3082 GOSUB2000:PRINT#1,"{+}{3 SPACES}{+} {+}
     {3 SPACES}{+} {+}{2 SPACES}{2 +}"         :rem 36
3083 GOSUB2000:PRINT#1,"{5 +} {+}{3 SPACES}{+} {+}
     {3 SPACES}{+}"                            :rem 113
3084 GOSUB2000:PRINT#1,"{+}{3 SPACES}{+}{2 SPACES}
     {3 +}{3 SPACES}{3 +} "                    :rem 204
3085 RETURN                                    :rem 176
3090 GOSUB2000:PRINT#1," {4 +} {5 +} {4 +} ":rem 7
3091 GOSUB2000:PRINT#1,"{+}{5 SPACES}{+}{5 SPACES}
     {+}{3 SPACES}{+}"                         :rem 50
3092 GOSUB2000:PRINT#1," {3 +}{2 SPACES}{4 +}
     {2 SPACES}{4 +} "                         :rem 189
```

```
3093 GOSUB2000:PRINT#1,"{4 SPACES}E+3 E+3
     {5 SPACES}E+3{4 SPACES}"                  :rem 142
3094 GOSUB2000:PRINT#1,"E4 +3{2 SPACES}E5 +3 E+3
     {4 SPACES}"                               :rem 25
3095 RETURN                                    :rem 177
3100 GOSUB2000:PRINT#1," E3 +3{3 SPACES}E3 +3
     {2 SPACES}E5 +3"                          :rem 179
3101 GOSUB2000:PRINT#1,"E+3{3 SPACES}E+3 E+3
     {3 SPACES}E+3{3 SPACES}E+3{2 SPACES}":rem 208
3102 GOSUB2000:PRINT#1,"E+3{3 SPACES}E+3 E+3
     {7 SPACES}E+3{2 SPACES}"                  :rem 43
3103 GOSUB2000:PRINT#1,"E+3{3 SPACES}E+3 E+3
     {3 SPACES}E+3{3 SPACES}E+3{2 SPACES}":rem 210
3104 GOSUB2000:PRINT#1," E3 +3{3 SPACES}E3 +3
     {4 SPACES}E+3{2 SPACES}"                  :rem 31
3105 RETURN                                    :rem 169
3110 GOSUB2000:PRINT#1,"E+3{3 SPACES}E+3{2 SPACES}
     E3 +3{2 SPACES}E+3{3 SPACES}E+3"          :rem 28
3111 GOSUB2000:PRINT#1,"E2 +3{2 SPACES}E+3 E+3
     {3 SPACES}E+3 E+3{3 SPACES}E+3"           :rem 29
3112 GOSUB2000:PRINT#1,"E+3 E+3 E+3 E+3{3 SPACES}
     E+3 E+3{3 SPACES}E+3"                     :rem 30
3113 GOSUB2000:PRINT#1,"E+3{2 SPACES}E2 +3 E+3
     {3 SPACES}E+3{2 SPACES}E+3 E+3 "          :rem 31
3114 GOSUB2000:PRINT#1,"E+3{3 SPACES}E+3{2 SPACES}
     E3 +3{4 SPACES}E+3{2 SPACES}"             :rem 122
3115 RETURN                                    :rem 170
3120 GOSUB2000:PRINT#1,"E4 +3{2 SPACES}E5 +3
     {2 SPACES}E3 +3 "                         :rem 91
3121 GOSUB2000:PRINT#1,"E+3{3 SPACES}E+3 E+3
     {5 SPACES}E+3{3 SPACES}E+3"               :rem 210
3122 GOSUB2000:PRINT#1,"E+3{3 SPACES}E+3 E4 +3
     {2 SPACES}E+3{4 SPACES}"                  :rem 31
3123 GOSUB2000:PRINT#1,"E+3{3 SPACES}E+3 E+3
     {5 SPACES}E+3{3 SPACES}E+3"               :rem 212
3124 GOSUB2000:PRINT#1,"E4 +3{2 SPACES}E5 +3
     {2 SPACES}E3 +3 "                         :rem 95
3125 RETURN                                    :rem 171
3200 I1=INT(Y/1000):J1=Y-I1*1000:I2=INT(J1/100):J2
     =J1-I2*100:I3=INT(J2/10)                  :rem 83
3210 I4=J2-I3*10                               :rem 48
3211 IFI2=0THENI2=10                           :rem 134
3212 IFI3=0THENI3=10                           :rem 137
3213 IFI4=0THENI4=10                           :rem 140
3214 GOSUB2000:X=I1:GOSUB6000:GOSUB2000:X=I2:GOSUB
     6000:GOSUB2000:X=I3:GOSUB6000             :rem 98
3215 GOSUB2000:X=I4:FL=1:GOSUB6000             :rem 19
3314 GOSUB2000:X=I1:GOSUB7000:GOSUB2000:X=I2:GOSUB
     7000:GOSUB2000:X=I3:GOSUB7000             :rem 102
```

```
3315 GOSUB2000:X=I4:FL=1:GOSUB7000             :rem 21
3414 GOSUB2000:X=I1:GOSUB8000:GOSUB2000:X=I2:GOSUB
     8000:GOSUB2000:X=I3:GOSUB8000            :rem 106
3415 GOSUB2000:X=I4:FL=1:GOSUB8000             :rem 23
3514 GOSUB2000:X=I1:GOSUB9000:GOSUB2000:X=I2:GOSUB
     9000:GOSUB2000:X=I3:GOSUB9000            :rem 110
3515 GOSUB2000:X=I4:FL=1:GOSUB9000             :rem 25
3614 GOSUB2000:X=I1:GOSUB10000:GOSUB2000:X=I2:GOSU
     B10000:GOSUB2000:X=I3                     :rem 60
3615 GOSUB10000:GOSUB2000:X=I4:FL=1:GOSUB10000:RET
     URN                                        :rem 7
4000 PRINT"{CLR}{DOWN}{2 SPACES}THIS IS A PROGRAM"
     :PRINT"{5 RIGHT}TO PRINT A"              :rem 115
4020 PRINT"{2 SPACES}{PUR}MONTHLY CALENDAR{BLU}":P
     RINT"{3 RIGHT}ON THE PRINTER"            :rem 187
4030 PRINT"{DOWN}{2 RIGHT}PLEASE TYPE IN THE":PRIN
     T"{3 RIGHT}{RED}MONTH{BLU} AND {RED}YEAR{BLU}
     "                                        :rem 185
4035 PRINT" THAT YOU WISH TO SEE":PRINT"{2 SPACES}
     (EXAMPLE: {RED}12,1983{BLU}){PUR}{DOWN}":PRIN
     TTAB(5);                                 :rem 211
4060 INPUTM0,Y                                 :rem 92
4080 PRINT"{2 DOWN}{2 SPACES}{BLU}THANK YOU! NOW--
     ":PRINT" PLEASE {PUR}TURN ON{BLU} THE" :rem 7
4085 PRINT"PRINTER AND THEN TYPE":PRINTTAB(8)"
     {PUR}OK{DOWN}":INPUTR$                   :rem 252
4110 IFR$<>"OK"THEN4080                        :rem 30
4130 PRINT"{BLU}PRINTING{DOWN}":FORI=1TO800:NEXT:R
     ETURN                                    :rem 218
4999 PRINT#1,CHR$(15)" "                      :rem 232
5000 GOSUB1620                                 :rem 14
5001 CLOSE1:END                               :rem 126
6000 ONXGOSUB2011,2021,2031,2041,2051,2061,2071,20
     81,2091,2001                             :rem 146
6010 IFFL<>1THENPRINT#1," ";:RETURN           :rem 104
6020 PRINT#1,"":FL=0:RETURN                   :rem 108
7000 ONXGOSUB2012,2022,2032,2042,2052,2062,2072,20
     82,2092,2002                             :rem 157
7010 IFFL<>1THENPRINT#1," ";:RETURN           :rem 105
7020 PRINT#1,"":FL=0:RETURN                   :rem 109
8000 ONXGOSUB2013,2023,2033,2043,2053,2063,2073,20
     83,2093,2003                             :rem 168
8010 IFFL<>1THENPRINT#1," ";:RETURN           :rem 106
8020 PRINT#1,"":FL=0:RETURN                   :rem 110
9000 ONXGOSUB2014,2024,2034,2044,2054,2064,2074,20
     84,2094,2004                             :rem 179
9010 IFFL<>1THENPRINT#1," ";:RETURN           :rem 107
9020 PRINT#1,"":FL=0:RETURN                   :rem 111
10000 ONXGOSUB2015,2025,2035,2045,2055,2065,2075,2
      085,2095,2005                           :rem 229
```

```
10010 IFFL<>1THENPRINT#1," ";:RETURN           :rem 147
10020 PRINT#1,"":FL=0:RETURN                    :rem 151
11000 ONXGOSUB2000,2111,2021,2031:PRINT#1," ";
                                                :rem 195
11010 ONX1GOSUB2000,2001,2011,2021,2031,2041,2051,
      2061,2071,2081,2091                       :rem 222
11020 IFFL<>1THENPRINT#1,"{2 SPACES}";:RETURN
                                                :rem 149
11030 FL=0:PRINT#1,"":RETURN                    :rem 153
12000 ONXGOSUB2000,2112,2022,2032:PRINT#1," ";
                                                :rem 199
12010 ONX1GOSUB2000,2002,2012,2022,2032,2042,2052,
      2062,2072,2082,2092                       :rem 233
12020 IFFL<>1THENPRINT#1,"{2 SPACES}";:RETURN
                                                :rem 150
12030 FL=0:PRINT#1,"":RETURN                    :rem 154
13000 ONXGOSUB2000,2113,2023,2033:PRINT#1," ";
                                                :rem 203
13010 ONX1GOSUB2000,2003,2013,2023,2033,2043,2053,
      2063,2073,2083,2093                       :rem 244
13020 IFFL<>1THENPRINT#1,"{2 SPACES}";:RETURN
                                                :rem 151
13030 FL=0:PRINT#1,"":RETURN                    :rem 155
14000 ONXGOSUB2000,2114,2024,2034:PRINT#1," ";
                                                :rem 207
14010 ONX1GOSUB2000,2004,2014,2024,2034,2044,2054,
      2064,2074,2084,2094                       :rem 255
14020 IFFL<>1THENPRINT#1,"{2 SPACES}";:RETURN
                                                :rem 152
14030 FL=0:PRINT#1,"":RETURN                    :rem 156
15000 ONXGOSUB2000,2115,2025,2035:PRINT#1," ";
                                                :rem 211
15010 ONX1GOSUB2000,2005,2015,2025,2035,2045,2055,
      2065,2075,2085,2095                        :rem 10
15020 IFFL<>1THENPRINT#1,"{2 SPACES}";:RETURN
                                                :rem 153
15030 FL=0:PRINT#1,"":RETURN                    :rem 157
```

Program 2. Appointment Calendar

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
80 DIMM$(12),W$(7):FORI=1TO12:READM$(I):NEXTI:FORI
   =1TO7:READW$(I):NEXTI                        :rem 118
90 SYS65517:A=PEEK(781):IFA=40THENPOKE53281,1
                                                :rem 167
100 PRINT"{CLR}{DOWN}{2 SPACES}THIS IS A PROGRAM":
    PRINT"{6 RIGHT}TO SHOW A"                    :rem 17
```

```
105 PRINT"{2 RIGHT}{PUR}MONTHLY CALENDAR{BLU}":PRI
    NT"{3 RIGHT}ON THE PRINTER{DOWN}"       :rem 214
110 PRINT"{RIGHT}PLEASE TYPE IN THE":PRINT"
    {3 RIGHT}{RED}MONTH{BLU} AND {RED}YEAR{BLU}"
                                             :rem 86
111 PRINT"THAT YOU WISH TO SEE":PRINT"{RIGHT}(EXAM
    PLE: {RED}12,1983{BLU}){PUR}{2 DOWN}" :rem 105
120 PRINTTAB(5);:INPUTMØ,Y                  :rem 132
130 PRINT"{2 DOWN}{2 SPACES}{BLU}THANK YOU! NOW--"
    :PRINT" PLEASE {PUR}TURN ON{BLU} THE" :rem 207
131 PRINT"PRINTER AND THEN TYPE":PRINTTAB(9)"{PUR}
    OK{DOWN}":INPUTR$                       :rem 193
151 IFR$<>"OK"THEN13Ø                       :rem 183
154 PRINT"{BLU}PRINTING{DOWN}":FORI=1TO8ØØ:NEXT:GO
    SUB1292:OPEN1,4                          :rem 23
2Ø2 PRINT#1,CHR$(14)"{3 SPACES}";M$(MØ);" ";Y:GOSU
    B16ØØ:GOSUB17ØØ:FORD=1TOE1:J1=J1+1      :rem 225
21Ø GOSUB1Ø5Ø:IFD<1ØTHENG$=" "              :rem 158
213 IFD>=1ØTHENG$=""                         :rem 96
214 IFD1=1THENPRINT#1,CHR$(15)"{3 SPACES}"W$(D1);C
    HR$(14)G$;"{RVS}"D"{OFF}";CHR$(15)"(" J1;")"
                                             :rem 71
215 IFD1=1THENGOSUB16ØØ                     :rem 128
217 IFD1=1THENGOTO22Ø                         :rem 8
219 PRINT#1,CHR$(15)"{3 SPACES}"W$(D1);CHR$(14)G$;
    D;CHR$(15)"(";J1;")":GOSUB16ØØ            :rem Ø
22Ø NEXTD                                    :rem 23
1ØØØ CLOSE1:END                             :rem 121
1Ø5Ø IFMØ=1THENMØ=13:Y=Y-1:GOTO1Ø8Ø          :rem 8Ø
1Ø6Ø IFMØ=2THENMØ=14:Y=Y-1                   :rem 23
1Ø8Ø M=MØ-2                                  :rem 47
11ØØ D1=INT(2.6*M-Ø.2)+D+Y-19ØØ+INT((Y-19ØØ)/4)
                                            :rem 2Ø7
115Ø D1=D1+INT(19/4)-2*19                    :rem 21
12ØØ D1=D1-INT(D1/7)*7+1                    :rem 235
121Ø IFMØ=4ORMØ=9THEND1=D1+1                :rem 135
123Ø IFMØ=13THENMØ=1:Y=Y+1:GOTO1245          :rem 81
124Ø IFMØ=14THENMØ=2:Y=Y+1:D1=D1+1          :rem 21Ø
1244 IFD1=8THEND1=1                          :rem 86
1245 IF(Y=21ØØANDMØ>=3)OR(Y>21ØØ)THEND1=D1-1:IFD1=
     ØTHEND1=7                              :rem 198
1247 IF(Y=22ØØANDMØ>=3)OR(Y>22ØØ)THEND1=D1-1:IFD1=
     ØTHEND1=7                              :rem 2Ø2
1249 IF(Y=23ØØANDMØ>=3)OR(Y>23ØØ)THEND1=D1-1:IFD1=
     ØTHEND1=7                              :rem 2Ø6
125Ø RETURN                                 :rem 168
1292 IFMØ=1ORMØ=3ORMØ=5ORMØ=7ORMØ=8ORMØ=1ØORMØ=12T
     HENE1=31                                :rem 75
1293 IFMØ=4ORMØ=6ORMØ=9ORMØ=11THENE1=3Ø     :rem 115
1294 IFMØ=2ANDY/4<>INT(Y/4)THENE1=28        :rem 152
```

```
1295 IFMØ=2ANDY/4=INT(Y/4)THENGOSUB14ØØ   :rem 132
1296 RETURN                                :rem 178
14ØØ IF(Y/1ØØ=INT(Y/1ØØ))AND(Y/4ØØ<>INT(Y/4ØØ))THE
     NE1=28:GOTO141Ø                       :rem 231
14Ø5 E1=29                                 :rem 232
141Ø RETURN                                :rem 166
16ØØ FORI=1TO2Ø:PRINT#1,CHR$(15)"  ";:NEXTI:rem 17Ø
16Ø5 FORK=1TO18:PRINT#1,".";" ";" ";:NEXTK:PRINT#1
     ,"."                                  :rem 231
161Ø RETURN                                :rem 168
17ØØ IFMØ=1THENJ1=Ø                         :rem 89
17Ø2 IFMØ=2THENJ1=31                       :rem 144
17Ø4 IFMØ=3THENJ1=59                       :rem 157
17Ø6 IFMØ=4THENJ1=9Ø                       :rem 155
17Ø7 IFMØ=5THENJ1=12Ø                      :rem 199
17Ø9 IFMØ=6THENJ1=151                      :rem 2Ø6
1711 IFMØ=7THENJ1=181                      :rem 2Ø3
1713 IFMØ=8THENJ1=212                      :rem 2Ø1
1715 IFMØ=9THENJ1=243                      :rem 2Ø8
1717 IFMØ=1ØTHENJ1=273                     :rem 253
1719 IFMØ=11THENJ1=3Ø4                     :rem 251
1721 IFMØ=12THENJ1=334                     :rem 248
1723 IFY/4<>INT(Y/4)THENGOTO173Ø           :rem 189
1725 IF(Y/1ØØ=INT(Y/1ØØ))AND(Y/4ØØ<>INT(Y/4ØØ))THE
     NGOTO173Ø                             :rem 159
1727 IF(Y/4=INT(Y/4))AND(MØ>=3)THENJ1=J1+1:rem 175
173Ø RETURN                                :rem 171
2ØØØ DATA "{2 SPACES}JANUARY"," FEBRUARY","
     {4 SPACES}MARCH","{4 SPACES}APRIL","
     {6 SPACES}MAY"                         :rem 36
2Ø1Ø DATA "{5 SPACES}JUNE","{5 SPACES}JULY","
     {3 SPACES}AUGUST","SEPTEMBER","{2 SPACES}OCTO
     BER"                                  :rem 229
2Ø2Ø DATA " NOVEMBER"," DECEMBER"           :rem 39
2Ø3Ø DATA "{4 SPACES}{RVS}SUNDAY{OFF}","{4 SPACES}
     MONDAY","{3 SPACES}TUESDAY"," WEDNESDAY","
     {2 SPACES}THURSDAY"                    :rem 9Ø
2Ø4Ø DATA "{4 SPACES}FRIDAY","{2 SPACES}SATURDAY"
                                           :rem 192
```

Program 3. Yearly Calendar

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
3 SYS65517:A=PEEK(781):IFA=4ØTHENPOKE53281,1
                                           :rem 113
5 OPEN1,4:DIMW4(3):GOSUB151Ø:I=1:J=2       :rem 128
7 PRINT#1,CHR$(14)SPC(13)"HAPPY NEW YEAR ";Y:PRINT
  #1                                        :rem 38
```

```
10 PRINT#1,CHR$(14)SPC(8)"JANUARY"SPC(13)"FEBRUARY
   "                                        :rem 49
12 GOSUB1ØØ9:GOSUB1ØØØ:GOSUB1Ø12:CØ=6:GOSUB1Ø19:GO
   SUB1ØØØ:GOSUB1Ø22                        :rem 69
15 MØ=I:M8=1:GOSUB292:GOSUB2Ø:GOTO35       :rem 228
2Ø D=1:GOSUB1Ø5Ø:W2=8-D1:W4(M8)=W2+1:GOSUB321
                                           :rem 123
22 IFD1=7THENGOTO3Ø                        :rem 167
25 FORD=2TOW2:GOSUB1Ø5Ø:GOSUB331:NEXTD     :rem 187
3Ø RETURN                                   :rem 67
35 GOSUB99Ø:MØ=J:M8=2:GOSUB292:GOSUB2Ø     :rem 1Ø5
44 W3=1                                     :rem 96
45 MØ=I:M8=1:GOSUB292:GOSUB2ØØ              :rem 6Ø
46 IFW4(2)=9THENPRINT#1,CHR$(15)SPC(1);     :rem 2Ø
5Ø GOSUB991:MØ=J:M8=2:GOSUB292:GOSUB2ØØ    :rem 151
56 IFW3=1ANDW4(1)>9THENPRINT#1,CHR$(15)SPC(Ø);
                                           :rem 223
57 IFW3=1ANDW4(1)<1ØTHENPRINT#1,CHR$(15)SPC(1);
                                             :rem 7
58 IFW3=4ANDW4(2)>3ØTHENPRINT#1,CHR$(15)SPC(Ø);
                                            :rem 15
65 W3=W3+1                                  :rem 24
7Ø IFW3<CØTHENGOTO45                         :rem Ø
71 PRINT#1," "                             :rem 185
72 IFI=1THENGOTO86                         :rem 133
73 IFI=3THENGOTO96                         :rem 137
74 IFI=5THENGOTO1Ø6                        :rem 18Ø
75 IFI=7THENGOTO116                        :rem 184
76 IFI=9THENGOTO126                        :rem 188
77 IFI=11THENGOTO199                       :rem 24Ø
86 PRINT#1,CHR$(14)SPC(9)"MARCH"SPC(16)"APRIL"
                                           :rem 171
88 I=3:J=4:GOTO12                          :rem 244
96 PRINT#1,CHR$(14)SPC(1Ø)"MAY"SPC(17)"JUNE"
                                            :rem 11
98 I=5:J=6:GOTO12                          :rem 249
1Ø6 PRINT#1,CHR$(14)SPC(9)"JULY"SPC(16)"AUGUST"
                                            :rem 14
1Ø8 I=7:J=8:GOTO12                          :rem 37
116 PRINT#1,CHR$(14)SPC(7)"SEPTEMBER"SPC(13)"OCTOB
    ER"                                    :rem 162
118 I=9:J=1Ø:GOTO12                         :rem 81
126 PRINT#1,CHR$(14)SPC(7)"NOVEMBER"SPC(13)"DECEMB
    ER"                                    :rem 131
128 I=11:J=12:GOTO12                       :rem 125
199 PRINT#1,CHR$(15)SPC(1):CLOSE1:END      :rem 194
2ØØ D4=W4(M8):D7=W4(M8)+6                   :rem 92
2Ø5 D=D4:GOSUB1Ø5Ø                          :rem 16
21Ø IFD1<>1THENPRINT"WHY D1=";D1           :rem 156
212 IFM8=1AND(D+1)<1ØTHENGOSUB528          :rem 198
```

```
213 IFM8=1AND(D+1)>9THENGOSUB530              :rem 154
214 IFM8=2AND(D+1)<10THENGOSUB428             :rem 200
215 IFM8=2ANDD4>=30ANDD4<=E1THENGOSUB433:GOTO217
                                              :rem 212
216 IFM8=2AND(D+1)>9THENGOSUB430              :rem 157
217 FORD=D4+1TOD7:GOSUB1050:GOSUB331:NEXTD:rem 130
220 W4(M8)=D7+1                                 :rem 9
225 RETURN                                    :rem 121
292 IFM0=1ORM0=3ORM0=5ORM0=7ORM0=8ORM0=10ORM0=12TH
    ENE1=31                                    :rem 26
293 IFM0=4ORM0=6ORM0=9ORM0=11THENE1=30         :rem 66
294 IFM0=2ANDY/4<>INT(Y/4)THENE1=28           :rem 103
295 IFM0=2ANDY/4=INT(Y/4)THENGOSUB1400         :rem 83
296 RETURN                                    :rem 129
321 IFD1=7THENPRINT#1,CHR$(15)SPC(36);D;:GOTO330
                                              :rem 101
322 IFD1=6THENPRINT#1,CHR$(15)SPC(31);D;:GOTO330
                                               :rem 96
323 IFD1=5THENPRINT#1,CHR$(15)SPC(26);D;:GOTO330
                                              :rem 100
324 IFD1=4THENPRINT#1,CHR$(15)SPC(21);D;:GOTO330
                                               :rem 95
325 IFD1=3THENPRINT#1,CHR$(15)SPC(16);D;:GOTO330
                                               :rem 99
326 IFD1=2THENPRINT#1,CHR$(15)SPC(11);D;:GOTO330
                                               :rem 94
327 IFD1=1THENPRINT#1,CHR$(15)SPC(6);D;:GOTO330
                                               :rem 50
328 PRINT#1,CHR$(15)SPC(3);D;:GOTO330         :rem 143
329 PRINT#1,CHR$(15)SPC(2);D;                 :rem 134
330 RETURN                                    :rem 118
331 IFD>E1THENPRINT#1,CHR$(15)SPC(5);:GOTO350
                                              :rem 196
332 IFD1=1ANDD<=9THENPRINT#1,D;:GOTO350       :rem 153
333 IFD1=1ANDD>9THENPRINT#1,D;:GOTO350         :rem 95
335 IFD<=9THENPRINT#1,CHR$(15)SPC(2);D;:GOTO350
                                               :rem 66
336 PRINT#1,CHR$(15)SPC(1);D;                 :rem 131
350 RETURN                                    :rem 120
428 IFD>E1THENPRINT#1,CHR$(15)SPC(9);:GOTO435
                                              :rem 211
429 GOTO328                                   :rem 117
430 IFD>E1THENPRINT#1,CHR$(15)SPC(9);:GOTO435
                                              :rem 204
431 GOTO329                                   :rem 111
433 PRINT#1,CHR$(15)SPC(1);D;                 :rem 129
435 RETURN                                    :rem 124
528 IFD>E1THENPRINT#1,CHR$(15)SPC(9);:GOTO535
                                              :rem 213
```

```
529 GOTO532                                       :rem 115
530 IFD>E1THENPRINT#1,CHR$(15)SPC(9);:GOTO535
                                                  :rem 206
531 GOTO533                                       :rem 109
532 PRINT#1,CHR$(15)SPC(5);D;:GOTO535             :rem 149
533 PRINT#1,CHR$(15)SPC(4);D;                     :rem 133
535 RETURN                                        :rem 125
990 PRINT#1,CHR$(15)SPC(3);:GOTO992                :rem 35
991 PRINT#1,CHR$(15)SPC(6);                        :rem 16
992 RETURN                                        :rem 132
1000 PRINT#1,CHR$(15)SPC(7);                       :rem 47
1001 RETURN                                       :rem 162
1009 PRINT#1,CHR$(15)SPC(3);                       :rem 52
1010 PRINT#1,"{4 SPACES}S{4 SPACES}M{4 SPACES}T
     {4 SPACES}W{4 SPACES}T{4 SPACES}F{4 SPACES}S"
     ;                                            :rem 134
1011 RETURN                                       :rem 163
1012 PRINT#1,"{4 SPACES}S{4 SPACES}M{4 SPACES}T
     {4 SPACES}W{4 SPACES}T{4 SPACES}F{4 SPACES}S"
                                                   :rem 77
1013 RETURN                                       :rem 165
1019 PRINT#1,CHR$(15)SPC(3);                       :rem 53
1020 PRINT#1,"{4 SPACES}{T}{4 SPACES}{T}{4 SPACES}
     {T}{4 SPACES}{T}{4 SPACES}{T}{4 SPACES}{T}
     {4 SPACES}{T}";                              :rem 196
1021 RETURN                                       :rem 164
1022 PRINT#1,"{4 SPACES}{T}{4 SPACES}{T}{4 SPACES}
     {T}{4 SPACES}{T}{4 SPACES}{T}{4 SPACES}{T}
     {4 SPACES}{T}"                               :rem 139
1023 RETURN                                       :rem 166
1050 IFM0=1THENM0=13:Y=Y-1:GOTO1080                :rem 80
1060 IFM0=2THENM0=14:Y=Y-1                         :rem 23
1080 M=M0-2                                        :rem 47
1100 D1=INT(2.6*M-0.2)+D+Y-1900+INT((Y-1900)/4)
                                                  :rem 207
1150 D1=D1+INT(19/4)-2*19                          :rem 21
1200 D1=D1-INT(D1/7)*7+1                          :rem 235
1210 IFM0=4ORM0=9THEND1=D1+1                      :rem 135
1230 IFM0=13THENM0=1:Y=Y+1:GOTO1250                :rem 77
1240 IFM0=14THENM0=2:Y=Y+1:D1=D1+1                :rem 210
1244 IFD1=8THEND1=1                                :rem 86
1245 IF(Y=2100ANDM0>3)OR(Y>2100)THEND1=D1-1:IFD1=0
     THEND1=7                                     :rem 137
1247 IF(Y=2200ANDM0>3)OR(Y>2200)THEND1=D1-1:IFD1=0
     THEND1=7                                     :rem 141
1249 IF(Y=2300ANDM0>3)OR(Y>2300)THEND1=D1-1:IFD1=0
     THEND1=7                                     :rem 145
1250 RETURN                                       :rem 168
1400 IF(Y/100=INT(Y/100))AND(Y/400<>INT(Y/400))THE
     NE1=28:GOTO1410                              :rem 231
```

```
1405 E1=29                                    :rem 232
1410 RETURN                                   :rem 166
1510 PRINT"{CLR}{DOWN}{2 RIGHT}THIS IS A PROGRAM":
     PRINT"{6 RIGHT}TO SHOW A"                :rem 129
1520 PRINT"{3 RIGHT}{PUR}YEARLY CALENDAR{BLU}":PRI
     NT"{3 RIGHT}ON THE PRINTER{DOWN}"        :rem 208
1530 PRINT"{RIGHT}PLEASE TYPE IN THE":PRINT"
     {3 RIGHT}YEAR THAT YOU":PRINT"{4 RIGHT}WISH T
     O SEE"                                    :rem 38
1535 PRINT"{3 RIGHT}(EXAMPLE:{PUR}1984{BLU})
     {2 DOWN}":PRINTTAB(6);:INPUTY            :rem 195
1570 PRINT"{DOWN}{3 RIGHT}THANK YOU! NOW--":PRINT"
     {RIGHT}PLEASE TURN ON THE"               :rem 145
1573 PRINT"PRINTER AND THEN TYPE"               :rem 9
1575 PRINTTAB(8)"{PUR}OK{BLU}{DOWN}"          :rem 105
1580 INPUTR$                                  :rem 212
1585 IFR$<>"OK"THEN1570                        :rem 44
1590 PRINT"PRINTING{DOWN}":FORI=1TO800:NEXT:RETURN
                                              :rem 194
```

Heat Seeker

Jeff Wolverton
Version by Tim Victor

Your jet climbs upward to avoid the missile, then dives for the ground. You can't shake the programmed missiles as they home in on your plane. You'll have to out-maneuver them or shoot them before they launch. But you'd better be fast. Joystick required.

The Heat Is On

Heat-seeking missiles are dangerous. They sense the heat from your jet engine and home in on you. They'll catch you, too—they're faster than a jet.

Your assignment: Eliminate the heat-seeker base. It's easy enough to strafe the missiles on the ground, but if any are launched, you'll have to take evasive action.

Piloting the Jet

Use the joystick to control the movement of the plane. The controls may seem a little confusing at first. You pull back to loop upward (counterclockwise) and push forward to loop down (clockwise), like a real airplane. The jet moves at a constant velocity—you can't speed up or slow down. Press the fire button to launch a missile at the heat seekers on the ground.

If you manage to eliminate all the heat seekers, you get to start all over again, with a new group of heat seekers. You have eight jets to work with—the number remaining is displayed on the screen, next to the score. To pause the game, press SHIFT LOCK.

The jets and missiles are *sprites* (rather than redefined characters), so the movement is smoother. And the program is written entirely in machine language, so it plays much faster.

You can fire at heat seekers on the ground. But it does no good to fire at a moving heat seeker. They're equipped with an Improved Electronic Evasion (IEE) circuit which makes them impossible to hit. The only way to get rid of a seeker is to make it crash into the ground.

When you're being pursued, dive for the ground and pull up at the last second. Seekers are faster, but they can't turn as quickly. Don't worry about dodging your shots since the plane is protected. If your jet is destroyed, all missiles reappear.

A two-player mode is available as well, but it's not competitive: Instead, the players take turns flying the plane, trying for the highest possible score. The game reads both joystick ports, so if you're using two joysticks, the inactive player should put down the joystick to avoid interfering.

There are three levels of difficulty: Novice, Intermediate, and Expert. The higher levels have faster action and tighter curves. A flight-time bonus of ten points is awarded every few seconds, just for staying in the air.

Special Instructions

"Heat Seeker" is written in machine language and loads into the area generally used by BASIC programs. You'll need "MLX," the machine language entry program (Appendix D), to enter it, but first you'll have to move the start of BASIC up. Follow these directions:

1. If you don't have a copy of MLX (Appendix D), type it in and save to tape or disk.
2. Turn the computer off and then on, and type **POKE642,32:SYS58260**. If you omit the POKE and SYS, you'll get an error in line 550 of MLX.
3. Load MLX and type RUN.
4. Answer these prompts:
   Starting Address: 2049
   Ending Address: 6470
5. When you've finished typing in Heat Seeker—and have saved a copy to tape or disk—turn off the computer, then turn it back on and go to 64 mode.
6. The enabling SYS is built into the program. After loading Heat Seeker, type RUN.

Heat Seeker

For mistake-proof program entry, be sure to use "MLX" (Appendix D).

```
2049 :011,008,001,000,158,050,229
2055 :048,054,049,000,000,000,158
2061 :076,027,008,000,000,000,124
2067 :000,000,000,000,000,000,019
2073 :000,000,169,014,141,033,126
2079 :208,169,002,141,032,208,023
2085 :160,024,169,000,153,255,030
```

```
2Ø91 :211,136,2Ø8,25Ø,169,ØØ2,251
2Ø97 :141,Ø23,212,169,Ø31,141,254
21Ø3 :Ø24,212,169,ØØ8,141,Ø22,119
21Ø9 :212,169,ØØ3,141,ØØ8,212,Ø38
2115 :169,Ø61,141,Ø12,212,169,Ø63
2121 :ØØØ,141,Ø15,212,141,Ø14,Ø84
2127 :212,169,Ø32,141,Ø19,212,Ø96
2133 :169,127,141,Ø2Ø,212,169,155
2139 :129,141,Ø18,212,169,ØØ1,249
2145 :141,ØØ3,212,169,Ø25,141,Ø2Ø
2151 :ØØ5,212,169,ØØØ,141,Ø25,143
2157 :ØØ8,Ø32,244,Ø2Ø,Ø32,1Ø8,Ø41
2163 :Ø19,169,Ø48,16Ø,ØØ6,153,158
2169 :2ØØ,ØØ7,136,2Ø8,25Ø,14Ø,Ø38
2175 :Ø21,ØØ8,172,248,Ø2Ø,Ø48,132
2181 :Ø18,16Ø,ØØ6,153,225,ØØ7,19Ø
2187 :136,2Ø8,25Ø,169,Ø5Ø,141,Ø69
2193 :198,ØØ7,169,Ø49,141,223,164
2199 :ØØ7,169,252,141,Ø17,ØØ8,233
22Ø5 :169,Ø11,162,ØØ4,157,Ø5Ø,198
2211 :Ø17,232,232,224,Ø16,2Ø8,Ø68
2217 :247,Ø32,141,Ø13,169,ØØ8,Ø11
2223 :141,Ø22,ØØ8,141,Ø23,ØØ8,ØØ6
2229 :Ø76,Ø75,Ø11,169,ØØØ,141,141
2235 :Ø66,Ø17,141,Ø67,Ø17,Ø32,Ø15
2241 :111,Ø13,173,Ø84,Ø17,2Ø1,Ø24
2247 :255,2Ø8,Ø34,Ø32,Ø74,Ø13,Ø47
2253 :173,212,Ø14,2Ø1,ØØ8,144,189
2259 :ØØ4,2Ø1,248,144,Ø2Ø,173,233
2265 :213,Ø14,2Ø1,ØØ8,144,ØØ4,Ø33
2271 :2Ø1,248,144,ØØ9,Ø32,Ø84,173
2277 :Ø13,Ø32,135,Ø13,Ø76,Ø34,Ø2Ø
2283 :Ø11,173,Ø66,Ø17,24Ø,ØØ3,233
2289 :Ø32,145,Ø1Ø,12Ø,169,253,2Ø2
2295 :141,ØØØ,22Ø,173,ØØ1,22Ø,234
23Ø1 :Ø41,128,24Ø,243,169,247,Ø41
23Ø7 :141,ØØØ,22Ø,Ø88,169,ØØ4,113
2313 :141,Ø18,ØØ8,162,ØØØ,189,Ø15
2319 :Ø68,Ø17,2Ø1,127,144,ØØ6,Ø66
2325 :173,Ø18,ØØ8,Ø32,122,Ø1Ø,128
2331 :Ø14,Ø18,ØØ8,232,224,ØØ6,Ø17
2337 :2Ø8,235,165,161,2Ø5,Ø2Ø,ØØ3
2343 :ØØ8,24Ø,ØØ6,141,Ø2Ø,ØØ8,2Ø6
2349 :Ø32,127,Ø12,173,Ø84,Ø17,234
2355 :Ø16,Ø33,2Ø1,192,24Ø,Ø29,25Ø
2361 :2Ø1,255,24Ø,Ø25,Ø32,ØØ2,Ø44
2367 :Ø12,144,ØØ7,169,192,141,216
2373 :Ø84,Ø17,2Ø8,Ø13,169,255,Ø47
2379 :141,Ø84,Ø17,169,128,141,243
2385 :212,Ø14,141,213,Ø14,16Ø,Ø67
```

```
2391 :009,169,255,217,074,017,060
2397 :240,013,136,208,248,173,087
2403 :084,017,201,192,208,003,036
2409 :076,018,011,173,031,208,110
2415 :141,016,008,041,001,240,046
2421 :009,032,071,010,032,084,099
2427 :013,076,075,011,173,016,231
2433 :008,041,002,240,003,032,199
2439 :033,010,173,016,008,041,160
2445 :252,208,003,076,192,008,112
2451 :141,016,008,169,004,141,114
2457 :019,008,170,173,019,008,038
2463 :045,016,008,240,003,032,247
2469 :179,009,014,019,008,232,114
2475 :232,224,016,208,236,076,139
2481 :192,008,045,016,208,240,118
2487 :002,056,036,024,189,000,234
2493 :208,106,056,233,008,176,208
2499 :002,169,000,201,160,144,103
2505 :002,169,144,074,074,074,226
2511 :074,168,185,074,017,201,158
2517 :255,208,072,169,192,153,238
2523 :074,017,169,000,157,034,158
2529 :017,157,035,017,169,226,078
2535 :157,001,208,189,000,208,226
2541 :056,233,016,041,224,024,063
2547 :105,028,157,000,208,032,005
2553 :154,012,138,074,170,169,198
2559 :064,157,066,017,169,255,215
2565 :157,248,007,138,010,170,223
2571 :032,063,013,152,010,010,035
2577 :168,169,096,153,113,007,211
2583 :153,114,007,153,153,007,098
2589 :153,154,007,096,173,084,184
2595 :017,201,255,208,030,032,010
2601 :181,012,032,063,013,169,255
2607 :000,141,036,017,141,037,163
2613 :017,141,084,017,169,226,195
2619 :141,003,208,169,002,141,211
2625 :040,208,032,084,013,096,026
2631 :169,000,141,034,017,141,061
2637 :035,017,169,226,141,001,154
2643 :208,169,002,141,039,208,082
2649 :169,001,013,028,208,141,137
2655 :028,208,032,135,013,032,031
2661 :063,013,160,192,132,162,055
2667 :173,031,208,041,002,240,034
2673 :003,032,033,010,164,162,005
2679 :208,242,096,013,017,008,191
2685 :141,017,008,173,018,008,234
```

```
2691 :073,255,045,021,208,141,106
2697 :021,208,169,254,157,250,172
2703 :007,096,173,067,017,201,192
2709 :028,176,006,169,000,141,157
2715 :066,017,096,173,017,008,020
2721 :208,001,096,169,000,141,008
2727 :066,017,141,067,017,169,132
2733 :004,170,168,045,017,008,073
2739 :208,010,152,010,168,232,191
2745 :232,224,016,208,242,096,179
2751 :141,018,008,013,021,208,088
2757 :141,021,208,173,018,008,254
2763 :073,255,168,045,017,008,001
2769 :141,017,008,173,016,208,004
2775 :041,001,240,012,173,018,188
2781 :008,013,016,208,141,016,111
2787 :208,076,238,010,152,045,188
2793 :016,208,141,016,208,173,227
2799 :034,017,157,034,017,173,159
2805 :035,017,157,035,017,173,167
2811 :000,208,157,000,208,173,229
2817 :001,208,157,001,208,138,202
2823 :074,168,169,000,153,066,125
2829 :017,088,076,090,013,174,215
2835 :021,008,254,022,008,160,236
2841 :010,032,181,012,136,208,092
2847 :250,240,041,169,000,141,104
2853 :034,017,141,036,017,141,167
2859 :035,017,141,037,017,032,066
2865 :063,013,173,028,208,009,031
2871 :001,141,028,208,169,002,092
2877 :141,039,208,141,040,208,070
2883 :169,192,133,162,165,162,026
2889 :208,252,120,169,100,141,039
2895 :000,208,169,100,141,001,186
2901 :208,169,000,141,016,208,059
2907 :169,001,141,021,208,169,032
2913 :240,141,248,007,169,015,149
2919 :141,039,208,169,254,045,191
2925 :028,208,141,028,208,169,123
2931 :000,133,160,133,161,133,067
2937 :162,141,020,008,032,111,083
2943 :019,162,009,169,255,157,130
2949 :074,017,202,016,250,169,093
2955 :000,141,084,017,173,031,073
2961 :208,173,030,208,044,248,032
2967 :020,048,028,160,009,185,089
2973 :197,007,170,185,222,007,177
2979 :153,197,007,138,153,222,009
2985 :007,136,208,239,169,001,161
```

```
2991 :056,237,021,008,141,021,147
2997 :008,174,021,008,189,022,091
3003 :008,208,019,160,000,044,114
3009 :248,020,048,001,200,185,127
3015 :022,008,208,202,136,016,023
3021 :248,076,188,012,222,022,205
3027 :008,189,022,008,024,105,055
3033 :049,141,214,007,173,001,034
3039 :220,045,000,220,041,016,253
3045 :208,246,173,001,220,045,098
3051 :000,220,041,016,240,246,230
3057 :169,000,141,035,017,141,232
3063 :036,017,169,085,141,034,217
3069 :017,088,076,184,008,160,018
3075 :009,185,074,017,201,255,232
3081 :240,013,136,016,246,169,061
3087 :253,045,021,208,141,021,192
3093 :208,056,096,169,192,153,127
3099 :074,017,152,010,010,168,202
3105 :169,096,153,113,007,153,212
3111 :114,007,153,153,007,153,114
3117 :154,007,152,010,010,010,132
3123 :072,144,010,169,002,013,205
3129 :016,208,141,016,208,208,086
3135 :008,169,253,045,016,208,250
3141 :141,016,208,104,024,105,155
3147 :028,141,002,208,169,226,081
3153 :141,003,208,169,247,141,222
3159 :249,007,169,171,141,037,093
3165 :017,173,031,208,044,017,071
3171 :208,048,251,173,018,208,237
3177 :201,242,208,244,169,007,152
3183 :141,040,208,169,002,013,172
3189 :021,208,141,021,208,173,121
3195 :031,208,024,096,072,138,180
3201 :072,162,005,254,200,007,061
3207 :169,058,221,200,007,208,230
3213 :008,169,048,157,200,007,218
3219 :202,208,238,104,170,104,149
3225 :096,072,138,072,173,205,141
3231 :007,024,105,005,201,058,047
3237 :176,005,141,205,007,208,139
3243 :234,233,010,141,205,007,233
3249 :162,004,208,207,072,138,200
3255 :072,162,004,208,200,120,181
3261 :169,049,141,020,003,169,228
3267 :234,141,021,003,088,169,083
3273 :048,141,214,007,032,132,007
3279 :255,160,016,185,046,013,114
3285 :201,064,144,003,056,233,146
```

```
3291 :064,153,011,004,169,003,111
3297 :153,011,216,136,208,235,160
3303 :200,140,026,008,173,026,036
3309 :008,208,012,169,045,141,052
3315 :025,004,169,062,141,026,158
3321 :004,208,010,169,060,141,073
3327 :025,004,169,045,141,026,153
3333 :004,173,000,220,045,001,192
3339 :220,074,074,074,176,005,122
3345 :160,001,140,026,008,074,170
3351 :176,005,160,000,140,026,018
3357 :008,074,176,202,032,129,138
3363 :255,173,026,008,240,003,228
3369 :076,027,008,133,198,000,227
3375 :080,076,065,089,032,065,198
3381 :071,065,073,078,063,032,179
3387 :089,032,032,078,169,128,075
3393 :141,011,212,169,129,141,100
3399 :011,212,096,169,255,056,102
3405 :237,003,208,141,015,212,125
3411 :096,169,000,141,015,212,204
3417 :096,169,024,141,025,008,040
3423 :165,162,141,024,008,169,252
3429 :064,141,004,212,169,065,244
3435 :141,004,212,096,165,162,119
3441 :205,024,008,240,016,141,235
3447 :024,008,173,025,008,201,046
3453 :048,176,007,141,001,212,198
3459 :238,025,008,096,169,000,155
3465 :141,001,212,096,076,153,048
3471 :013,011,000,000,000,000,167
3477 :000,000,000,000,120,169,182
3483 :171,141,020,003,169,013,160
3489 :141,021,003,169,014,141,138
3495 :005,220,088,096,173,000,237
3501 :220,045,001,220,141,146,178
3507 :013,173,034,017,013,035,208
3513 :017,208,003,076,195,014,186
3519 :173,146,013,041,016,208,020
3525 :003,238,066,017,173,146,072
3531 :013,041,003,201,003,208,160
3537 :003,076,195,014,173,144,046
3543 :013,141,145,013,173,034,222
3549 :017,141,149,013,048,003,080
3555 :169,000,044,169,255,141,237
3561 :150,013,160,008,136,014,202
3567 :145,013,144,250,192,000,215
3573 :240,037,014,149,013,046,232
3579 :150,013,014,145,013,144,218
3585 :023,024,173,149,013,109,236
```

```
3591 :034,017,141,149,013,144,249
3597 :003,238,150,013,173,034,112
3603 :017,016,003,206,150,013,168
3609 :136,208,219,173,146,013,152
3615 :041,002,208,022,056,173,021
3621 :148,013,237,149,013,141,226
3627 :148,013,173,035,017,237,154
3633 :150,013,141,035,017,076,225
3639 :076,014,024,173,148,013,247
3645 :109,149,013,141,148,013,122
3651 :173,035,017,109,150,013,052
3657 :141,035,017,173,144,013,084
3663 :141,145,013,173,035,017,091
3669 :141,151,013,048,003,169,098
3675 :000,044,169,255,141,152,084
3681 :013,160,008,136,014,145,061
3687 :013,144,250,192,000,240,174
3693 :037,014,151,013,046,152,010
3699 :013,014,145,013,144,023,211
3705 :024,173,151,013,109,035,114
3711 :017,141,151,013,144,003,084
3717 :238,152,013,173,035,017,249
3723 :016,003,206,152,013,136,153
3729 :208,219,173,146,013,041,177
3735 :002,208,022,024,173,147,215
3741 :013,109,151,013,141,147,219
3747 :013,173,034,017,109,152,149
3753 :013,141,034,017,076,195,133
3759 :014,056,173,147,013,237,047
3765 :151,013,141,147,013,173,051
3771 :034,017,237,152,013,141,013
3777 :034,017,076,198,014,076,096
3783 :216,014,008,000,000,000,181
3789 :000,000,000,000,000,000,205
3795 :000,000,000,000,000,173,128
3801 :000,208,056,237,002,208,160
3807 :144,002,024,036,056,106,079
3813 :141,212,014,173,001,208,210
3819 :056,237,003,208,144,002,117
3825 :024,036,056,106,141,213,049
3831 :014,169,003,045,016,208,190
3837 :240,046,201,003,240,042,001
3843 :201,001,240,019,173,212,081
3849 :014,056,233,128,201,165,038
3855 :176,003,024,105,172,141,124
3861 :212,014,076,069,015,173,068
3867 :212,014,024,105,128,201,199
3873 :085,144,003,056,233,172,214
3879 :141,212,014,076,069,015,054
3885 :173,212,014,048,009,201,190
```

```
3891 :085,048,012,056,233,172,145
3897 :144,007,201,165,016,003,081
3903 :024,105,172,141,212,014,219
3909 :173,084,017,048,003,076,214
3915 :012,017,169,000,044,037,098
3921 :017,048,009,044,036,017,252
3927 :016,014,169,001,208,010,249
3933 :044,036,017,016,003,169,122
3939 :002,044,169,003,141,210,156
3945 :014,169,000,044,213,014,047
3951 :048,009,044,212,014,016,198
3957 :014,169,001,208,010,044,051
3963 :212,014,016,003,169,002,027
3969 :044,169,003,141,211,014,199
3975 :173,210,014,056,237,211,012
3981 :014,074,144,008,041,001,167
3987 :141,203,014,076,034,016,119
3993 :240,018,169,000,056,237,105
3999 :212,014,141,212,014,169,153
4005 :000,056,237,213,014,141,058
4011 :213,014,169,000,141,214,154
4017 :014,141,215,014,173,212,178
4023 :014,013,213,014,208,003,136
4029 :076,012,017,173,214,014,183
4035 :024,109,212,014,141,214,141
4041 :014,173,215,014,024,109,238
4047 :213,014,141,215,014,173,209
4053 :214,014,056,237,036,017,019
4059 :077,036,017,016,014,173,040
4065 :215,014,056,237,037,017,033
4071 :077,037,017,048,212,016,126
4077 :023,173,215,014,056,077,027
4083 :037,017,048,011,172,210,226
4089 :014,204,211,014,208,003,135
4095 :076,012,017,169,000,044,061
4101 :169,001,044,212,014,016,205
4107 :002,073,001,044,213,014,102
4113 :016,002,073,001,172,210,235
4119 :014,204,211,014,240,005,199
4125 :073,001,141,203,014,173,122
4131 :201,014,141,202,014,173,012
4137 :036,017,141,206,014,048,247
4143 :003,169,000,044,169,255,175
4149 :141,207,014,160,008,136,207
4155 :014,202,014,144,250,192,107
4161 :000,240,037,014,206,014,064
4167 :046,207,014,014,202,014,056
4173 :144,023,024,173,206,014,149
4179 :109,036,017,141,206,014,094
4185 :144,003,238,207,014,173,100
```

```
4191 :036,017,016,003,206,207,068
4197 :014,136,208,219,173,203,030
4203 :014,208,022,056,173,205,017
4209 :014,237,206,014,141,205,162
4215 :014,173,037,017,237,207,036
4221 :014,141,037,017,076,151,049
4227 :016,024,173,205,014,109,160
4233 :206,014,141,205,014,173,122
4239 :037,017,109,207,014,141,156
4245 :037,017,173,201,014,141,220
4251 :202,014,173,037,017,141,227
4257 :208,014,048,003,169,000,091
4263 :044,169,255,141,209,014,231
4269 :160,008,136,014,202,014,195
4275 :144,250,192,000,240,037,018
4281 :014,208,014,046,209,014,178
4287 :014,202,014,144,023,024,100
4293 :173,208,014,109,037,017,243
4299 :141,208,014,144,003,238,183
4305 :209,014,173,037,017,016,163
4311 :003,206,209,014,136,208,223
4317 :219,173,203,014,208,022,036
4323 :024,173,204,014,109,208,191
4329 :014,141,204,014,173,036,047
4335 :017,109,209,014,141,036,253
4341 :017,076,012,017,056,173,084
4347 :204,014,237,208,014,141,045
4353 :204,014,173,036,017,237,170
4359 :209,014,141,036,017,076,244
4365 :015,017,076,108,017,000,246
4371 :000,000,000,000,000,000,019
4377 :000,000,000,000,000,000,025
4383 :000,000,000,000,000,000,031
4389 :000,000,000,000,000,000,037
4395 :000,000,000,000,000,000,043
4401 :000,000,000,000,000,000,049
4407 :000,000,000,000,000,000,055
4413 :000,000,000,000,000,000,061
4419 :000,000,000,000,000,000,067
4425 :000,000,000,000,000,000,073
4431 :000,000,000,000,000,000,079
4437 :001,254,002,253,004,251,082
4443 :008,247,016,239,032,223,088
4449 :064,191,128,127,000,000,095
4455 :000,000,000,000,000,162,009
4461 :000,189,050,017,141,101,095
4467 :017,189,034,017,141,102,103
4473 :017,048,003,169,000,044,146
4479 :169,255,141,103,017,189,233
4485 :035,017,141,104,017,048,239
```

```
4491 :003,169,000,044,169,255,011
4497 :141,105,017,160,008,136,200
4503 :014,101,017,144,250,192,101
4509 :000,240,066,014,102,017,084
4515 :046,103,017,014,104,017,208
4521 :046,105,017,014,101,017,213
4527 :144,046,024,173,102,017,169
4533 :125,034,017,141,102,017,105
4539 :144,003,238,103,017,189,113
4545 :034,017,016,003,206,103,060
4551 :017,024,173,104,017,125,147
4557 :035,017,141,104,017,144,151
4563 :003,238,105,017,189,035,030
4569 :017,016,003,206,105,017,069
4575 :136,208,190,169,000,141,043
4581 :106,017,189,085,017,045,176
4587 :016,208,240,003,238,106,022
4593 :017,024,189,018,017,109,103
4599 :102,017,157,018,017,189,235
4605 :000,208,109,103,017,157,079
4611 :000,208,144,003,238,106,190
4617 :017,044,103,017,016,003,209
4623 :206,106,017,044,106,017,255
4629 :016,014,169,001,141,106,212
4635 :017,024,189,000,208,105,058
4641 :088,157,000,208,169,001,144
4647 :205,106,017,208,019,189,015
4653 :000,208,201,088,144,012,186
4659 :206,106,017,056,189,000,113
4665 :208,233,088,157,000,208,183
4671 :173,106,017,208,012,189,000
4677 :086,017,045,016,208,141,070
4683 :016,208,076,089,018,189,159
4689 :085,017,013,016,208,141,049
4695 :016,208,024,189,019,017,048
4701 :109,104,017,157,019,017,004
4707 :189,001,208,109,105,017,216
4713 :201,029,176,002,169,029,199
4719 :201,250,144,002,169,250,103
4725 :157,001,208,232,232,224,147
4731 :016,240,003,076,110,017,073
4737 :044,067,017,048,003,238,034
4743 :067,017,162,005,254,068,196
4749 :017,202,016,250,173,084,115
4755 :017,048,003,238,084,017,042
4761 :173,036,017,208,013,173,005
4767 :037,017,208,008,169,255,085
4773 :141,249,007,076,001,019,146
4779 :169,000,141,107,017,173,010
4785 :036,017,048,012,201,032,011
```

```
4791 :144,017,169,004,141,107,253
4797 :017,076,202,018,201,224,159
4803 :176,005,169,006,141,107,031
4809 :017,173,037,017,048,016,253
4815 :201,032,144,025,024,169,034
4821 :008,109,107,017,141,107,190
4827 :017,076,236,018,201,224,223
4833 :176,009,024,169,009,109,209
4839 :107,017,141,107,017,173,025
4845 :107,017,201,004,208,002,008
4851 :169,010,201,006,208,002,071
4857 :169,011,024,105,238,141,169
4863 :249,007,173,034,017,208,175
4869 :013,173,035,017,208,008,203
4875 :169,255,141,248,007,076,139
4881 :105,019,169,000,141,107,046
4887 :017,173,034,017,048,012,068
4893 :201,035,144,017,169,004,087
4899 :141,107,017,076,050,019,189
4905 :201,224,176,005,169,006,054
4911 :141,107,017,173,035,017,025
4917 :048,016,201,032,144,025,007
4923 :024,169,008,109,107,017,237
4929 :141,107,017,076,084,019,253
4935 :201,224,176,009,024,169,106
4941 :009,109,107,017,141,107,055
4947 :017,173,107,017,201,004,090
4953 :208,002,169,010,201,006,173
4959 :208,002,169,011,024,105,102
4965 :230,141,248,007,076,049,084
4971 :234,076,114,019,076,219,077
4977 :020,032,181,255,120,173,126
4983 :022,208,009,016,141,022,025
4989 :208,169,029,141,024,208,136
4995 :169,007,141,035,208,169,092
5001 :000,141,037,208,169,007,187
5007 :141,038,208,169,015,141,087
5013 :039,208,169,147,133,254,075
5019 :169,022,133,255,169,128,007
5025 :133,252,169,059,133,253,136
5031 :160,000,177,254,208,024,222
5037 :230,254,208,002,230,255,072
5043 :177,254,170,169,000,145,070
5049 :252,230,252,208,002,230,079
5055 :253,202,208,243,240,008,065
5061 :145,252,230,252,208,002,006
5067 :230,253,230,254,208,002,100
5073 :230,255,165,252,201,064,096
5079 :208,208,169,254,141,028,199
5085 :208,165,001,041,251,133,252
```

```
5091 :001,160,000,185,000,220,025
5097 :153,000,048,185,000,221,072
5103 :153,000,049,200,208,241,066
5109 :165,001,009,004,133,001,046
5115 :160,000,185,154,020,153,155
5121 :000,050,200,192,032,208,171
5127 :245,160,000,152,153,000,205
5133 :051,200,192,008,208,248,152
5139 :169,004,133,255,169,216,197
5145 :133,253,169,000,133,254,199
5151 :133,252,168,169,096,145,226
5157 :254,169,008,145,252,200,041
5163 :208,245,230,255,230,253,184
5169 :165,255,201,007,208,235,096
5175 :169,096,153,000,007,169,137
5181 :008,153,000,219,200,192,065
5187 :192,208,241,032,219,020,211
5193 :160,000,169,254,153,250,035
5199 :007,169,002,153,041,208,147
5205 :200,192,006,208,241,169,077
5211 :255,141,248,007,141,249,108
5217 :007,160,000,185,186,020,143
5223 :201,064,144,003,056,233,036
5229 :064,153,192,007,169,003,185
5235 :153,192,219,200,044,248,147
5241 :020,016,006,192,024,208,075
5247 :228,240,004,192,033,208,008
5253 :222,169,003,153,192,219,067
5259 :200,192,040,208,248,088,091
5265 :096,096,066,064,096,096,147
5271 :067,065,096,000,000,000,123
5277 :000,000,003,003,003,003,169
5283 :003,011,011,043,043,040,058
5289 :040,000,000,000,000,000,209
5295 :192,192,192,192,192,224,079
5301 :224,232,232,040,040,080,005
5307 :076,065,089,069,082,049,105
5313 :058,032,032,032,032,032,155
5319 :032,032,032,083,072,073,011
5325 :080,083,058,032,032,032,010
5331 :080,076,065,089,069,082,160
5337 :050,058,160,000,162,004,139
5343 :189,145,020,153,112,007,081
5349 :189,149,020,153,152,007,131
5355 :200,202,208,240,192,040,037
5361 :208,234,096,076,249,020,100
5367 :000,000,169,000,141,247,036
5373 :020,160,110,162,172,169,022
5379 :021,032,073,022,173,247,059
5385 :020,010,170,189,231,004,121
```

```
5391 :073,128,157,231,004,032,128
5397 :086,022,240,030,189,231,051
5403 :004,073,128,157,231,004,112
5409 :152,024,109,247,020,201,018
5415 :255,208,002,169,000,201,106
5421 :003,208,002,169,001,141,057
5427 :247,020,208,208,173,247,130
5433 :020,024,105,004,141,050,145
5439 :017,105,002,141,052,017,141
5445 :105,003,141,201,014,105,126
5451 :001,141,144,013,169,002,033
5457 :205,247,020,208,003,238,234
5463 :144,013,169,255,141,248,033
5469 :020,160,025,162,026,169,143
5475 :022,032,073,022,173,248,157
5481 :020,010,170,232,232,189,190
5487 :044,005,073,128,157,044,050
5493 :005,032,086,022,240,035,025
5499 :024,109,248,020,189,044,245
5505 :005,073,128,157,044,005,029
5511 :152,024,109,248,020,201,121
5517 :254,208,002,169,255,201,206
5523 :001,208,002,169,000,141,156
5529 :248,020,076,103,021,160,013
5535 :021,162,051,169,022,032,104
5541 :073,022,032,086,022,208,096
5547 :251,096,197,032,201,032,212
5553 :206,032,058,084,082,069,196
5559 :080,088,069,032,044,069,053
5565 :084,065,073,068,069,077,113
5571 :082,069,084,078,073,032,101
5577 :044,069,067,073,086,079,107
5583 :078,032,017,013,045,084,220
5589 :067,069,076,069,083,032,097
5595 :079,084,032,078,079,084,143
5601 :084,085,066,032,068,078,126
5607 :065,032,075,067,073,084,115
5613 :083,089,079,074,032,069,151
5619 :083,085,032,017,013,053,014
5625 :056,057,049,032,033,069,033
5631 :084,085,080,077,079,195,087
5637 :032,044,210,197,203,197,120
5643 :197,211,032,212,193,197,029
5649 :200,032,032,032,032,032,121
5655 :155,017,147,014,050,032,182
5661 :049,032,058,083,082,069,146
5667 :089,065,076,080,032,070,191
5673 :079,032,082,069,066,077,190
5679 :085,078,032,017,013,078,094
5685 :073,071,069,066,032,079,187
```

```
5691 :084,032,069,082,073,070,213
5697 :032,083,083,069,082,080,238
5703 :017,013,134,254,133,255,109
5709 :177,254,032,210,255,136,117
5715 :208,248,096,173,000,220,004
5721 :045,001,220,041,028,201,113
5727 :028,208,244,169,000,133,109
5733 :162,169,028,197,162,208,003
5739 :252,173,000,220,045,001,030
5745 :220,041,004,208,003,160,237
5751 :255,096,173,000,220,045,140
5757 :001,220,041,008,208,003,094
5763 :160,001,096,173,000,220,013
5769 :045,001,220,041,016,208,156
5775 :220,160,000,096,000,008,115
5781 :192,000,001,027,192,000,049
5787 :001,063,192,000,001,063,219
5793 :128,000,001,062,000,002,098
5799 :060,000,002,060,000,002,035
5805 :060,000,002,062,000,002,043
5811 :063,000,002,063,000,002,053
5817 :063,000,002,062,000,002,058
5823 :062,000,002,060,000,002,061
5829 :060,000,002,056,000,002,061
5835 :048,000,009,012,000,002,018
5841 :028,000,002,060,000,002,045
5847 :060,000,002,124,000,002,147
5853 :124,000,002,252,000,002,089
5859 :252,000,002,252,000,002,223
5865 :124,000,002,060,000,002,165
5871 :060,000,002,060,000,002,107
5877 :124,000,001,001,252,000,111
5883 :001,003,252,000,001,003,255
5889 :216,000,001,003,000,027,248
5895 :224,000,002,248,000,002,227
5901 :126,003,224,063,255,252,168
5907 :127,255,255,063,255,255,205
5913 :000,055,255,255,252,255,073
5919 :255,254,063,255,252,007,093
5925 :192,126,000,002,031,000,132
5931 :002,007,000,022,003,000,077
5937 :002,007,000,002,015,000,075
5943 :002,015,000,002,063,000,137
5949 :002,063,128,000,001,015,014
5955 :224,000,001,003,248,000,031
5961 :002,255,128,000,001,063,010
5967 :224,000,001,015,224,000,031
5973 :001,003,248,000,002,248,075
5979 :000,002,060,000,002,012,167
5985 :000,032,003,252,000,001,129
```

```
5991 :015,252,000,001,031,240,130
5997 :000,001,031,192,224,127,172
6003 :000,001,249,252,000,001,106
6009 :127,240,000,001,127,192,040
6015 :000,001,063,000,002,060,253
6021 :000,002,016,000,035,008,194
6027 :000,002,060,000,002,252,199
6033 :000,001,003,254,000,001,148
6039 :015,254,000,001,063,159,131
6045 :000,001,254,007,003,248,158
6051 :000,001,015,248,000,001,172
6057 :063,240,000,001,063,192,216
6063 :000,032,048,000,002,060,061
6069 :000,002,031,000,002,031,247
6075 :192,000,001,007,240,000,115
6081 :001,007,252,000,001,001,199
6087 :255,000,002,031,192,000,167
6093 :001,007,240,000,001,001,199
6099 :252,000,002,252,000,002,207
6105 :240,000,002,240,000,002,189
6111 :224,000,002,192,000,019,148
6117 :040,040,000,001,040,040,134
6123 :000,001,041,104,000,001,126
6129 :041,104,000,001,009,096,236
6135 :000,001,009,096,000,001,098
6141 :001,064,000,001,001,064,128
6147 :000,001,001,064,000,001,070
6153 :001,064,000,001,001,064,140
6159 :000,032,001,064,000,001,113
6165 :001,064,000,001,001,064,152
6171 :000,001,001,064,000,001,094
6177 :001,064,000,001,009,096,204
6183 :000,001,009,096,000,001,146
6189 :041,104,000,001,041,104,080
6195 :000,001,040,040,000,001,133
6201 :040,040,000,035,170,000,086
6207 :002,170,128,000,001,021,129
6213 :085,000,001,021,085,000,005
6219 :001,021,085,000,001,170,097
6225 :128,000,001,170,000,046,170
6231 :170,000,001,002,170,000,174
6237 :001,085,084,000,001,085,093
6243 :084,000,001,085,084,000,097
6249 :001,002,170,000,002,170,194
6255 :000,044,008,000,002,010,175
6261 :000,002,006,128,000,001,254
6267 :021,128,000,001,165,064,246
6273 :000,001,041,080,000,001,252
6279 :010,084,000,002,021,000,252
6285 :002,005,000,039,005,000,192
```

```
6291 :002,021,000,001,010,084,009
6297 :000,001,041,080,000,001,020
6303 :165,064,000,001,021,128,026
6309 :000,001,006,128,000,001,045
6315 :010,000,002,008,000,040,231
6321 :032,000,002,160,000,001,116
6327 :002,144,000,001,002,084,160
6333 :000,001,001,090,000,001,026
6339 :005,104,000,001,021,160,230
6345 :000,001,084,000,002,080,112
6351 :000,039,080,000,002,084,156
6357 :000,002,021,160,000,001,141
6363 :005,104,000,001,001,090,164
6369 :000,001,002,084,000,001,057
6375 :002,144,000,002,160,000,027
6381 :002,032,000,026,008,128,177
6387 :000,001,010,168,000,001,167
6393 :043,224,000,001,011,224,240
6399 :000,001,011,232,000,001,244
6405 :042,160,000,001,002,032,242
6411 :000,051,136,000,001,002,201
6417 :170,000,001,002,174,000,108
6423 :001,002,238,128,010,255,145
6429 :160,010,255,224,011,254,175
6435 :168,011,255,224,042,255,222
6441 :168,043,255,224,043,255,005
6447 :232,011,255,224,047,255,047
6453 :160,042,255,224,047,255,012
6459 :248,043,187,224,010,170,173
6465 :168,255,013,013,013,013,028
```

Campaign Manager

Todd Heimarck

Campaign, advertise, poll regions, take stands on issues, and learn about the electoral process in this two-player national election simulation. The right strategy and a good candidate can lead you and your candidate to the White House.

Countdown to November

The Democratic delegates are gathered in Moscone Center, wearing straw hats, carrying balloons and signs. The floor fights are done. The time has come to nominate.

"Maryland?"

"Mister Chairman—the great state of Maryland, the Free State, Home of the World Champion Baltimore Orioles, casts all of its votes for the senator from Arizona."

The chairman pounds his gavel. The din of cheers and jeers subsides. The convention is deadlocked. And you control a large block of uncommitted delegates. It's all up to you.

The vice president from Rhode Island has good charisma and intelligence, but you know his health is poor. The reverend from Arkansas is attractive, but a bit conservative. Although the senator from Arizona is experienced, he's not very smart. Perhaps the New Jersey doctor? No, the Ohio senator has the best combination of personality and issues, plus you'll get a home region advantage in the populous Heartland.

Now it's the Republicans' turn. Of the five choices, the woman from South Carolina is the best all-around candidate. She has high charisma and fundraising appeal, which translates well into television ads.

It's time for the candidates and their campaign managers to hit the trail.

On the Road

The Democratic senator starts with $9 million and 59 health points. He rests two days (to build up his health), then spends two days fundraising. Campaign stops in Illinois and Texas sway the voters slightly to the Democratic side.

The Republican campaigns in her home state of South

Carolina. She then moves on to North Carolina, Virginia, and Florida, followed by a couple of days resting.

As the campaign progresses, the Democrat concentrates on personal appearances in the industrial Northeast, plus forays into the larger states of Texas, California, and Florida. The Republican candidate does less actual campaigning, preferring to spend more time on fundraising to pay for the (expensive) television ads.

In the crucial eighth week, both candidates rest and fundraise in preparation for the last-minute campaigning. The Democrat does a media blitz in the Pacific, Southern, and Atlantic states. The Republican hits the Heartland, Arklatex, and the Urban Northeast.

Initial returns from New England show the Republicans sweeping the region, but the large states of New York and Pennsylvania go Democratic. The Republicans win most states from Ohio to the Great Plains, but the Democrats pick up the Southern Atlantic states (except Florida). Texas votes for the GOP, while the rest of the region goes Democratic. The Rocky Mountain states are solid Republican. The Democrats win the Pacific States.

The final results show the Republicans winning six of nine regions and capturing the presidency, with 315 electoral votes to the Democrats' 223. Three of the four biggest states voted Democratic, but Ohio and Illinois (with 47 electoral votes between them) made the difference. The TV ads in the last week moved these two key states into the Republican camp.

Managing the Candidate

Written entirely in machine language, "Campaign Manager" pits you against an opponent. Each of you manages the campaign of your candidate. The player who makes the right decisions gets his or her candidate elected.

You have nine weeks to campaign. Each week you plan your moves and enter them via the menu on the itinerary. You have two defensive moves, resting and fundraising, and two ways to gain votes, campaigning (personal appearances) and advertising on television.

At the beginning of each turn you see a medium-resolution map of the U.S. which indicates which way each state is leaning. The MAP option allows you to move a cursor around the country, to identify which states are which. If the Republicans are ahead, the state is red. Democratic states are cyan (light blue). If you're using a black-and-white television, the Republican states are the darker ones. You may notice that states occasionally switch back and forth, even though neither candidate campaigned or advertised there. This indicates that the voters in that state are split down the middle, and because of slight errors in polling, seem to be leaning one way or the other.

Since you have only 63 days (nine weeks), you have enough time to campaign in each state once or twice. But in terms of electoral votes, California (with 47) is far more important than some of the smaller, three-vote states like North Dakota or Vermont. Generally, it makes sense to campaign more heavily in the ten biggest states, sometimes called "megastates."

State	**Electoral Votes**
CA	47
NY	36
TX	29
PA	25
IL	24
OH	23
FL	21
MI	20
NJ	16
NC	13

Winning the election requires 270 electoral votes (of a possible 538). The ten biggest states account for 254, just 16 short of a majority.

At the beginning of the campaign, each state has a large pool of undecided voters. As the game progresses, they make up their minds and the pool diminishes. It's possible, but unlikely, for all of a state's voters to decide before the end of the campaign. You would have to go to the state at least eight times before the undecided points were used up.

Each state has a built-in bias toward one party, based on past elections for president, senator, governor, and so on. The District of Columbia, for example, is staunchly Democratic, so the Democratic candidate will automatically get seven campaign points there, compared with a Republican's two.

Since the Republicans have won four of the last five elections (including landslide victories in 1972 and 1984), you might expect them to begin the game with a huge advantage. But if you look at nonpresidential elections, you will find a lot of states that elect Democratic governors, senators, and representatives and then vote for a Republican president. And a lot of those basically Democratic states were split by third-party campaigns (Wallace in 1968, Anderson in 1980).

To even things up, and make the game more playable, the Democrats begin with an electoral vote advantage of 282 to 256, although four of the megastates (Pennsylvania, Ohio, Florida, and North Carolina) are barely leaning to the Democratic side. The Republicans have the advantage of beginning with 29 of the 51 states (since DC has three electoral votes, it counts as a state). Most of the states west of the Mississippi are Republican, while the Democrats have most of the industrial Northeast and the South.

In addition to the natural political leanings, each state believes certain things about five general issues:

1. Unemployment/Inflation
2. Poverty/Crime
3. Agriculture
4. Education
5. Defense

(The issues are based on census reports, almanacs, and so forth.) A very urban state might be conservative on crime, for example, but not care much about agriculture. Each candidate has certain stands on these issues. When you campaign or advertise in a state, you can get up to three extra campaign points for each issue if you agree with the citizens there.

Finally, the candidate you choose has a campaign effectiveness rating based on charisma and intelligence. This factor translates to votes each time you campaign in a state.

Starting the Bandwagon

To start the game, choose which party will go first. You might want to flip a coin, the winner choosing either a party, or the first turn or second turn. In testing, we found that the second player has the very slight advantage of making the last move. Next, decide if one of you will start out as the campaign manager for the president running for a second term. Being incumbent gives you some extra campaigning strength and is not recommended if you want an even game.

Note that all choices can be made with a joystick in either port. Move the pointer to a menu item and press the fire button twice to make your choice. If you don't own a joystick, use I, J, K, and L for up, left, down, and right respectively. Press M in place of the fire button.

Players then pick which candidate will represent their party. Five randomly chosen candidates are available. To the right of the candidate's stats is the YES/NO counter. Before making your choice, pick NO for each possibility until you have seen all five. They will cycle around again so you can make your choice.

Although the heart of the game is the actual campaign, in some ways the convention is more important. Nominate a terrible candidate and you'll spend most of your campaign trying to catch up.

A candidate's personality greatly affects the outcome of the election. In the lower-left corner you'll see a list of five attributes, each associated with a number from one (worst) to eight (best). With a couple of exceptions, the ideal candidate is the one with straight eights.

First is charisma (CHAR), which is personal magnetism, panache, the ability to influence and excite people. This is the most important personality trait because it is part of both campaign effectiveness and advertising effectiveness.

Stamina (STAM) rates your candidate's health. A candidate with low stamina will have to rest frequently to regain health and strength.

Intelligence (INTL) adds points to campaign effectiveness and last-minute campaigning.

Experience (EXPR) helps you with fundraising. If your candidate has lots of experience, he or she has more contacts and connections for raising money. Since experience comes with age, it counts against your health, although stamina counts for more health points.

Appeal (APPL) also contributes to fundraising appeals. But if you have maximum appeal (eight) you may be tainted by your affiliations with special interest groups, and there is a backlash when you advertise. It's best to have an appeal of six or seven.

The candidates' attributes are generated by adding three random numbers, so candidates are more likely to have a middle number (four or five) than one of the extremes.

The personality traits translate into these five campaign factors:

Campaign Effectiveness (CHAR*2 + INTL): the key factor in campaign stops.

Strength/Health (STAM*4 + 9 − EXPR): determines the effectiveness of a rest day.

Fundraising Appeal (EXPR*3 + APPL): determines how much money can be raised in a day.

TV Ads (APPL OR 8 + CHAR): translates into votes when advertising.

Last-Minute Campaigning (INTL + STAM): wins last-minute votes to your side after the ninth week.

The significance of each factor is discussed later.

Taking a Stand

Next to the personality factors are the candidate's stands on various issues. You see five issues, each with a sliding scale of one (at the far left, representing liberal) to six (conservative). A Republican who wants to get tough on crime, for example, will have a rank of six. A Democrat who wants to solve the unemployment problem will have a rating of one.

Candidates will range from two to five on the issues of agriculture and education. On the other three issues, the Democrats will have stands from one to four; the Republicans will go from three to six.

You will generally get more votes with middle-of-the-road beliefs. Look for a candidate with twos or threes if you're the Democrat. Fours and fives are best for the Republican. The exceptions are agriculture and education, where you do best with a three or a four.

Common sense tells you which issues are important in most states. Agriculture is a major issue in the farming states. Your stand on defense makes a difference in states with a lot of military-related industry.

The candidate's personality is generally more crucial than the stands on issues. If you have a lot of charisma, intelligence, and appeal, it doesn't matter that you may have radical views on one or two issues.

If you have five very bad candidates, press RUN/STOP–RESTORE and try again. It's not much fun to run a campaign you are destined to lose.

Strategies

After the nominees have been chosen, the first week begins. You may notice that some states have changed colors. That's because each nominee gets the equivalent of campaigning once in each state. Some people make up their minds before the campaign even starts. If one candidate is much more charismatic or happens to hit the right issues, a state may jump over to his or her side. In addition, each gets a home state and home region advantage.

You should develop a strategy. If your appeal and charisma are strong, concentrate on television ads. If your candidate has a strong anticrime stance, visit the more urban states. At the very least, you should plan to visit each of the megastates.

You begin in your home state where it is traditional to campaign once (but not twice). And the first week usually means some fundraising and resting as purely defensive moves.

Under the week's itinerary are two numbers representing money and health. At the beginning of each week, your treasurer tells you how much money you have, up to a maximum of $25 million. Your personal physician figures out how healthy you are. At most you'll have 255 health points.

If you fall below $4 million any time during the week, television advertising will be useless until you replenish the campaign coffers. If you have less than one million, you won't be able to pay the pollster (the bar graph to the left of the map will disappear). When your bank account falls to zero, the campaign is paralyzed until you sponsor a fundraiser. You can't even afford to pay your doctor or staff.

It takes time away from campaigning, but you have to raise money once in a while. Each fundraising point (experience times three plus appeal) is worth $200,000.

Campaigning takes a lot out of you, so you have to occasionally take a day to rest and relax. When you decide to catch some Z's, the itinerary will be filled with (you guessed it) Z's. Each day of rest adds double your strength factor, plus campaign effectiveness, plus the number of states you are winning to the health you have. A high campaign effectiveness gives you optimism; you rest better. If you're behind, you lose sleep worrying about it. Resting two days in a row gets you 16 extra health points.

There are two reasons to keep your health up. First, when you campaign in a state, you get an extra campaign point for every 32 health points you possess. Second, if your health falls below eight you look haggard and stutter; campaigning does you no good.

Polls

The treasurer counts dollars, the doctor counts your health, and your pollster counts votes.

The pollster does three things. First, you get a bar chart that shows how many electoral votes would go to the Democrats and Republicans if the election were held at that time. You can see it to the left of the map. The gray bar marked U represents undecided states too close to call. Second, you have a map of the U.S. to show you, at a glance, which way each state is leaning. Republican states are red; Democratic states are blue. These first two services are part of the pollster's contract and cost you nothing. Of course, if your money drops lower than $1 million, you have to stop paying the pollster; all you get is the map.

The third service is the most important—regional polls. To get a poll of all states in a region, move to POLL on the main menu and press the fire button twice. You'll see a bar chart showing which way each state in the region is leaning, from one (half a character wide) to four (two characters). The poll reflects the political situation at the beginning of the week; whatever campaigning you have planned for the week is not included. A state with a thin bar can usually be taken with a single campaign stop.

Don't use polls in the first couple of weeks, because most states start out fairly even and you won't learn much. But polling can be a powerful tool toward the end of the game. If New York is firmly committed to you, forget about further efforts in that state. And if you find a whole region weakly supporting your opponent, you can hit them with TV ads and score a few dozen electoral votes.

Regional polls cost $100,000 and are not available if you begin the week with less than $1 million.

More Campaigning Options

The final character (although transparent) in your entourage is the jet pilot. Your jet can carry you on short hops within a region for almost nothing. But if you travel to a new region, you shell out $100,000 for fuel, maintenance, and so on. As long as you're in a region, you might as well stay there a few days to avoid a lot of travel expenses. Again, you don't actually move to a new region until you have campaigned in one of the states. You can use the travel option to conduct regional polls; you'll pay $100,000 for the poll, and another $100,000 if you decide to campaign in a region. If you travel to a region to poll and decide not to campaign, you won't be charged for traveling.

Benjamin Franklin once said that after three days, guests and fish begin to smell. The same principle applies to campaigning.

Campaign once and you gain some votes. Stay for a second day, and the voters of a state are flattered; you gain a couple of bonus votes. But stick around for a third or fourth day and you have overstayed your welcome. Do not campaign in a state more than two days in a row.

Each state begins with 255 undecided voter points. Your main goal is to use campaigning and television advertising to sway the undecided. And you have to maintain your health and money.

The effects of a personal appearance can vary. You get up to three points for each issue (if the state agrees with you), one point for every 32 health points, and up to 24 for your campaign effectiveness (intelligence plus double charisma), and a two-point bonus if it's your second day in the state.

If your money is down to zero, you get no campaign points. If your health is below eight, you get a single vote.

Each campaign stop decreases your health and money. It's possible to run out in the middle of the week, making each succeeding visit ineffective until you rest or raise money. Let's say you go to Connecticut and impress 23 of the 255 undecideds. The pool of available voters is reduced by that number. Half of 23 (11 points) is charged against your health. Half again (5 points) times $100,000 is subtracted from your money. In addition, each state has some people who don't agree with you, so a quarter of your total (five points) goes to your opponent as a reaction against your speech. If you had previously been in a different region, travel expenses of $100,000 are subtracted.

Television advertising is a little different. It affects every state in the region and quickly swings voters to your side. To advertise, first travel to the region and make at least one campaign stop to establish your presence.You can then place the cursor on TV ADS and press the fire button twice. After campaigning once, advertise as much as you like.

Unlike resting and campaigning, the effects of advertising do not accumulate from day to day. If you advertise two days in a row, you don't get bonus points. Advertising does grow in strength from week to week, however, and will be more effective toward the end of the campaign.

If you flood the region with ads, it's possible to bring a whole section of the country to your side. But it is costly. In each state, advertising credits you with half your campaign effectiveness, half your TV ad effectiveness rating, points for issues, plus two times the week number (in the seventh week, for example, you get 14 extra campaign points).

The cost is the usual one-fourth of campaign points gained, plus double the TV ad effectiveness. The large regions can cost a lot. Going on TV in the Atlantic States (all nine) or in the Rocky Mountains (eight states) can easily deplete your treasury.

On the day you plan to advertise, you must have at least $4 million. If you don't, you waste the day and gather no new votes. So, if you begin the week with $5 million and campaign in six states, it's likely you'll have less than $4 million by Saturday. Your ad campaign will do you no good.

There is one more item you can choose: RECONSIDER. If you make a mistake, this option wipes your itinerary clean so that you can start the week anew. Your choices are not permanent until you fill out the seventh day and press the fire button. (If you pull down on the joystick, your slate will be wiped clean—a quicker way to reconsider.)

The ninth week is usually the most hectic. If you sponsored some fundraisers in the eighth week, you will want to spend a lot on TV advertising in the regions where you have a chance. Polls can tell you which states are most vulnerable.

After both candidates have finished their last week of campaigning, a couple of things happen. The last region to be visited by a candidate gives a few extra votes to him or her. And the last-week routine goes into action, as all the undecided voters make up their minds. Both candidates get their last-minute campaigning points (intelligence plus stamina) added to each state in the country. The undecided voters are split between the candidates, and ties are resolved (based on the built-in bias to one party or the other).

Main Menu Command Summary

CAMPAIGN—allows you to make a personal appearance in one of the states of the region you're visiting. Results depend on campaign effectiveness, built-in party bias of the state, health, and issues. Does not work if you have zero health or money, or if all undecided voters have been claimed. Gains votes; costs health and money.

TV ADS—blankets the region with advertising. Reduces health and costs a lot of money, but can quickly deliver a big chunk of votes. Net votes based on TV advertising effectiveness, campaign effectiveness, and issues. Does not work if you have less than $4 million.

FUNDRAIS—raises money for your campaign based on fundraising ability. Takes a day, gains no votes, costs nothing.

REST—builds up your health points, according to strength factor. Extra points if you rest two days in a row. Gains no new votes, costs nothing.

MAP—moves the cursor around the map, prints the state name, electoral votes, and region number. For information only; costs nothing.

POLL—provides a bar graph showing which way the states in the region are leaning. Costs $100,000 (immediately). Not available if money falls below $1 million.

RECONSIDER—erases the week's itinerary if you make a mistake.

TRAVEL—takes you to a new region of the country. Costs $100,000 (not charged to you until you actually campaign there).

Election Night Coverage

The map is drawn for the final time. The final bar chart appears to the left, which should indicate at a glance which candidate won. (If you want a suspenseful end to the game, hide the bar chart.) Beginning with region 1 (New England), the electoral votes are displayed, with region totals below.

The winner is the candidate with the most electoral votes. There is a slight chance that there will be a tie, in which case you'd have to flip a coin. If you want to play again, press RUN/STOP–RESTORE and type RUN.

Finally, here are a few rules of etiquette which help to make a fairer game. First, if you're playing with two joysticks, try to avoid interfering with your opponent's choices. This is like rudely interrupting during a debate. Remember, the joystick routine reads *both* joysticks.

Second, when you have filled out your itinerary and the prompt PRESS FIRE BUTTON TO CONTINUE appears, let your opponent study what moves you made, and he or she can then press the fire button. It is a courtesy to let your opponent know where you will be so that you don't accidentally meet on the campaign circuit.

Third, since polls cost money, they should be kept private. When the other player is taking a poll, avoid looking at the screen. Let's hope we all learned from Watergate.

Special Instructions for Entering Campaign Manager

Since the program is written entirely in machine language, you must use the "MLX" machine language editor (Appendix D) to enter it. Before loading MLX, you have to protect part of BASIC memory by typing the following line:

POKE 642,50: SYS 58260

You'll then see the usual startup message, but you'll notice less than the normal 39K RAM. Next, load MLX, using a starting address of 2049 and ending address of 9518, and begin typing. The program uses about 10K, which was crunched down to about 7K to make typing it in a little easier. Since it's such a long program, you may want to enter it in parts. If you choose to do so, make sure you follow the MLX instructions for loading and saving, **and *always* enter the above POKE**

and SYS before loading MLX. The newest version of MLX has a numeric keypad, which should save you some time.

When you have finished typing Campaign Manager, make sure to save it to tape or disk (maybe a couple of backup copies as well). Turn your computer off and then on, go to 64 mode, load the program, and type RUN.

Campaign Manager

See special instructions in article before entering this program. For mistake-proof program entry, be sure to use "MLX" (Appendix D).

```
2049 :011,008,010,000,158,050,238
2055 :048,054,049,000 000,000,158
2061 :032,110,012,032,241,012,196
2067 :032,122,017,032,108,031,105
2073 :069,250,204,204,204,204,136
2079 :220,192,000,000,000,005,192
2085 :229,255,167,255,255.255,173
2091 :255,178,030,128,000,000,122
2097 :219,095,250,031,255,255,130
2103 :255,255,143,045,000,004,245
2109 :245,037,255,255,031,255,115
2115 :255,255,241,197,250,076,061
2121 :255,248,095,095,255,255,252
2127 :255,255,143,191,175,245,063
2133 :255,115,037,245,255,255,223
2139 :255,252,204,254,250,247,017
2145 :035,076,032,015,247,255,245
2151 :255,255,255,250,254,162,254
2157 :250,047,018,000,095,021,028
2163 :255,255,227,255,092,252,171
2169 :204,060,204,000,000,127,204
2175 :175,255,255,255,250,255,036
2181 :204,060,207,176,000,001,013
2187 :242,255,255,191,255,239,040
2193 :175,250,247,224,000,000,017
2199 :000,001,051,127,255,255,072
2205 :242,255,255,240,000,000,125
2211 :079,160,128,000,119,255,136
2217 :047,225,035,127,000,000,091
2223 :013,255,000,096,000,007,034
2229 :176,000,000,000,126,000,227
2235 :000,211,058,000,112,000,056
2241 :002,000,000,000,001,250,190
2247 :000,016,000,160,000,000,119
2253 :000,000,000,000,000,001,206
2259 :032,000,000,000,000,000,243
2265 :032,227,008,032,041,009,054
2271 :032,078,009,096,173,014,113
```

```
2277 :220,041,254,141,014,220,095
2283 :165,001,041,251,133,001,059
2289 :169,209,133,252,169,057,206
2295 :133,254,160,000,132,251,153
2301 :132,253,177,251,145,253,184
2307 :136,208,249,198,252,198,220
2313 :254,169,055,197,254,208,122
2319 :239,165,001,009,004,133,054
2325 :001,173,014,220,009,001,183
2331 :141,014,220,173,024,208,039
2337 :041,240,009,014,141,024,246
2343 :208,096,169,057,133,252,186
2349 :133,254,169,080,133,251,041
2355 :169,208,133,253,032,068,146
2361 :009,169,024,133,251,169,044
2367 :216,133,253,198,254,160,253
2373 :039,177,251,145,253,136,046
2379 :016,249,096,169,255,141,233
2385 :003,056,169,240,141,002,180
2391 :056,169,015,141,001,056,013
2397 :162,000,142,000,056,134,075
2403 :251,138,032,117,009,138,016
2409 :032,114,009,232,224,016,220
2415 :208,243,096,234,074,074,016
2421 :041,003,168,185,000,056,058
2427 :160,003,145,251,136,016,066
2433 :251,230,251,230,251,230,036
2439 :251,230,251,096,169,054,162
2445 :133,252,169,000,133,251,055
2451 :168,170,224,188,208,001,082
2457 :096,189,025,008,072,074,105
2463 :056,106,074,074,145,251,097
2469 :032,181,009,104,041,015,035
2475 :009,032,145,251,032,181,053
2481 :009,232,208,224,201,032,059
2487 :208,004,009,192,145,251,224
2493 :200,192,025,240,001,096,175
2499 :169,000,145,251,168,024,184
2505 :169,026,101,251,133,251,108
2511 :144,002,230,252,096,012,175
2517 :001,003,002,014,160,004,141
2523 :185,212,009,153,032,208,250
2529 :136,016,247,173,017,208,254
2535 :009,064,141,017,208,096,254
2541 :032,247,009,032,110,010,165
2547 :032,185,010,096,169,147,114
2553 :032,210,255,160,003,032,173
2559 :087,010,169,144,032,210,139
2565 :255,169,171,032,210,255,073
2571 :169,163,032,101,010,169,143
```

```
2577 :167,032,210,255,162,015,090
2583 :160,003,032,082,010,169,223
2589 :170,032,210,255,169,154,251
2595 :032,210,255,169,160,032,125
2601 :101,010,169,144,032,210,195
2607 :255,169,165,032,210,255,109
2613 :202,208,223,160,003,032,113
2619 :082,010,169,174,032,210,224
2625 :255,169,172,032,101,010,036
2631 :169,173,032,210,255,169,055
2637 :146,032,210,255,096,169,217
2643 :013,032,210,255,169,032,026
2649 :032,210,255,136,208,250,156
2655 :169,018,032,210,255,096,107
2661 :160,025,032,210,255,136,151
2667 :208,250,096,169,004,133,199
2673 :254,169,044,133,253,169,111
2679 :054,133,252,169,000,133,092
2685 :251,169,000,168,162,015,122
2691 :177,251,208,007,032,160,198
2697 :010,202,208,246,096,145,020
2703 :253,200,208,240,041,063,124
2709 :170,189,192,055,041,192,220
2715 :017,247,145,247,096,024,163
2721 :169,026,101,251,133,251,068
2727 :144,002,230,252,169,040,236
2733 :024,101,253,133,253,144,057
2739 :002,230,254,160,000,096,153
2745 :169,015,133,249,169,216,112
2751 :133,254,169,044,133,253,153
2757 :133,247,169,004,133,248,107
2763 :169,034,133,252,169,173,109
2769 :133,251,160,024,177,251,181
2775 :201,000,240,043,133,002,066
2781 :041,063,170,189,192,055,163
2787 :041,015,145,253,169,192,018
2793 :036,002,240,025,048,008,080
2799 :189,120,034,032,147,010,003
2805 :208,015,080,007,169,192,148
2811 :032,155,010,208,006,189,083
2817 :121,034,032,147,010,234,067
2823 :136,016,203,169,025,024,068
2829 :101,251,133,251,144,002,127
2835 :230,252,198,249,208,001,133
2841 :096,169,040,024,101,247,190
2847 :133,247,144,002,230,248,011
2853 :169,040,024,101,253,133,245
2859 :253,144,165,230,254,208,017
2865 :161,173,018,208,072,101,014
```

```
2871 :162,074,074,074,168,104,199
2877 :229,162,074,141,032,208,139
2883 :140,036,208,096,031,067,133
2889 :065,077,080,065,073,071,248
2895 :078,032,077,065,078,065,218
2901 :071,069,082,013,000,162,226
2907 :018,160,008,024,032,240,061
2913 :255,162,000,189,071,011,017
2919 :240,006,032,210,255,232,054
2925 :208,245,160,005,169,001,129
2931 :141,134,002,169,018,032,099
2937 :210,255,162,040,173,134,071
2943 :002,073,003,141,134,002,226
2949 :169,163,032,210,255,202,140
2955 :208,250,136,208,235,169,065
2961 :146,076,210,255,169,146,123
2967 :133,254,169,000,133,253,069
2973 :162,000,232,236,137,036,192
2979 :240,047,189,137,036,133,177
2985 :249,041,007,133,247,165,243
2991 :249,074,074,074,074,041,249
2997 :007,133,248,160,002,032,251
3003 :230,011,165,247,160,001,233
3009 :032,230,011,169,255,160,026
3015 :005,145,253,169,005,024,032
3021 :101,253,133,253,076,159,156
3027 :011,169,000,170,168,185,146
3033 :068,034,157,000,120,232,060
3039 :232,200,192,052,208,243,070
3045 :096,145,253,200,200,145,244
3051 :253,096,169,145,133,248,255
3057 :169,000,133,247,230,247,243
3063 :133,254,170,162,000,189,131
3069 :189,036,133,249,074,074,240
3075 :074,074,133,250,189,240,195
3081 :036,133,251,074,074,133,198
3087 :252,074,074,133,253,160,193
3093 :004,162,004,181,249,072,181
3099 :041,003,024,105,001,145,090
3105 :247,104,074,074,041,003,064
3111 :024,105,003,010,010,010,201
3117 :010,017,247,145,247,136,079
3123 :202,016,226,230,247,160,108
3129 :002,169,015,049,247,170,197
3135 :232,138,010,010,010,010,217
3141 :133,002,138,005,002,145,238
3147 :247,136,208,235,230,247,098
3153 :230,247,230,247,230,247,232
3159 :230,254,166,254,224,051,242
3165 :208,157,096,169,255,141,095
```

```
3171 :Ø15,212,169,128,141,Ø18,Ø14
3177 :212,141,Ø24,212,Ø96,162,184
3183 :Ø64,169,ØØØ,157,ØØØ,143,132
3189 :157,Ø64,143,2Ø2,2Ø8,247,114
3195 :169,128,141,138,ØØ2,169,1Ø2
32Ø1 :ØØ8,Ø32,21Ø,255,Ø32,149,Ø47
32Ø7 :Ø11,Ø32,25Ø,Ø26,Ø32,1Ø8,Ø82
3213 :Ø27,Ø32,128,Ø23,Ø32,139,Ø1Ø
3219 :ØØ9,Ø32,Ø3Ø,Ø28,Ø32,217,239
3225 :ØØ8,Ø32,237,Ø11,Ø32,217,178
3231 :ØØ9,Ø32,237,ØØ9,169,158,ØØ5
3237 :Ø32,21Ø,255,Ø32,Ø9Ø,Ø11,Ø27
3243 :Ø32,Ø3Ø,Ø2Ø,Ø32,Ø5Ø,Ø11,Ø9Ø
3249 :Ø32,Ø26,Ø31,173,1Ø7,Ø31,Ø65
3255 :24Ø,245,Ø32,217,ØØ9,Ø32,19Ø
3261 :Ø96,Ø12,162,ØØ4,16Ø,ØØ5,116
3267 :Ø32,163,Ø28,141,Ø21,143,211
3273 :141,Ø35,Ø37,162,ØØ7,16Ø,231
3279 :ØØ9,Ø32,163,Ø28,162,ØØØ,Ø89
3285 :16Ø,ØØØ,2Ø1,ØØØ,24Ø,ØØ7,Ø53
3291 :Ø41,ØØ1,24Ø,ØØ2,2Ø2,2ØØ,137
3297 :136,142,Ø15,143,14Ø,Ø79,112
33Ø3 :143,Ø32,Ø46,Ø17,2Ø8,ØØ3,168
33Ø9 :Ø76,157,Ø12,Ø96,169,ØØØ,235
3315 :141,Ø36,Ø37,169,128,133,119
3321 :247,169,143,133,248,169,Ø78
3327 :ØØ5,133,ØØ2,16Ø,ØØ5,162,21Ø
3333 :ØØ3,173,Ø27,212,Ø41,ØØ3,2Ø8
3339 :149,249,2Ø2,2Ø8,246,169,21Ø
3345 :ØØ1,Ø37,25Ø,Ø24,1Ø5,ØØ1,179
3351 :1Ø1,251,1Ø1,252,145,247,Ø96
3357 :136,2Ø8,228,16Ø,ØØ6,173,172
3363 :Ø27,212,Ø41,ØØ3,17Ø,192,168
3369 :ØØ8,24Ø,Ø1Ø,192,ØØ9,24Ø,228
3375 :ØØ6,173,Ø21,143,24Ø,ØØ2,12Ø
3381 :232,232,232,138,145,247,255
3387 :2ØØ,192,Ø11,2Ø8,226,173,Ø45
3393 :Ø27,212,Ø41,Ø63,24Ø,249,129
3399 :2Ø1,Ø52,176,245,145,247,113
34Ø5 :2ØØ,173,Ø15,143,145,247,232
3411 :2Ø8,ØØ9,173,Ø27,212,Ø41,241
3417 :ØØ7,Ø1Ø,Ø1Ø,145,247,Ø32,Ø28
3423 :22Ø,Ø14,198,ØØ2,2Ø8,157,126
3429 :16Ø,ØØØ,14Ø,Ø45,Ø17,169,12Ø
3435 :128,133,247,169,143,133,Ø36
3441 :248,173,Ø45,Ø17,2Ø1,ØØ5,Ø34
3447 :176,236,17Ø,24Ø,ØØ6,Ø32,211
3453 :22Ø,Ø14,2Ø2,2Ø8,25Ø,238,233
3459 :Ø45,Ø17,16Ø,ØØ5,177,247,Ø14
```

```
3465 :153,015,143,136,208,248,016
3471 :160,006,162,000,177,247,127
3477 :157,027,143,200,232,224,108
3483 :005,208,245,177,247,141,154
3489 :012,143,141,010,143,200,042
3495 :177,247,141,013,143,032,152
3501 :228,014,032,238,014,208,139
3507 :003,076,106,013,032,046,199
3513 :017,240,169,162,000,134,139
3519 :248,160,006,024,032,240,133
3525 :255,173,021,143,205,035,005
3531 :037,240,002,162,012,134,022
3537 :247,189,158,020,240,006,045
3543 :032,210,255,232,208,245,117
3549 :169,063,032,210,255,166,092
3555 :247,160,010,169,044,157,246
3561 :158,020,232,136,208,249,212
3567 :032,228,255,240,251,201,166
3573 :013,240,039,201,032,240,242
3579 :008,201,065,144,239,201,085
3585 :091,176,235,230,248,166,123
3591 :248,224,011,240,019,164,145
3597 :247,153,158,020,041,063,183
3603 :157,005,004,230,247,169,063
3609 :047,157,006,004,208,208,143
3615 :032,038,015,032,046,017,211
3621 :240,149,173,015,143,041,030
3627 :002,024,109,016,143,010,091
3633 :109,018,143,141,022,143,113
3639 :173,017,143,010,010,105,001
3645 :009,056,237,019,143,141,154
3651 :023,143,173,027,212,041,174
3657 :031,010,109,023,143,105,238
3663 :032,141,008,143,173,015,079
3669 :143,041,004,109,019,143,032
3675 :010,109,019,143,109,020,245
3681 :143,141,024,143,010,109,155
3687 :018,143,105,048,141,009,055
3693 :143,173,020,143,009,008,093
3699 :109,016,143,141,025,143,180
3705 :173,015,143,041,007,024,012
3711 :109,018,143,109,017,143,154
3717 :141,026,143,162,000,173,010
3723 :012,143,232,221,127,036,142
3729 :176,250,142,032,143,142,006
3735 :011,143,142,033,143,032,143
3741 :132,027,173,021,143,205,090
3747 :035,037,240,003,076,241,027
3753 :012,173,015,143,041,003,044
3759 :141,129,143,032,243,027,122
```

```
3765 :169,ØØØ,141,129,143,174,169
3771 :Ø33,143,189,127,Ø36,168,115
3777 :2Ø2,189,127,Ø36,17Ø,2Ø2,Ø95
3783 :Ø32,247,Ø27,Ø32,132,Ø27,184
3789 :173,Ø21,143,2Ø5,Ø35,Ø37,Ø51
3795 :2Ø8,213,Ø32,25Ø,Ø26,Ø32,2Ø4
38Ø1 :1Ø8,Ø27,Ø96,169,Ø16,Ø24,145
38Ø7 :1Ø1,247,133,247,Ø96,Ø32,Ø55
3813 :237,ØØ9,Ø32,2Ø5,Ø21,Ø32,253
3819 :Ø38,Ø15,Ø96,169,Ø15,133,189
3825 :253,169,Ø22,133,254,169,217
3831 :Ø29,133,167,162,24Ø,16Ø,114
3837 :Ø16,Ø32,184,Ø2Ø,173,Ø21,187
3843 :143,24Ø,Ø13,162,Ø1Ø,189,248
3849 :117,Ø2Ø,Ø41,Ø63,157,156,Ø51
3855 :ØØ6,2Ø2,2Ø8,245,173,Ø21,1Ø2
3861 :143,2Ø5,Ø35,Ø37,24Ø,ØØ3,172
3867 :238,125,ØØ6,162,Ø2Ø,16Ø,226
3873 :Ø21,Ø32,163,Ø28,Ø96,174,Ø35
3879 :Ø21,143,189,Ø4Ø,Ø37,Ø32,245
3885 :21Ø,255,169,Ø17,133,253,Ø58
3891 :169,Ø25,133,254,169,ØØØ,Ø33
3897 :133,167,162,Ø81,16Ø,Ø16,ØØ8
39Ø3 :Ø32,184,Ø2Ø,169,Ø31,Ø32,Ø19
39Ø9 :21Ø,255,169,Ø2Ø,133,253,Ø85
3915 :169,Ø25,133,254,169,ØØ9,Ø66
3921 :133,167,162,171,16Ø,Ø16,122
3927 :Ø32,184,Ø2Ø,162,Ø19,232,224
3933 :16Ø,Ø15,Ø24,Ø32,24Ø,255,Ø51
3939 :162,Ø49,138,Ø32,21Ø,255,177
3945 :232,224,Ø55,2Ø8,247,Ø56,1Ø3
3951 :Ø32,24Ø,255,224,Ø24,2Ø8,Ø7Ø
3957 :23Ø,173,Ø12,143,Ø1Ø,17Ø,Ø87
3963 :189,22Ø,Ø33,Ø41,Ø63,141,Ø42
3969 :171,ØØ6,232,189,22Ø,Ø33,212
3975 :Ø41,Ø63,141,172,ØØ6,162,2Ø8
3981 :Ø18,16Ø,ØØ2,Ø24,Ø32,24Ø,1Ø5
3987 :255,174,Ø21,143,189,Ø4Ø,2Ø1
3993 :Ø37,Ø32,21Ø,255,174,Ø13,1Ø6
3999 :143,Ø48,Ø14,16Ø,ØØ4,189,2Ø5
4ØØ5 :Ø49,Ø16,Ø32,21Ø,255,232,191
4Ø11 :136,2Ø8,246,24Ø,Ø13,162,152
4Ø17 :ØØØ,189,228,Ø16,24Ø,ØØ6,Ø88
4Ø23 :Ø32,21Ø,255,232,2Ø8,245,Ø85
4Ø29 :169,158,133,247,169,Ø2Ø,Ø61
4Ø35 :133,248,16Ø,ØØØ,173,Ø21,162
4Ø41 :143,2Ø5,Ø35,Ø37,24Ø,ØØ2,Ø95
4Ø47 :16Ø,Ø12,177,247,24Ø,ØØ6,Ø25
4Ø53 :Ø32,21Ø,255,2ØØ,2Ø8,246,Ø84
4Ø59 :173,Ø21,143,24Ø,Ø32,162,222
```

```
4Ø65 :Ø1Ø,189,117,Ø2Ø,Ø41,Ø63,153
4Ø71 :157,248,ØØ6,2Ø2,2Ø8,245,Ø17
4Ø77 :169,Ø2Ø,133,253,169,Ø22,235
4Ø83 :133,254,169,ØØ9,133,167,Ø84
4Ø89 :162,21Ø,16Ø,Ø16,Ø32,184,245
4Ø95 :Ø2Ø,162,ØØ4,16Ø,16Ø,189,182
41Ø1 :Ø16,143,ØØ9,Ø48,153,Ø39,157
41Ø7 :ØØ7,152,Ø56,233,Ø4Ø,168,155
4113 :2Ø2,Ø16,24Ø,162,ØØ4,16Ø,Ø33
4119 :16Ø,152,Ø24,125,Ø27,143,142
4125 :168,185,Ø46,ØØ7,ØØ9,Ø64,252
4131 :153,Ø46,ØØ7,152,Ø56,233,17Ø
4137 :Ø4Ø,Ø41,248,168,2Ø2,Ø16,244
4143 :232,Ø96,Ø83,Ø69,Ø78,Ø32,125
4149 :Ø71,Ø79,Ø86,Ø32,Ø82,Ø69,216
4155 :Ø8Ø,Ø32,Ø82,Ø69,Ø86,Ø32,184
4161 :Ø32,Ø77,Ø83,Ø32,Ø68,Ø82,183
4167 :Ø62,Ø32,Ø86,Ø61,Ø8Ø,Ø32,168
4173 :Ø71,Ø69,Ø78,Ø32,Ø27,Ø44,142
4179 :ØØØ,Ø27,Ø44,ØØØ,255,Ø44,197
4185 :Ø68,Ø69,Ø77,Ø79,Ø67,Ø82,Ø19
4191 :Ø65,Ø84,Ø73,Ø67,Ø32,Ø67,227
4197 :Ø65,Ø78,Ø68,Ø73,Ø68,Ø65,ØØ6
42Ø3 :Ø84,Ø69,Ø32,Ø44,Ø44,Ø44,168
42Ø9 :Ø44,Ø44,Ø44,ØØØ,255,156,144
4215 :Ø47,Ø32,Ø67,Ø72,Ø65,Ø82,228
4221 :Ø32,Ø88,ØØØ,255,Ø47,Ø32,Ø67
4227 :Ø83,Ø84,Ø65,Ø77,Ø32,Ø88,Ø48
4233 :ØØØ,255,Ø47,Ø32,Ø73,Ø78,11Ø
4239 :Ø84,Ø76,Ø32,Ø88,ØØØ,255,166
4245 :Ø47,Ø32,Ø69,Ø88,Ø8Ø,Ø82,Ø35
4251 :Ø32,Ø88,ØØØ,255,Ø47,Ø32,Ø97
4257 :Ø65,Ø8Ø,Ø8Ø,Ø76,Ø32,Ø88,Ø7Ø
4263 :ØØØ,ØØØ,ØØØ,ØØØ,255,Ø31,197
4269 :Ø85,Ø78,Ø69,Ø77,Ø8Ø,ØØØ,Ø5Ø
4275 :255,Ø8Ø,Ø79,Ø86,Ø84,Ø89,Ø84
4281 :ØØØ,255,Ø65,Ø71,Ø82,Ø73,219
4287 :Ø67,ØØØ,255,Ø69,Ø68,Ø85,223
4293 :Ø67,Ø78,ØØØ,255,Ø68,Ø7Ø,223
4299 :Ø69,Ø78,Ø83,ØØØ,ØØØ,ØØØ,177
43Ø5 :ØØØ,255,Ø31,Ø73,Ø78,Ø7Ø,2Ø4
4311 :Ø76,Ø78,ØØØ,255,Ø67,Ø82,ØØ5
4317 :Ø73,Ø77,Ø69,ØØØ,ØØØ,ØØØ,184
4323 :ØØØ,157,Ø8Ø,Ø82,Ø69,Ø83,186
4329 :Ø73,Ø68,Ø69,Ø78,Ø84,Ø32,125
4335 :ØØØ,255,151,Ø8Ø,Ø76,Ø65,Ø98
4341 :Ø89,Ø69,Ø82,Ø32,Ø91,Ø49,145
4347 :ØØØ,255,Ø68,Ø69,Ø77,Ø79,Ø31
4353 :Ø67,Ø82,Ø65,Ø84,Ø73,Ø67,183
4359 :ØØØ,255,Ø67,Ø79,Ø78,Ø86,Ø6Ø
```

```
4365 :069,078,084,073,079,078,218
4371 :000,010,166,000,005,032,232
4377 :000,255,030,032,047,032,165
4383 :078,079,000,255,032,047,010
4389 :032,089,069,083,000,000,054
4395 :000,000,000,173,005,004,225
4401 :072,169,000,133,162,133,206
4407 :198,169,032,197,162,208,253
4413 :252,162,023,189,098,017,034
4419 :041,063,157,004,004,202,026
4425 :016,245,032,026,031,173,084
4431 :107,031,240,248,162,023,122
4437 :104,157,004,004,202,016,060
4443 :250,173,107,031,041,016,197
4449 :096,058,070,073,082,069,033
4455 :066,085,084,084,079,078,067
4461 :032,084,079,032,067,079,226
4467 :078,084,073,078,085,069,070
4473 :058,173,035,037,205,021,138
4479 :143,208,011,238,036,037,032
4485 :173,036,037,201,010,208,030
4491 :001,096,032,237,009,032,034
4497 :205,021,032,038,015,169,113
4503 :007,141,000,143,032,244,206
4509 :020,162,005,160,012,032,036
4515 :163,028,170,208,003,076,043
4521 :003,018,202,208,003,076,167
4527 :147,018,202,208,003,076,061
4533 :197,018,202,208,003,076,117
4539 :239,018,202,208,006,032,124
4545 :043,029,076,155,017,202,203
4551 :208,008,032,022,019,208,184
4557 :205,076,003,018,202,208,149
4563 :014,032,046,017,240,197,245
4569 :173,011,143,141,032,143,092
4575 :076,141,017,202,240,003,134
4581 :076,155,017,076,200,019,004
4587 :162,000,169,128,024,109,059
4593 :032,143,168,169,000,133,118
4599 :253,169,014,133,254,169,215
4605 :030,133,167,076,184,020,095
4611 :032,235,017,174,032,143,124
4617 :189,127,036,202,056,253,104
4623 :127,036,072,105,003,168,014
4629 :162,003,032,163,028,201,098
4635 :000,208,007,032,043,029,090
4641 :104,076,006,018,133,002,116
4647 :104,197,002,176,003,076,085
4653 :155,017,198,002,174,032,111
```

```
4659 :143,202,189,127,036,024,004
4665 :101,002,174,000,143,157,122
4671 :000,143,133,251,134,252,208
4677 :032,250,019,169,030,032,089
4683 :210,255,165,251,010,170,112
4689 :189,220,033,032,210,255,252
4695 :189,221,033,032,210,255,003
4701 :169,032,032,210,255,189,212
4707 :000,120,072,170,169,000,118
4713 :032,205,189,104,201,010,078
4719 :176,005,169,032,032,210,223
4725 :255,169,032,032,210,255,046
4731 :169,152,032,210,255,173,090
4737 :032,143,009,048,032,210,091
4743 :255,206,000,143,208,003,182
4749 :076,007,020,076,006,018,088
4755 :174,000,143,169,240,157,006
4761 :000,143,134,252,032,250,196
4767 :019,169,129,032,210,255,205
4773 :162,000,189,112,021,240,121
4779 :006,032,210,255,232,208,090
4785 :245,173,032,143,009,048,059
4791 :032,210,255,206,000,143,005
4797 :208,003,076,007,020,076,067
4803 :155,017,174,000,143,169,085
4809 :255,157,000,143,134,252,118
4815 :032,250,019,169,154,032,095
4821 :210,255,162,000,189,125,130
4827 :021,240,006,032,210,255,215
4833 :232,208,245,206,000,143,235
4839 :208,003,076,007,020,076,109
4845 :155,017,174,000,143,169,127
4851 :000,157,000,143,134,252,161
4857 :032,250,019,169,155,032,138
4863 :210,255,169,090,162,005,122
4869 :032,210,255,202,208,250,138
4875 :206,000,143,208,003,076,135
4881 :007,020,076,155,017,173,209
4887 :009,143,201,010,176,001,051
4893 :096,206,009,143,032,153,156
4899 :033,174,032,143,189,127,221
4905 :036,133,248,202,189,127,208
4911 :036,133,247,169,150,133,147
4917 :249,169,004,133,250,169,003
4923 :047,133,251,133,252,166,017
4929 :247,228,248,208,003,076,051
4935 :046,017,165,249,024,105,165
4941 :040,133,249,144,002,230,107
4947 :250,189,000,144,133,253,028
4953 :133,254,162,004,006,254,134
```

```
4959 :202,208,251,006,254,176,168
4965 :028,169,037,133,251,006,213
4971 :254,176,020,169,032,133,123
4977 :251,006,254,176,012,169,213
4983 :037,133,252,006,254,176,209
4989 :004,169,032,133,252,160,107
4995 :000,169,032,145,249,200,158
5001 :165,251,145,249,200,165,032
5007 :252,145,249,169,047,133,114
5013 :251,133,252,006,253,176,196
5019 :028,169,042,133,252,006,017
5025 :253,176,020,169,032,133,176
5031 :252,006,253,176,012,169,011
5037 :042,133,251,006,253,176,010
5043 :004,169,032,133,251,160,160
5049 :007,165,251,145,249,200,178
5055 :165,252,145,249,230,247,199
5061 :076,058,019,032,103,023,252
5067 :174,032,143,232,232,232,224
5073 :160,031,024,032,240,255,183
5079 :169,058,032,210,255,162,077
5085 :003,160,013,032,163,028,108
5091 :201,000,208,006,032,043,205
5097 :029,076,200,019,201,010,000
5103 :208,003,076,155,017,141,071
5109 :032,143,076,155,017,169,069
5115 :022,056,229,252,170,160,116
5121 :032,024,032,240,255,096,168
5127 :032,046,017,208,003,076,133
5133 :217,017,032,104,025,032,184
5139 :250,026,032,108,027,032,238
5145 :132,027,076,122,017,169,056
5151 :000,133,253,169,010,133,217
5157 :254,169,030,133,167,162,184
5163 :049,160,020,076,184,020,040
5169 :255,018,144,160,213,211,026
5175 :197,160,202,207,217,189,203
5181 :160,000,255,160,211,212,035
5187 :201,195,203,160,207,210,219
5193 :160,000,255,201,202,203,070
5199 :204,146,205,018,160,203,247
5205 :197,217,211,000,255,018,215
5211 :155,080,076,091,049,032,062
5217 :080,065,082,084,089,146,131
5223 :000,255,031,068,069,077,091
5229 :079,067,082,065,084,073,047
5235 :067,000,255,082,069,080,156
5241 :085,066,076,073,067,065,041
5247 :078,000,255,018,155,032,153
5253 :073,078,067,085,077,066,067
```

```
5259 :069,078,084,146,000,255,003
5265 :031,032,032,032,078,079,173
5271 :078,069,032,032,032,000,138
5277 :255,032,080,076,065,089,242
5283 :069,082,032,049,032,000,171
5289 :255,032,080,076,065,089,254
5295 :069,082,032,050,032,000,184
5301 :000,000,000,134,251,132,186
5307 :252,208,011,200,152,024,010
5313 :101,251,133,251,144,002,051
5319 :230,252,166,253,228,254,046
5325 :208,001,096,230,253,164,133
5331 :167,024,032,240,255,160,065
5337 :000,162,255,177,251,016,054
5343 :016,200,177,251,240,217,044
5349 :032,210,255,202,016,250,170
5355 :240,209,200,208,241,170,223
5361 :200,208,237,169,000,133,164
5367 :253,169,014,133,254,169,215
5373 :146,032,210,255,169,144,185
5379 :032,210,255,169,030,133,064
5385 :167,162,072,160,021,032,111
5391 :184,020,174,021,143,189,234
5397 :037,037,041,063,141,071,155
5403 :004,173,036,037,009,048,078
5409 :141,078,004,173,032,143,092
5415 :010,010,010,024,109,032,234
5421 :143,170,173,032,143,009,203
5427 :048,141,150,004,160,000,042
5433 :189,037,036,041,063,240,151
5439 :007,153,152,004,232,200,043
5445 :208,242,096,009,058,000,170
5451 :255,032,032,032,087,069,070
5457 :069,075,032,032,032,000,065
5463 :009,058,000,009,032,000,195
5469 :009,032,000,255,031,032,196
5475 :032,067,065,077,080,065,229
5481 :073,071,078,000,255,032,102
5487 :032,084,086,032,065,068,222
5493 :083,032,032,157,000,255,164
5499 :032,032,070,085,078,068,232
5505 :082,065,073,083,000,255,175
5511 :032,032,082,069,083,084,005
5517 :032,032,032,032,000,255,012
5523 :018,155,032,077,065,080,062
5529 :032,032,032,032,032,032,089
5535 :000,255,032,080,079,076,169
5541 :076,032,032,032,032,032,145
5547 :000,255,146,150,082,069,105
5553 :067,079,078,083,073,068,113
```

```
5559 :069,082,000,255,084,082,243
5565 :065,086,069,076,032,032,037
5571 :032,032,154,000,009,032,198
5577 :000,000,000,000,169,014,128
5583 :133,253,169,025,133,254,150
5589 :169,028,133,167,169,030,141
5595 :032,210,255,162,034,160,048
5601 :023,032,184,020,169,043,184
5607 :141,076,006,169,046,141,042
5613 :140,007,173,036,037,208,070
5619 :011,169,020,162,008,032,133
5625 :210,255,202,208,250,096,190
5631 :174,008,143,169,000,032,013
5637 :205,189,162,023,160,030,006
5643 :024,032,240,255,169,030,249
5649 :032,210,255,174,009,143,072
5655 :224,100,176,022,169,032,234
5661 :032,210,255,224,010,176,168
5667 :013,032,210,255,032,210,019
5673 :255,138,009,048,032,210,221
5679 :255,096,169,000,032,205,036
5685 :189,173,184,007,141,185,164
5691 :007,162,006,173,027,212,134
5697 :041,015,201,010,176,247,243
5703 :009,048,157,185,007,202,167
5709 :208,239,169,060,141,184,054
5715 :007,141,188,007,032,122,068
5721 :022,169,052,133,248,169,114
5727 :000,133,247,168,162,002,039
5733 :149,252,202,016,251,032,235
5739 :166,022,169,032,162,002,148
5745 :149,249,202,016,251,032,244
5751 :211,022,096,169,017,133,255
5757 :252,169,000,133,251,166,072
5763 :251,228,252,208,001,096,143
5769 :160,000,024,032,240,255,080
5775 :162,000,189,040,037,032,091
5781 :210,255,169,037,032,210,038
5787 :255,232,224,003,208,240,037
5793 :230,251,076,130,022,166,012
5799 :247,232,232,134,247,200,179
5805 :196,248,208,001,096,189,087
5811 :000,120,074,133,002,185,181
5817 :000,144,162,002,041,238,004
5823 :240,006,202,041,014,240,166
5829 :001,202,181,252,024,101,190
5835 :002,176,216,149,252,076,050
5841 :166,022,169,004,133,248,183
5847 :169,000,133,247,160,002,158
5853 :169,015,133,002,185,037,250
```

```
5859 :Ø37,Ø41,Ø63,145,247,136,128
5865 :Ø16,246,169,Ø4Ø,Ø24,1Ø1,Ø61
5871 :247,133,247,169,ØØØ,1Ø1,112
5877 :248,133,248,16Ø,ØØ2,162,174
5883 :ØØ2,169,Ø16,Ø24,117,252,Ø63
5889 :149,252,176,Ø13,181,249,253
5895 :145,247,2Ø2,136,Ø16,239,224
59Ø1 :198,ØØ2,Ø16,218,Ø96,Ø72,1Ø3
59Ø7 :169,Ø37,149,249,1Ø4,Ø74,Ø33
5913 :Ø74,Ø74,Ø41,ØØ1,ØØ9,Ø36,ØØ4
5919 :Ø76,ØØ7,Ø23,Ø11,Ø35,ØØØ,183
5925 :255,Ø42,Ø32,Ø83,ØØØ,255,192
5931 :Ø42,Ø32,Ø77,ØØØ,255,Ø42,235
5937 :Ø32,Ø84,ØØØ,255,Ø42,Ø32,238
5943 :Ø87,ØØØ,255,Ø42,Ø32,Ø84,Ø43
5949 :ØØØ,255,Ø42,Ø32,Ø7Ø,ØØØ,2Ø4
5955 :255,Ø42,Ø32,Ø83,ØØØ,Ø11,234
5961 :Ø44,ØØØ,255,Ø92,Ø32,Ø48,Ø32
5967 :Ø48,Ø6Ø,Ø48,Ø48,Ø48,Ø6Ø,135
5973 :Ø48,Ø48,Ø48,ØØØ,255,154,126
5979 :Ø32,Ø72,Ø69,Ø65,Ø76,Ø84,233
5985 :Ø72,Ø32,ØØØ,ØØØ,ØØØ,ØØØ,2Ø1
5991 :169,Ø28,Ø32,21Ø,255,169,198
5997 :ØØØ,133,253,169,Ø15,133,Ø44
6ØØ3 :254,169,Ø3Ø,133,167,162,ØØ6
6ØØ9 :ØØØ,16Ø,128,Ø32,184,Ø2Ø,133
6Ø15 :Ø96,169,128,133,248,169,Ø46
6Ø21 :ØØØ,133,247,169,ØØØ,133,Ø47
6Ø27 :25Ø,133,249,169,ØØØ,133,Ø49
6Ø33 :253,133,254,168,162,ØØ1,Ø92
6Ø39 :Ø32,241,Ø23,162,ØØØ,16Ø,ØØ1
6Ø45 :ØØ4,189,Ø37,Ø36,145,247,Ø47
6Ø51 :2ØØ,232,224,ØØ8,2Ø8,245,ØØØ
6Ø57 :162,ØØ8,172,Ø54,Ø25,136,214
6Ø63 :23Ø,253,165,253,2Ø1,Ø1Ø,ØØ7
6Ø69 :2Ø8,ØØ3,Ø76,225,Ø23,169,117
6Ø75 :255,145,247,2ØØ,169,Ø28,2Ø7
6Ø81 :145,247,2ØØ,169,Ø42,145,117
6Ø87 :247,2ØØ,165,253,ØØ9,Ø48,Ø97
6Ø93 :145,247,2ØØ,169,Ø31,145,118
6Ø99 :247,2ØØ,232,189,Ø37,Ø36,128
61Ø5 :145,247,2Ø8,247,2ØØ,Ø76,Ø6Ø
6111 :175,Ø23,Ø32,ØØ2,Ø24,169,136
6117 :ØØØ,162,ØØ4,145,247,2ØØ,219
6123 :2Ø2,2Ø8,25Ø,Ø76,Ø17,Ø24,244
6129 :162,ØØ1,16Ø,ØØØ,189,Ø54,Ø39
6135 :Ø25,145,247,2ØØ,232,236,Ø52
6141 :Ø54,Ø25,2Ø8,244,Ø96,162,Ø18
6147 :ØØ1,189,Ø89,Ø25,145,247,187
6153 :2ØØ,232,236,Ø89,Ø25,2Ø8,231
```

```
6159 :244,096,169,001,133,253,143
6165 :133,254,208,009,230,253,084
6171 :165,253,201,010,208,001,097
6177 :096,230,248,169,009,024,041
6183 :101,249,133,249,169,000,172
6189 :101,250,133,250,032,241,028
6195 :023,166,249,160,004,165,050
6201 :253,073,048,145,247,200,255
6207 :200,189,037,036,240,006,003
6213 :145,247,232,200,208,245,066
6219 :166,253,189,127,036,133,211
6225 :250,172,054,025,136,165,115
6231 :254,010,170,169,048,133,103
6237 :251,133,252,169,255,145,018
6243 :247,200,169,028,145,247,111
6249 :200,169,042,145,247,200,084
6255 :169,154,145,247,200,165,167
6261 :254,201,010,144,007,230,195
6267 :252,233,010,076,118,024,068
6273 :101,251,133,251,165,252,002
6279 :145,247,200,165,251,145,008
6285 :247,200,169,032,145,247,157
6291 :200,169,151,145,247,200,235
6297 :189,220,033,145,247,200,163
6303 :232,189,220,033,145,247,201
6309 :200,202,169,032,145,247,136
6315 :200,169,048,133,251,133,081
6321 :252,189,000,120,201,010,181
6327 :144,007,230,252,233,010,035
6333 :076,181,024,101,251,133,187
6339 :251,165,252,145,247,200,175
6345 :165,251,145,247,200,169,098
6351 :032,145,247,200,169,000,232
6357 :145,247,200,230,254,165,174
6363 :254,197,250,240,003,076,215
6369 :086,024,032,002,024,165,046
6375 :250,133,254,166,253,202,209
6381 :189,127,036,133,002,232,188
6387 :189,127,036,056,229,002,114
6393 :133,002,169,008,229,002,024
6399 :133,002,048,038,169,009,142
6405 :145,247,200,169,035,145,178
6411 :247,200,169,000,145,247,251
6417 :200,198,002,048,019,169,141
6423 :009,145,247,200,169,032,057
6429 :145,247,200,169,000,145,167
6435 :247,200,198,002,016,237,167
6441 :169,000,162,004,145,247,000
6447 :200,202,208,250,076,025,240
6453 :024,035,009,035,000,255,155
```

```
6459 :032,032,032,032,032,032,251
6465 :032,032,032,032,000,009,202
6471 :044,000,255,018,154,037,067
6477 :144,205,193,208,160,160,123
6483 :160,160,160,160,146,000,101
6489 :014,255,028,042,077,069,062
6495 :078,085,032,032,032,032,130
6501 :032,000,000,173,011,143,204
6507 :141,032,143,169,008,141,229
6513 :000,143,206,000,143,208,045
6519 :001,096,174,000,143,189,210
6525 :000,143,208,009,032,177,182
6531 :026,032,208,026,076,115,102
6537 :025,016,023,106,176,003,230
6543 :076,166,027,173,024,143,240
6549 :010,109,009,143,144,002,054
6555 :169,255,141,009,143,076,180
6561 :115,025,172,009,143,240,097
6567 :203,072,162,000,232,221,033
6573 :127,036,176,250,236,011,241
6579 :143,240,009,142,032,143,120
6585 :142,011,143,206,009,143,071
6591 :104,032,207,025,032,233,056
6597 :025,032,093,026,032,140,033
6603 :026,076,115,025,133,002,068
6609 :133,251,198,251,165,251,178
6615 :010,010,024,101,251,133,232
6621 :251,133,253,169,146,133,026
6627 :252,169,145,133,254,096,252
6633 :173,008,143,041,248,208,030
6639 :005,169,001,133,255,096,130
6645 :169,003,024,109,021,143,202
6651 :168,177,251,133,255,173,128
6657 :010,143,016,003,230,255,146
6663 :096,197,002,208,009,169,176
6669 :002,032,087,026,169,255,072
6675 :133,002,165,002,141,010,216
6681 :143,173,008,143,160,005,145
6687 :074,136,208,252,032,087,052
6693 :026,173,022,143,032,087,008
6699 :026,160,006,136,208,001,068
6705 :096,185,026,143,209,253,193
6711 :208,007,169,003,032,087,049
6717 :026,208,238,170,202,138,019
6723 :209,253,208,007,169,001,146
6729 :032,087,026,208,224,232,114
6735 :232,138,209,253,208,217,056
6741 :240,240,024,101,255,133,054
6747 :255,096,160,005,177,251,011
6753 :056,229,255,176,004,198,247
```

```
6759  :255,208,243,145,251,165,090
6765  :255,170,172,021,143,200,046
6771  :024,113,251,144,002,169,050
6777  :255,145,251,152,073,003,232
6783  :168,138,074,074,113,251,177
6789  :144,002,169,255,145,251,075
6795  :096,070,255,208,001,096,097
6801  :173,008,143,056,229,255,241
6807  :176,002,169,000,141,008,135
6813  :143,070,255,208,001,096,162
6819  :173,009,143,056,229,255,004
6825  :176,002,169,000,141,009,154
6831  :143,096,160,000,162,015,239
6837  :173,021,143,240,002,162,154
6843  :240,134,251,162,052,202,204
6849  :208,003,132,002,096,189,055
6855  :000,144,037,251,240,243,090
6861  :200,208,240,165,002,024,020
6867  :109,023,143,010,109,022,115
6873  :143,109,008,143,144,003,255
6879  :024,169,255,141,008,143,195
6885  :173,010,143,208,010,169,174
6891  :016,109,008,143,176,003,178
6897  :141,008,143,169,000,141,075
6903  :010,143,096,169,146,133,176
6909  :252,169,000,133,251,169,203
6915  :000,170,240,007,160,005,073
6921  :230,251,136,208,251,232,037
6927  :224,052,208,001,096,160,244
6933  :001,177,251,200,056,241,179
6939  :251,208,006,032,088,027,127
6945  :076,007,027,176,010,234,051
6951  :073,255,024,105,001,160,145
6957  :128,208,002,160,008,133,172
6963  :253,132,254,041,224,240,171
6969  :002,208,020,070,254,165,008
6975  :253,041,016,240,002,208,055
6981  :010,070,254,165,253,041,094
6987  :008,208,002,070,254,165,014
6993  :254,157,000,144,076,007,207
6999  :027,173,000,144,041,240,200
7005  :240,004,169,001,208,002,205
7011  :169,016,141,000,144,157,214
7017  :000,144,096,162,052,202,249
7023  :240,018,189,000,144,041,231
7029  :015,240,004,169,067,208,052
7035  :002,169,130,157,192,055,060
7041  :208,235,096,173,021,143,237
7047  :072,162,063,189,064,143,060
```

```
7053 :157,128,143,189,000,143,133
7059 :157,064,143,189,128,143,203
7065 :157,000,143,202,208,235,074
7071 :104,073,001,141,021,143,130
7077 :096,173,009,143,201,040,059
7083 :144,067,174,032,143,189,152
7089 :127,036,133,250,202,189,090
7095 :127,036,133,249,198,249,151
7101 :173,036,037,010,024,109,066
7107 :025,143,133,255,230,249,206
7113 :165,249,197,250,240,014,036
7119 :032,023,028,070,255,032,135
7125 :093,026,032,140,026,076,094
7131 :189,027,173,009,143,056,048
7137 :237,025,143,144,005,237,248
7143 :025,143,176,002,169,001,235
7149 :141,009,143,076,115,025,234
7155 :162,000,160,052,134,249,232
7161 :132,250,230,249,165,249,244
7167 :197,250,240,019,174,129,240
7173 :143,134,255,032,023,028,108
7179 :032,044,026,070,255,032,214
7185 :093,026,076,251,027,096,074
7191 :032,207,025,032,038,026,127
7197 :096,162,000,169,000,157,101
7203 :000,063,202,208,250,169,159
7209 :000,170,168,185,010,031,093
7215 :157,000,063,185,018,031,245
7221 :157,064,063,232,232,232,009
7227 :200,192,007,208,236,185,063
7233 :010,031,157,000,063,157,227
7239 :001,063,157,002,063,185,030
7245 :018,031,157,064,063,169,067
7251 :252,141,248,007,169,253,129
7257 :141,249,007,162,007,169,056
7263 :012,157,039,208,202,016,217
7269 :250,169,001,141,029,208,131
7275 :169,001,141,016,208,169,043
7281 :004,141,000,208,169,050,173
7287 :141,001,208,169,054,141,065
7293 :002,208,169,056,141,003,192
7299 :208,169,000,160,004,153,057
7305 :002,031,136,016,250,169,229
7311 :034,141,007,031,169,173,186
7317 :141,006,031,169,054,141,179
7323 :009,031,169,000,141,008,001
7329 :031,096,169,000,133,253,075
7335 :169,004,141,000,208,152,073
7341 :032,250,030,133,252,138,240
7347 :032,250,030,133,251,141,248
```

```
7353 :001,208,169,012,141,039,243
7359 :208,173,016,208,009,001,038
7365 :141,016,208,173,021,208,196
7371 :009,001,141,021,208,032,103
7377 :026,031,173,107,031,240,049
7383 :248,041,019,240,244,170,153
7389 :041,016,208,039,138,041,192
7395 :001,240,017,173,001,208,099
7401 :197,251,240,227,198,253,063
7407 :056,233,008,141,001,208,118
7413 :208,217,173,001,208,197,225
7419 :252,240,210,230,253,024,180
7425 :105,008,141,001,208,208,160
7431 :200,169,000,141,039,208,252
7437 :032,026,031,173,107,031,157
7443 :240,248,041,016,208,007,011
7449 :169,012,141,039,208,208,034
7455 :176,173,021,208,041,254,136
7461 :141,021,208,165,253,096,153
7467 :162,007,189,002,031,149,071
7473 :247,202,016,248,169,001,164
7479 :141,040,208,173,021,208,078
7485 :009,002,141,021,208,032,218
7491 :026,031,173,107,031,240,163
7497 :248,106,176,020,106,176,137
7503 :067,106,176,110,106,176,052
7509 :005,106,176,005,144,231,240
7515 :076,240,029,076,231,030,005
7521 :165,248,240,221,173,003,123
7527 :208,056,233,004,141,003,236
7533 :208,198,248,165,248,106,002
7539 :176,003,076,036,030,165,089
7545 :253,233,026,133,253,176,171
7551 :002,198,254,165,251,056,029
7557 :233,025,133,251,144,003,154
7563 :076,036,030,198,252,076,039
7569 :036,030,165,248,201,029,086
7575 :240,169,173,003,208,024,200
7581 :105,004,141,003,208,230,080
7587 :248,165,248,106,176,123,205
7593 :165,253,105,026,133,253,080
7599 :144,002,230,254,165,251,197
7605 :024,105,025,133,251,144,095
7611 :104,230,252,076,036,030,147
7617 :165,247,208,003,076,066,190
7623 :029,173,002,208,056,233,132
7629 :004,141,002,208,198,247,237
7635 :165,247,106,144,076,165,090
7641 :253,233,001,133,253,176,242
```

```
7647 :002,198,254,165,251,056,125
7653 :233,001,133,251,176,057,056
7659 :198,252,076,036,030,165,224
7665 :247,201,049,208,003,076,001
7671 :066,029,173,002,208,024,237
7677 :105,004,141,002,208,230,175
7683 :247,165,247,106,144,003,147
7689 :076,036,030,165,253,105,162
7695 :001,133,253,144,002,230,010
7701 :254,165,251,024,105,001,053
7707 :133,251,144,005,230,252,018
7713 :076,036,030,169,001,133,222
7719 :249,165,248,074,144,004,155
7725 :006,249,006,249,165,247,199
7731 :106,176,002,006,249,160,238
7737 :000,177,251,133,002,165,017
7743 :249,049,253,208,038,169,005
7749 :192,036,002,048,013,165,013
7755 :002,041,063,170,189,120,148
7761 :034,133,002,076,106,030,206
7767 :080,007,169,000,133,002,222
7773 :076,106,030,165,002,041,001
7779 :063,170,189,121,034,133,041
7785 :002,162,015,160,016,024,228
7791 :032,240,255,169,149,032,220
7797 :210,255,169,032,162,007,184
7803 :032,210,255,202,016,250,064
7809 :169,157,162,007,032,210,098
7815 :255,202,016,250,165,002,001
7821 :208,003,076,066,029,041,052
7827 :063,010,170,189,220,033,064
7833 :032,210,255,189,221,033,069
7839 :032,210,255,169,032,032,121
7845 :210,255,189,000,120,170,085
7851 :201,010,176,005,169,032,252
7857 :032,210,255,169,000,032,107
7863 :205,189,169,029,032,210,249
7869 :255,169,144,032,210,255,230
7875 :169,018,032,210,255,169,024
7881 :160,032,210,255,165,002,001
7887 :041,063,162,000,232,221,158
7893 :127,036,176,250,138,105,021
7899 :176,032,210,255,169,146,183
7905 :032,210,255,076,066,029,125
7911 :173,021,208,041,253,141,044
7917 :021,208,162,007,181,247,039
7923 :157,002,031,202,016,248,131
7929 :096,234,010,010,010,024,121
7935 :105,050,096,000,000,000,250
7941 :000,000,000,000,000,192,197
```

```
7947  :192,224,240,224,192,200,003
7953  :255,255,153,129,195,195,175
7959  :129,153,255,169,000,141,102
7965  :107,031,173,000,220,041,089
7971  :031,073,031,208,045,173,084
7977  :001,220,041,031,073,031,182
7983  :208,036,032,228,255,208,246
7989  :001,096,056,233,073,144,144
7995  :222,170,232,233,005,176,073
8001  :216,138,041,002,240,004,194
8007  :138,073,001,170,169,000,110
8013  :141,107,031,056,042,202,144
8019  :208,252,141,107,031,173,227
8025  :000,220,045,001,220,041,104
8031  :016,240,246,169,006,101,105
8037  :162,197,162,208,252,096,154
8043  :000,032,250,026,032,177,112
8049  :026,165,002,201,026,144,165
8055  :003,032,132,027,032,122,211
8061  :033,032,132,027,032,122,247
8067  :033,169,001,032,207,025,086
8073  :160,005,177,251,074,074,110
8079  :170,160,002,138,024,113,238
8085  :251,144,002,165,255,145,087
8091  :251,136,208,243,160,002,131
8097  :209,251,208,019,160,003,243
8103  :177,251,200,056,241,251,063
8109  :169,128,042,168,200,177,033
8115  :251,233,001,145,251,165,201
8121  :251,024,105,005,133,251,186
8127  :201,255,208,198,032,250,055
8133  :026,032,177,026,162,051,159
8139  :189,000,144,041,017,240,066
8145  :003,030,000,144,202,208,028
8151  :243,032,108,027,032,237,126
8157  :009,032,087,022,032,090,237
8163  :011,032,205,021,162,015,161
8169  :134,002,160,029,024,032,102
8175  :240,255,169,152,032,210,017
8181  :255,169,032,162,011,032,138
8187  :210,255,202,208,250,230,070
8193  :002,166,002,224,024,208,115
8199  :227,173,100,007,141,140,027
8205  :007,141,180,007,141,220,197
8211  :007,169,032,162,011,157,045
8217  :220,007,202,208,250,169,057
8223  :020,141,226,007,169,000,082
8229  :162,003,149,003,202,016,060
8235  :251,169,009,133,174,169,180
8241  :000,141,032,143,238,032,123
```

```
8247 :143,173,Ø32,143,2Ø1,Ø1Ø,245
8253 :2Ø8,ØØ3,Ø76,Ø48,Ø32,Ø32,2Ø4
8259 :153,Ø33,169,ØØØ,133,178,221
8265 :133,179,162,ØØ4,134,251,168
8271 :16Ø,Ø31,132,252,169,19Ø,245
8277 :133,247,133,249,169,ØØ4,252
8283 :133,248,133,25Ø,166,167,164
8289 :16Ø,ØØ3,169,Ø32,145,247,Ø85
8295 :136,Ø16,251,165,247,Ø24,174
83Ø1 :1Ø5,Ø4Ø,133,247,144,ØØ2,Ø12
83Ø7 :23Ø,248,2Ø2,2Ø8,233,174,13Ø
8313 :Ø32,143,189,127,Ø36,133,Ø13
8319 :254,2Ø2,189,127,Ø36,133,Ø44
8325 :253,166,251,164,252,Ø24,219
8331 :Ø32,24Ø,255,166,253,189,25Ø
8337 :ØØØ,144,Ø41,Ø15,2Ø8,ØØ3,Ø44
8343 :Ø76,111,Ø33,189,Ø68,Ø34,15Ø
8349 :17Ø,Ø24,1Ø1,178,133,178,173
8355 :138,2Ø1,Ø1Ø,176,ØØ5,169,Ø94
8361 :Ø32,Ø32,21Ø,255,169,154,253
8367 :Ø32,21Ø,255,169,ØØØ,Ø32,1Ø5
8373 :2Ø5,189,166,251,16Ø,Ø37,165
8379 :Ø24,Ø32,24Ø,255,16Ø,ØØ3,133
8385 :169,Ø32,Ø32,21Ø,255,136,ØØ3
8391 :2Ø8,25Ø,23Ø,251,23Ø,253,Ø85
8397 :198,167,2Ø8,181,165,174,Ø18
84Ø3 :2Ø8,ØØ3,Ø76,1Ø5,Ø33,173,Ø41
84Ø9 :Ø32,143,Ø24,1Ø5,Ø14,17Ø,193
8415 :16Ø,Ø31,Ø24,Ø32,24Ø,255,197
8421 :169,154,Ø32,21Ø,255,165,19Ø
8427 :178,17Ø,2Ø1,Ø1Ø,176,ØØ5,2Ø7
8433 :169,Ø32,Ø32,21Ø,255,169,Ø84
8439 :ØØØ,Ø32,2Ø5,189,169,156,23Ø
8445 :Ø32,21Ø,255,169,Ø32,Ø72,255
8451 :Ø32,21Ø,255,173,Ø32,143,Ø8Ø
8457 :ØØ9,Ø48,Ø32,21Ø,255,1Ø4,155
8463 :Ø32,21Ø,255,Ø32,21Ø,255,241
8469 :165,179,17Ø,2Ø1,Ø1Ø,176,154
8475 :ØØ5,169,Ø32,Ø32,21Ø,255,218
8481 :169,Ø28,Ø32,21Ø,255,169,128
8487 :ØØØ,Ø32,2Ø5,189,162,Ø24,139
8493 :16Ø,Ø3Ø,Ø24,Ø32,24Ø,255,Ø18
8499 :169,152,Ø32,21Ø,255,165,Ø1Ø
85Ø5 :178,Ø24,1Ø1,ØØ3,133,ØØ3,243
8511 :169,ØØØ,1Ø1,ØØ4,133,ØØ4,218
8517 :165,179,1Ø1,ØØ5,133,ØØ5,145
8523 :169,ØØØ,1Ø1,ØØ6,133,ØØ6,234
8529 :166,ØØ3,165,ØØ4,Ø32,2Ø5,144
8535 :189,162,Ø24,16Ø,Ø36,Ø24,17Ø
8541 :Ø32,24Ø,255,166,ØØ5,165,188
```

```
8547 :006,032,205,189,198,174,135
8553 :032,046,017,076,053,032,105
8559 :189,068,034,024,101,179,194
8565 :133,179,076,201,032,173,143
8571 :026,143,141,129,143,032,225
8577 :243,027,169,000,141,129,070
8583 :143,174,032,143,189,127,175
8589 :036,168,202,189,127,036,131
8595 :170,202,032,247,027,096,153
8601 :169,156,032,210,255,032,239
8607 :235,017,162,003,160,030,254
8613 :024,032,240,255,032,193,173
8619 :033,174,032,143,189,127,101
8625 :036,202,056,253,127,036,119
8631 :133,167,105,003,170,160,153
8637 :030,032,240,255,162,000,140
8643 :189,207,033,208,001,096,161
8649 :032,210,255,232,208,244,102
8655 :154,068,069,077,032,032,127
8661 :032,032,028,082,069,080,024
8667 :000,032,032,077,069,078,251
8673 :072,086,084,077,065,082,179
8679 :073,067,084,078,089,078,188
8685 :074,080,065,079,072,073,168
8691 :078,073,076,077,073,087,195
8697 :073,077,078,073,065,077,180
8703 :079,078,068,083,068,078,197
8709 :069,075,083,068,069,077,190
8715 :068,068,067,086,065,087,196
8721 :086,078,067,083,067,071,213
8727 :065,070,076,075,089,084,226
8733 :078,065,076,077,083,065,217
8739 :082,076,065,079,075,084,240
8745 :088,077,084,073,068,087,006
8751 :089,067,079,078,077,065,246
8757 :090,085,084,078,086,087,051
8763 :065,079,082,067,065,065,226
8769 :075,072,073,000,004,004,037
8775 :003,013,004,008,036,016,151
8781 :025,023,012,024,020,011,192
8787 :010,008,011,003,003,005,123
8793 :007,003,010,003,012,006,130
8799 :013,008,012,021,009,011,169
8805 :009,007,006,010,008,029,170
8811 :004,004,003,008,005,007,138
8817 :005,004,010,007,047,003,189
8823 :004,000,001,003,003,004,134
8829 :005,008,009,008,010,026,191
8835 :031,011,014,016,014,012,229
8841 :037,012,019,016,017,020,002
```

```
8847 :025,009,032,025,025,029,032
8853 :027,030,026,031,029,032,068
8859 :017,034,038,035,040,040,103
8865 :041,042,038,037,046,049,158
8871 :040,040,044,000,000,000,035
8877 :239,239,047,111,231,231,247
8883 :231,231,231,210,210,210,222
8889 :207,207,207,000,000,000,038
8895 :000,000,000,000,000,193,128
8901 :193,239,047,047,111,103,169
8907 :039,039,039,039,018,018,139
8913 :018,015,079,079,077,205,170
8919 :205,000,000,000,000,000,164
8925 :193,193,240,048,048,112,031
8931 :103,039,039,039,039,019,249
8937 :019,019,015,015,079,014,138
8943 :206,205,000,000,000,199,081
8949 :007,066,193,240,048,048,079
8955 :040,040,103,041,041,041,045
8961 :019,019,019,016,016,078,168
8967 :140,205,013,205,201,199,202
8973 :007,007,004,196,241,049,005
8979 :110,046,046,045,041,041,092
8985 :041,020,020,020,084,016,226
8991 :080,012,076,011,010,073,037
8997 :009,009,071,070,197,241,122
9003 :049,110,046,046,045,045,128
9009 :042,042,042,149,149,149,110
9015 :017,145,012,076,011,138,198
9021 :074,137,088,151,200,000,199
9027 :000,049,049,110,046,045,110
9033 :045,042,042,042,021,021,030
9039 :021,085,017,145,076,075,242
9045 :031,095,090,025,087,214,115
9051 :000,000,241,049,110,172,151
9057 :044,044,043,043,171,101,031
9063 :037,037,081,099,017,096,214
9069 :096,096,089,091,091,091,151
9075 :000,000,000,000,241,049,149
9081 :113,044,044,043,043,043,195
9087 :038,037,037,165,035,035,218
9093 :098,161,160,093,156,027,060
9099 :219,000,000,000,000,241,087
9105 :049,113,044,044,043,043,225
9111 :107,038,038,038,102,035,253
9117 :163,034,033,097,029,092,093
9123 :156,000,000,000,000,000,063
9129 :000,000,000,236,236,235,108
9135 :230,038,038,038,038,038,083
9141 :036,100,034,033,033,029,190
```

```
9147 :029,000,000,000,000,000,216
9153 :242,050,242,000,243,000,202
9159 :000,000,230,230,038,038,223
9165 :230,036,228,226,225,222,092
9171 :222,030,000,000,000,000,207
9177 :000,242,050,050,000,000,047
9183 :243,000,000,000,000,230,184
9189 :230,000,000,000,000,000,203
9195 :000,000,222,222,000,000,167
9201 :000,000,242,242,242,242,185
9207 :000,000,243,000,000,000,234
9213 :000,230,000,000,000,000,227
9219 :000,000,000,222,030,222,221
9225 :000,000,242,000,000,000,251
9231 :242,000,000,000,000,000,001
9237 :000,000,000,000,000,000,021
9243 :000,000,000,000,000,222,249
9249 :222,000,000,255,032,082,112
9255 :069,071,073,079,078,083,236
9261 :000,078,069,087,032,069,124
9267 :078,071,076,000,085,082,187
9273 :066,065,078,032,078,069,189
9279 :000,072,069,065,082,084,179
9285 :076,078,068,000,071,032,138
9291 :080,076,065,073,078,083,018
9297 :000,065,084,076,065,078,193
9303 :084,073,067,000,083,079,217
9309 :085,084,072,069,082,078,051
9315 :000,065,082,075,076,065,206
9321 :084,069,088,000,077,079,246
9327 :085,078,084,065,073,078,062
9333 :000,080,065,067,073,070,216
9339 :073,067,032,000,001,007,047
9345 :010,015,022,031,035,039,025
9351 :047,052,052,220,243,243,224
9357 :047,063,220,078,228,077,086
9363 :077,228,227,206,092,062,015
9369 :243,092,227,242,227,243,147
9375 :099,063,047,228,063,069,216
9381 :100,190,069,070,100,077,003
9387 :077,070,070,212,078,212,122
9393 :243,243,197,212,228,243,007
9399 :197,235,242,228,242,047,094
9405 :033,059,033,246,104,126,022
9411 :202,189,036,097,089,189,229
9417 :220,052,118,122,081,038,064
9423 :003,171,186,238,254,204,239
9429 :171,002,080,070,070,235,073
9435 :000,145,069,001,001,134,057
```

```
9441 :Ø87,2Ø3,Ø97,Ø96,119,223,Ø26
9447 :Ø66,234,17Ø,246,245,234,146
9453 :158,124,254,111,247,Ø57,164
9459 :Ø67,159,211,Ø66,Ø27,Ø95,1ØØ
9465 :Ø29,1Ø4,164,179,ØØ5,Ø65,Ø27
9471 :Ø52,233,Ø44,Ø56,ØØ4,136,Ø12
9477 :Ø17,21Ø,Ø66,23Ø,Ø63,169,248
9483 :175,Ø77,154,Ø57,Ø61,Ø92,115
9489 :14Ø,Ø62,Ø47,12Ø,216,Ø37,127
9495 :Ø59,ØØ5,145,213,145,243,Ø65
95Ø1 :187,242,Ø11,23Ø,131,193,255
95Ø7 :ØØØ,ØØØ,Ø68,Ø82,Ø85,159,173
9513 :Ø28,152,ØØØ,ØØØ,Ø13,Ø13,247
```

4
Sound and Graphics

4

Sprite Magic: An All Machine Language Sprite Editor

Charles Brannon

Sprites make animation on the 64 fun and easy to program. But actually drawing and creating sprites with graph paper can be tedious. "Sprite Magic" simplifies their creation and lets you concentrate on the artistic aspects of sprite design. You can even animate minimovies!

What Is a Sprite Editor?

Most of what you've read about sprites covers how to program them: setting them up, protecting memory, moving and animating them, and using them in games. But sprite design is usually left up to you.

A sprite is defined by 63 binary numbers. The one bits (on) represent solid pixels. Zeros (off) represent blank areas, through which the screen background is visible. Normally, you sketch a sprite on a grid 24 squares across and 21 squares high. This is 3 bytes per row (8 bits * 3 bytes = 24 bits) and 21 rows of bytes (3 * 21 = 63 bytes). But after you've drawn the sprite, you have to convert the squares into binary, and then into decimal so that you can put the numbers in DATA statements.

There are utility programs that will do the conversion for you, even editors that let you clear and set squares with a joystick. Since you're using a computer, other functions can be supported to let you clear, invert, reflect, reverse, shift, and test out your sprite. The more work the computer does, the less you have to think in terms of binary numbers.

"Sprite Magic" offers the best features of most sprite editors, including true multicolor mode, and pulls it off with the speed and power of an all machine language program. Sprite Magic's style (and even some of the coding) is similar to "Ultrafont +," an all machine language character editor also in this book. Many of the commands are the same, so you can get up to speed quickly. If you've learned how to use Ultrafont +, it won't take much to become comfortable with Sprite Magic.

Typing It In

Since Sprite Magic is an all machine language program, you cannot enter it as you do a BASIC program. Machine language is basically a bunch of numbers: The numbers make no sense in and of themselves. Only the 6510-compatible microprocessor in your 128 can interpret and execute these numbers. Since typing in numbers is no fun, we've tried to make it as painless as possible with "MLX," the machine language editor. You'll find MLX and the explanation of its use and commands in Appendix D of this book. If you haven't already typed in MLX, do so before you try to enter Sprite Magic. Since MLX is used with other programs in this book, as well as in *COMPUTE!* magazine, *COMPUTE!'s Gazette,* and other books from COMPUTE! Publications, be sure to save it for future use.

After you've typed in MLX, run it, and answer the prompts of Starting Address and Ending Address:

Starting Address: 49152
Ending Address: 51875

You're ready to start typing in Sprite Magic. Enter each line from the program listing at the end of this article. The last number in each line is a checksum, so type it carefully. If the checksum you've typed matches the checksum computed from the line you typed, a pleasant bell tone tells you you've typed the line correctly. If the number doesn't match, a buzzer warns you to reenter the line. This way, you should be able to type in Sprite Magic correctly the first time.

Assuming you've typed and saved Sprite Magic, here's how you get it up and running. If you used the filename "SPRITE MAGIC", type

LOAD "SPRITE MAGIC",8,1 (for disk)

or

LOAD "SPRITE MAGIC",1,1 (for tape)

Be sure to add the **,1** to the end. **Type NEW and press RETURN.** This resets some important memory locations, but leaves Sprite Magic in its protected cubbyhole at $C000.

Doodle

Activate Sprite Magic with **SYS 49152.** Instantly, the main screen should appear, with a large 24 × 21 grid. The grid is a blowup of the sprite you're editing. The actual sprite will be seen to the right of the grid. The flashing square within the large grid is your cursor. Move the cursor with either the cursor keys or with a joystick plugged into port 2. To light up a blank spot (in other words, to turn that pixel on), press either the space bar or the joystick fire button. If the square is already lit, it will turn dark. This signifies that the pixel has been turned off. The button or space bar thus *toggles* points on or off. You can draw your sprite quite easily in this manner. One fine point: With the joystick, you can hold down the fire button and move the cursor. If the first point you change was set, then the fire button will continue to set points as you move the joystick, regardless of the other points' original states. If the first point you change was empty, then you can hold down the fire button and move about, clearing anything the cursor passes over. Notice how any changes are immediately visible in the actual sprite.

If you've just entered Sprite Magic, the grid is probably full of garbage pixels. To clear out the grid for a new picture, press SHIFT–CLR/HOME. You now have an empty area (a fresh canvas, so to speak) to draw upon. You can press CLR/HOME without holding down SHIFT to home the cursor to the upper-left corner of the grid.

Does the cursor move too slow or too fast? To change the velocity of the cursor, press V. Answer the prompt with a number key from 0 (slow) to 9 (very fast).

Shift, Expansion, and Symmetry

Sometimes when you're drawing, it's necessary to reposition the shape within the grid. The first two function keys let you shift the sprite shape around within the grid. If you shift something out of the grid, it wraps around to the opposite side. The f1 key shifts right; f3 shifts down. Use the SHIFT key along with the function key to move in the opposite directions: f2 moves the sprite shape left; f4, up.

After you've drawn something, press F. Instantly, the sprite is flipped upside down. Press it again to flip it back over. Remember F as the command for Flip. Now try M, for Mirror. The shape you've drawn is mirrored left to right. Of course, if you've drawn something symmetrical, you may not see any change.

Now try CONTROL-R or CONTROL-9. The sprite will become reversed. Every square that was on is now turned off, and vice versa.

A sprite can also be expanded or contracted either horizontally or vertically, or *both* horizontally and vertically. The X and Y keys on the keyboard let you do this. Press X to switch from wide to narrow, or vice versa. Press Y to switch from tall to short, or vice versa. The main grid will not change size or proportion (there's not enough room on the screen).

An unusual command is Symmetry. I added this command after some suggestions that many shapes are symmetrical from left to right, as if a mirror were put in the middle of the grid. To enter the Symmetry mode, press the back-arrow (←) key (found in the upper-left corner of the keyboard, right above the CONTROL key). Now, every square drawn on one side will be instantly mirrored to the left. Blank squares are not copied over, though, so you cannot erase in this mode. This command is not only quite useful, but it's also fun to play with. To return to normal editing, press the back-arrow key again.

Notice the number in the upper-right corner of the screen. This is the sprite page number, which can range from 0 to 255. You start out at the top of the sprite memory. The plus (+) and minus (−) keys are used to go forward or backward through the sprite shapes. Press the minus key and see how you now have a new shape in the grid.

There's a limit to how far back you can go. If you have no BASIC program in memory, you can step back to sprite page 36. However, character information resides in sprite pages below 128. You can still clear the page and draw a sprite shape on pages below 128, but it won't really register. To be safe, use only the sprite pages from 128 up. If you have a program in memory, Sprite Magic will not let you step back past its end. This protects your program from being accidentally overwritten by a sprite shape. If you want maximum space available for sprite shapes, be sure to NEW out any BASIC program before you SYS 49152. Sometimes, though, you'll want to keep a program in memory. You'll see why a bit later.

Programming note: The sprite page number, when multiplied by 64, gives you the starting memory location for the 63 numbers representing the sprite.

Put It in the Buffer

You might use Flip to design two views of a shape, such as a spaceship pointing in two directions. Draw one freehand, then do the other with Flip. Mirror can be used to design separate left and right views as well. But what you first need is a way to copy the original shape to another sprite area. One way to do this is to copy the sprite shape to an area of memory (a buffer). You can use + or − to step to another sprite page, then copy the buffer to the sprite. This is the same way you copy characters with Ultrafont +. The same keys are used in Sprite Magic. Press f7 to copy the sprite to the buffer. The grid flashes to affirm this. Then go to the sprite page where you want to put the copy and press f8 (SHIFT-f7). The shape in the buffer replaces any shape already in the sprite grid. You can also use the buffer as a fail-safe device. Before modifying an existing sprite, press f7 to save it in the buffer. Then, if you mangle the sprite or accidentally erase it, you can recall the previous shape from the buffer.

Computer Disney?

The buffer is also useful for animation. Since you can change sprite pages so easily, you can also use Sprite Magic as an animation design tool. Cartoons make only minor changes between frames. Too much change makes the animation jerky. So put the first frame into the buffer, copy it to the next area, then make a change. Put the new image into the buffer, copy it again to a new area, then make another small change. Continue in this fashion as you build up a whole series of frames. Put different but similar shapes on adjacent pages, then hold down + or − to step through the shapes. As with cartoon animation, you'll get the illusion of motion. Use a cursor velocity of 9 for maximum speed. Even if you don't care to program sprites, Sprite Magic is a fun tool for making moving cartoons.

A Bit of Color

The normal drawing mode lets you set or clear points, but in only one color. If you're willing to give up half as many horizontal points, you can have four colors to work with. Multicolor mode lets any square be one of four colors, but gives you only 12 pixels across instead of 24. This is because two dots are grouped together to give four combinations. The colors come from four memory locations:

Pattern	Color Location	
00	53281	Background color register
01	53285	Sprite multicolor register 0
10	53287–53294	Sprite color registers
11	53286	Sprite multicolor register 1

There are two multicolor sprite registers, which are shared among all sprites (in programming, but not in Sprite Magic, you can have eight sprites on the screen at the same time). The bit pattern marked 10 is unique to each sprite and comes from that sprite's own color register. Pattern 00 is blank, and whatever is underneath the sprite shape will show through.

The reason for this sojourn into bits and addresses is that only the bit pattern marked 10 has a unique color for that sprite. If you're designing several sprites for a game, remember that anything drawn in that color can be changed individually for each sprite. Squares drawn with bit pattern 01 or 11 will be colored from two locations shared by all sprites.

Many sprite editors let you see how the sprite would look in multicolor, but you still have to pair up the pixels yourself and keep track of binary bit pairs. No fun! Instead, Sprite Magic offers a multicolor mode. When you press f5, the screen instantly changes. Each square in the grid is now rectangular, two squares wide. The cursor has also been enlarged and can be moved about as before in the new grid. But the way you set and clear points has been changed, since you're now working with four colors.

Multicolor Palette

The fire button or the space bar always sets a point, but you have to tell Sprite Magic which color you are currently drawing in. The number keys 1 to 4 select the drawing color. The number you press is one number higher than the binary value of the bit pairs in the table above. The 1 key, for instance, chooses the 00 bit pair, which represents the background color. In practice, you're choosing from a palette of four colors. The 1 key can be used when you want to erase, although the fire button can still be used to toggle points on and off.

When you press a number key from 1 to 4, the small colored block beside the sprite number changes to remind you which color you're drawing with. If you want to change one of the four colors, hold down SHIFT while you type the number. The prompt ENTER COLOR KEY appears. Now you have to enter another key combination. Press CONTROL and one of the number keys from 1 to 8, or hold down the Commodore key and one of the number keys from 1 to 8. These are the same key combinations you use to change the text color in BASIC. You can also change the screen background color by pressing B on the keyboard until the color you want appears.

Some Sprite Magic commands act strangely in multicolor mode. For example, a shift left or shift right (done with the f1 and f2 keys respectively) moves the sprite over by only one bit, which changes the color assignments. In general, you must press f1 or f2 twice to preserve the same colors. Pressing the M key (for Mirror) reverses the bit pairs so that every 01 becomes a 10. The effect is that colors 2 and 3 are exchanged. The CONTROL-R and CONTROL-9 key combinations (Reverse) also invert the bits so that 01 becomes 10, 10 becomes 01, 00 becomes 11, and 11 becomes 00. Colors 2 and 3 are switched as well as colors 1 and 4. Flip, however, works identically in multicolor and normal (nonmulticolor) modes.

If you want to go back to normal mode, press the f6 key (SHIFT-f5). There's nothing to prevent you from designing both normal and multicolor sprites on different pages.

If you changed colors in the multicolor mode, some of the colors in the normal mode may have been changed. You can alter these colors as in multicolor mode. Press SHIFT-1 to change the color of the empty pixels, and SHIFT-2 to change the color of the *on* pixels. (You'll be prompted to press a color number key after each SHIFT-1 or SHIFT-2 combination. Remember to press either CONTROL or the Commodore key simultaneously with the color key.)

Mobilizing Your Sprite

If you want to try out your sprite in action, press J (for Joystick). You can now move the actual sprite around with the joystick. The speed of movement depends on the current cursor velocity. When you've finished putting your sprite through its paces, press the fire button to return to Sprite Magic. Also, if you want to test the animation while you are moving about, hold down the SHIFT key to step forward through the pages of your defined sprites or the Commodore key to step backward. You can lock the SHIFT key to keep the animation happening while you move around.

Saving Your Sprites

After all your work, you surely want to save your creations on tape or disk for future use. You can save an individual shape or all the sprites. Press S (for Save), then either D (Disk) or T (Tape). Next, enter the filename. You'll be asked if you want to "Save all from here?" If you press N (No), then only the sprite you're currently working on will be saved. If you press Y (Yes), then every sprite from the current sprite to sprite 255 will be saved. Thus, if you want to save a range of sprites, be sure to use the minus key to step back to the first sprite you want saved.

If you use a filename already present on the disk, Sprite Magic first scratches the old file, then saves the new file using the same name. This insures that your disk will not be damaged in any way. However, make sure you want only the *newest* version of the sprite information if you use the same filename. If you're not sure, simply call it something else.

To recall your sprites, press L. The Load command loads everything that was saved. If you're loading in more than one sprite, be sure you step backward far enough with the minus key so that all the sprites will fit between the current sprite and sprite 255. The sprites load starting at the current sprite page number. After you press L, enter T or D for tape or disk.

Let There Be DATA

If you're a programmer, you're probably more interested in DATA statements. That way, you can use BASIC to READ and POKE the numbers into memory. If you have some kind of DATA maker, you can run it on the memory used by the sprite in Sprite Magic (again, the memory location is the sprite number times 64). But Sprite Magic has a special DATA maker of its own. It's similar to the Create DATA option in Ultrafont +, but it's been enhanced.

Press CONTROL-D to create a series of DATA statements from the current sprite in memory. Just tap the key, or you'll get hundreds of DATA statements as the key repeats. Sprite Magic will create eight DATA statements, with eight bytes per line. The last byte is not strictly used. Sprite shapes are made from 63 bytes, but the sprite areas are padded so that they'll conveniently fall in 64-byte ranges. To create DATA statements for another sprite, use the + or − key to move to the correct sprite page; then press CONTROL-D again.

If you have a program already in memory, the DATA statements are appended to the end of the program, starting with the next available line number. To add DATA statements to an existing program, then, first load Sprite Magic. Type NEW, load your BASIC program, and SYS 49152 to enter Sprite Magic. You can then load in sprite shapes and use CONTROL-D to add those DATA statements to the end of the BASIC program in memory.

You can check to see that these DATA statements were added by exiting Sprite Magic (press CONTROL-X) and typing LIST. Your program should have eight new DATA lines for each sprite pattern. If there was no program in memory, the DATA statements form a program all their own, starting with line 1. If you want, you can save just the DATA statements to tape or disk, using the normal SAVE command.

To exit Sprite Magic and return to BASIC, press CONTROL-X. You can also use RUN/STOP–RESTORE.

Quick-Reference Chart

B	Cycles through background colors
F	Flips sprite upside-down
J	Moves sprite with joystick; press button when done
L	Loads sprite(s) from tape or disk
M	Mirrors sprite from left to right
S	Saves sprite(s) to tape or disk
V	Sets cursor velocity
X	Toggles X expansion on/off
Y	Toggles Y expansion on/off

CONTROL-D	Creates DATA statements
CONTROL-R or **CONTROL-9**	Reverses sprite
CONTROL-X	Exits to BASIC

+	Next sprite page
−	Previous sprite page
CLR/HOME	Homes sprite editing cursor
SHIFT-CLR/HOME	Erases grid
Space bar or **fire button**	Sets/clears points
CRSR keys or **joystick in port 2**	Moves cursor
Back arrow	Symmetry mode
Keys 1–4	Select drawing color for multicolor mode
SHIFT 1–4	Change a drawing color
CONTROL 1–8 or **Commodore 1–8**	Chooses new color

f1	Shifts right
f2	Shifts left
f3	Shifts down
f4	Shifts up
f5	Multicolor mode
f6	Normal mode
f7	Stores sprite to buffer
f8	Recalls sprite from buffer

Sprite Magic

For mistake-proof program entry, be sure to use "MLX" (Appendix D).

```
49152 :076,050,195,000,001,003,069
49158 :004,032,198,192,169,004,093
49164 :133,252,169,000,133,251,182
49170 :133,167,169,216,133,168,236
49176 :169,021,141,040,002,169,054
49182 :003,141,041,002,160,000,121
49188 :177,253,170,173,048,002,091
49194 :240,003,076,152,192,169,106
49200 :207,145,251,138,010,170,201
49206 :176,008,173,003,192,145,239
49212 :167,076,069,192,173,004,229
49218 :192,145,167,200,192,008,202
49224 :208,221,024,165,251,105,022
49230 :008,133,251,133,167,165,167
49236 :252,105,000,133,252,105,163
49242 :212,133,168,230,253,208,014
49248 :002,230,254,206,041,002,063
49254 :173,041,002,208,183,024,221
49260 :165,251,105,016,133,251,005
49266 :133,167,165,252,105,000,168
49272 :133,252,105,212,133,168,099
49278 :206,040,002,173,040,002,077
49284 :240,003,076,029,192,169,073
49290 :160,141,026,004,174,051,182
49296 :002,189,003,192,141,026,185
49302 :216,096,134,097,169,000,094
49308 :141,042,002,006,097,046,234
49314 :042,002,006,097,046,042,141
49320 :002,174,042,002,169,207,252
49326 :145,251,200,169,247,145,051
49332 :251,136,189,003,192,145,072
49338 :167,200,145,167,200,192,233
49344 :008,208,215,076,074,192,197
49350 :169,000,133,254,173,043,202
49356 :002,133,253,006,253,038,121
49362 :254,006,253,038,254,006,253
49368 :253,038,254,006,253,038,034
49374 :254,006,253,038,254,006,009
49380 :253,038,254,096,032,198,075
49386 :192,160,000,177,253,073,065
49392 :255,145,253,200,192,064,069
49398 :208,245,096,032,198,192,193
49404 :160,062,136,136,177,253,152
49410 :010,008,200,200,162,003,073
49416 :177,253,040,042,008,145,161
49422 :253,136,202,208,245,040,074
49428 :192,255,208,230,096,032,009
```

```
49434 :198,192,160,000,200,200,208
49440 :177,253,074,008,136,136,048
49446 :162,003,177,253,040,106,011
49452 :008,145,253,200,202,208,036
49458 :245,040,192,063,208,230,004
49464 :096,032,198,192,160,000,222
49470 :177,253,153,227,202,200,250
49476 :192,003,208,246,177,253,123
49482 :136,136,136,145,253,200,056
49488 :200,200,200,192,063,208,119
49494 :241,162,000,160,060,189,130
49500 :227,202,145,253,200,232,071
49506 :224,003,208,245,096,032,138
49512 :198,192,160,060,162,000,108
49518 :177,253,157,227,202,200,046
49524 :232,224,003,208,245,160,164
49530 :060,177,253,200,200,200,188
49536 :145,253,136,136,136,136,046
49542 :016,243,160,000,185,227,197
49548 :202,145,253,200,192,003,111
49554 :208,246,096,032,198,192,094
49560 :160,000,152,170,232,232,074
49566 :169,003,133,097,169,008,225
49572 :141,055,002,177,253,074,098
49578 :145,253,062,227,202,206,241
49584 :055,002,173,055,002,208,159
49590 :240,200,202,198,097,165,004
49596 :097,208,227,192,063,144,095
49602 :215,160,000,185,227,202,159
49608 :145,253,200,192,063,208,237
49614 :246,096,169,147,032,210,082
49620 :255,173,000,220,133,097,066
49626 :041,015,073,015,170,173,193
49632 :000,208,024,125,080,194,087
49638 :141,000,208,173,016,208,208
49644 :125,091,194,141,016,208,243
49650 :173,001,208,024,125,102,107
49656 :194,141,001,208,032,036,092
49662 :195,173,141,002,041,001,039
49668 :024,109,248,007,141,248,013
49674 :007,173,141,002,041,002,120
49680 :074,073,255,056,109,248,063
49686 :007,141,248,007,165,097,175
49692 :041,016,208,181,173,000,135
49698 :220,041,016,240,249,173,205
49704 :043,002,141,248,007,032,001
49710 :082,196,169,255,141,000,121
49716 :208,169,000,141,016,208,026
49722 :169,128,141,001,208,076,013
49728 :195,194,032,198,192,160,011
```

```
49734 :000,152,145,253,200,192,244
49740 :063,208,249,096,000,000,180
49746 :000,000,255,255,255,000,079
49752 :001,001,001,000,000,000,091
49758 :000,255,255,255,000,000,091
49764 :000,000,000,255,001,000,100
49770 :000,255,001,000,000,255,105
49776 :001,018,083,080,082,073,193
49782 :084,069,032,077,065,071,004
49788 :073,067,032,050,046,048,184
49794 :146,095,069,082,082,079,171
49800 :082,032,079,078,032,083,010
49806 :065,086,069,047,076,079,052
49812 :065,068,095,018,084,146,112
49818 :065,080,069,032,079,082,049
49824 :032,018,068,146,073,083,068
49830 :075,063,095,070,073,076,106
49836 :069,078,065,077,069,058,076
49842 :095,080,082,069,083,083,158
49848 :032,067,079,076,079,082,087
49854 :032,075,069,089,095,169,207
49860 :113,160,194,133,251,132,155
49866 :252,160,040,169,032,153,240
49872 :191,007,136,208,250,177,153
49878 :251,200,201,095,208,249,138
49884 :136,132,097,152,074,073,116
49890 :255,056,105,020,168,162,224
49896 :024,024,032,240,255,169,208
49902 :146,032,210,255,160,000,017
49908 :177,251,032,210,255,200,089
49914 :196,097,144,246,096,133,138
49920 :251,132,252,160,040,169,236
49926 :032,153,191,007,136,208,221
49932 :250,162,024,160,000,024,120
49938 :032,240,255,160,000,177,114
49944 :251,201,095,240,006,032,081
49950 :210,255,200,208,244,096,219
49956 :174,053,002,240,008,160,161
49962 :000,200,208,253,202,208,089
49968 :250,096,169,147,032,210,184
49974 :255,169,000,141,134,002,243
49980 :141,056,002,169,008,032,212
49986 :210,255,169,128,141,138,083
49992 :002,169,048,141,053,002,231
49998 :169,255,141,043,002,169,089
50004 :000,141,048,002,173,006,198
50010 :192,141,038,208,173,004,078
50016 :192,141,037,208,141,039,086
50022 :208,169,001,141,051,002,162
50028 :032,007,192,169,255,141,136
```

```
50034 :000,208,169,128,141,001,249
50040 :208,173,043,002,141,248,167
50046 :007,169,001,141,021,208,161
50052 :169,000,141,028,208,169,079
50058 :012,141,033,208,141,032,193
50064 :208,141,044,002,141,045,213
50070 :002,032,195,194,032,082,175
50076 :196,032,007,192,032,053,156
50082 :196,173,000,220,072,041,096
50088 :015,073,015,141,046,002,204
50094 :104,041,016,141,047,002,013
50100 :032,228,255,240,006,032,205
50106 :005,197,076,157,195,032,080
50112 :036,195,173,047,002,208,085
50118 :003,032,112,196,032,053,114
50124 :196,173,047,002,073,016,199
50130 :141,052,002,173,046,002,114
50136 :240,195,174,046,002,189,038
50142 :080,194,172,048,002,240,190
50148 :001,010,024,109,044,002,162
50154 :141,044,002,024,173,045,151
50160 :002,125,102,194,141,045,081
50166 :002,174,044,002,016,017,245
50172 :162,000,142,044,002,162,252
50178 :023,173,048,002,240,002,234
50184 :162,022,142,044,002,174,042
50190 :044,002,224,024,144,005,201
50196 :162,000,142,044,002,172,030
50202 :045,002,016,005,160,020,018
50208 :140,045,002,172,045,002,182
50214 :192,021,144,005,160,000,048
50220 :140,045,002,032,053,196,000
50226 :076,157,195,174,045,002,187
50232 :172,044,002,032,240,255,033
50238 :164,211,173,048,002,208,100
50244 :005,169,032,145,209,096,212
50250 :169,032,145,209,200,145,206
50256 :209,096,162,000,160,030,225
50262 :024,032,240,255,169,018,056
50268 :032,210,255,174,043,002,040
50274 :142,248,007,169,000,032,184
50280 :205,189,169,032,032,210,173
50286 :255,096,032,198,192,173,032
50292 :045,002,010,109,045,002,073
50298 :133,097,173,044,002,074,133
50304 :074,074,024,101,097,168,154
50310 :173,044,002,041,007,073,218
50316 :007,170,232,134,097,056,068
50322 :169,000,042,202,208,252,251
50328 :174,048,002,208,047,133,252
```

```
50334 :097,173,052,002,208,016,194
50340 :169,000,141,049,002,177,190
50346 :253,037,097,208,005,169,171
50352 :001,141,049,002,165,097,119
50358 :073,255,049,253,174,049,011
50364 :002,240,002,005,097,145,167
50370 :253,173,056,002,240,003,153
50376 :032,054,202,096,133,098,047
50382 :074,005,098,133,098,174,020
50388 :052,002,208,014,162,000,138
50394 :142,049,002,049,253,208,153
50400 :005,169,001,141,049,002,079
50406 :165,098,073,255,049,253,099
50412 :166,097,202,133,097,173,080
50418 :051,002,074,042,202,208,053
50424 :252,174,049,002,208,002,167
50430 :169,000,005,097,145,253,155
50436 :096,141,050,002,174,033,244
50442 :197,221,033,197,240,004,134
50448 :202,208,248,096,202,138,086
50454 :010,170,189,074,197,072,222
50460 :189,073,197,072,096,039,182
50466 :133,137,134,138,077,074,215
50472 :147,018,145,017,157,029,041
50478 :135,139,049,050,051,052,010
50484 :019,136,140,033,034,035,193
50490 :036,086,083,076,024,088,195
50496 :089,066,032,160,043,045,243
50502 :004,095,070,024,193,248,192
50508 :192,102,193,056,193,148,192
50514 :193,207,193,065,194,231,141
50520 :192,150,197,160,197,166,126
50526 :197,180,197,214,197,003,058
50532 :198,028,198,028,198,028,010
50538 :198,028,198,050,198,061,071
50544 :198,089,198,121,198,121,013
50550 :198,121,198,121,198,195,125
50556 :198,052,200,219,200,242,211
50562 :200,196,197,205,197,156,001
50568 :197,111,196,111,196,253,176
50574 :198,011,199,089,201,105,177
50580 :202,114,202,206,045,002,151
50586 :076,192,197,238,033,208,074
50592 :096,238,045,002,076,192,041
50598 :197,206,044,002,173,048,068
50604 :002,240,017,206,044,002,171
50610 :076,192,197,238,044,002,159
50616 :173,048,002,240,003,238,120
50622 :044,002,104,104,076,247,255
50628 :195,173,029,208,073,001,107
```

```
50634 :141,029,208,096,173,023,104
50640 :208,073,001,141,023,208,094
50646 :096,169,016,141,048,002,174
50652 :169,001,141,028,208,141,140
50658 :051,002,032,007,192,173,171
50664 :004,192,141,037,208,173,219
50670 :005,192,141,039,208,173,228
50676 :006,192,141,038,208,173,234
50682 :044,002,041,254,141,044,008
50688 :002,076,192,197,169,000,124
50694 :141,048,002,141,032,208,066
50700 :141,028,208,169,001,141,188
50706 :051,002,173,004,192,141,069
50712 :039,208,076,007,192,173,207
50718 :048,002,208,001,096,056,185
50724 :173,050,002,233,049,141,172
50730 :051,002,170,189,003,192,137
50736 :076,007,192,169,000,141,121
50742 :044,002,141,045,002,076,108
50748 :192,197,032,232,192,032,169
50754 :007,192,032,232,192,032,241
50760 :007,192,032,198,192,160,085
50766 :000,177,253,153,163,202,002
50772 :200,192,064,208,246,096,066
50778 :032,198,192,160,000,185,089
50784 :163,202,145,253,200,192,227
50790 :064,208,246,096,144,005,097
50796 :028,159,156,030,031,158,158
50802 :129,149,150,151,152,153,230
50808 :154,155,169,179,160,194,107
50814 :032,199,194,032,157,202,174
50820 :162,000,221,106,198,240,035
50826 :008,232,224,016,208,246,048
50832 :076,195,194,056,173,050,120
50838 :002,233,033,168,138,153,109
50844 :003,192,173,048,002,208,014
50850 :009,173,004,192,141,039,208
50856 :208,076,190,198,173,004,249
50862 :192,141,037,208,173,005,162
50868 :192,141,039,208,173,006,171
50874 :192,141,038,208,032,195,224
50880 :194,076,007,192,169,231,037
50886 :160,198,032,199,194,032,245
50892 :228,255,056,233,048,048,048
50898 :248,201,010,176,244,133,198
50904 :097,056,169,009,229,097,105
50910 :010,010,010,141,053,002,192
50916 :076,195,194,067,085,082,159
50922 :083,079,082,032,086,069,153
50928 :076,079,067,073,084,089,196
```

```
50934 :032,040,048,045,057,041,253
50940 :063,095,173,043,002,201,061
50946 :255,240,006,238,043,002,018
50952 :032,082,196,096,206,043,151
50958 :002,032,198,192,165,046,137
50964 :197,254,144,004,238,043,132
50970 :002,096,032,082,196,096,018
50976 :160,000,140,055,002,169,046
50982 :164,032,210,255,169,157,001
50988 :032,210,255,032,157,202,164
50994 :172,055,002,133,097,169,166
51000 :032,032,210,255,169,157,143
51006 :032,210,255,165,097,201,254
51012 :013,240,043,201,020,208,025
51018 :013,192,000,240,211,136,098
51024 :169,157,032,210,255,076,211
51030 :034,199,041,127,201,032,208
51036 :144,196,192,020,240,192,052
51042 :165,097,153,000,002,032,035
51048 :210,255,169,000,133,212,059
51054 :200,076,034,199,169,095,115
51060 :153,000,002,152,096,032,039
51066 :231,255,169,151,160,194,002
51072 :032,199,194,032,157,202,176
51078 :162,001,201,084,240,011,065
51084 :162,008,201,068,240,005,056
51090 :104,104,076,195,194,141,192
51096 :054,002,160,000,169,001,026
51102 :032,186,255,169,169,160,105
51108 :194,032,255,194,032,032,135
51114 :199,208,007,173,054,002,045
51120 :201,084,208,237,173,054,109
51126 :002,201,068,208,069,169,131
51132 :083,141,020,002,169,048,139
51138 :141,021,002,169,058,141,214
51144 :022,002,160,000,185,000,057
51150 :002,153,023,002,200,204,022
51156 :055,002,208,244,200,200,097
51162 :200,173,050,002,201,083,159
51168 :208,026,152,072,160,002,076
51174 :162,020,032,189,255,169,033
51180 :015,162,008,160,015,032,116
51186 :186,255,032,192,255,032,170
51192 :231,255,104,168,136,076,194
51198 :014,200,160,000,185,000,045
51204 :002,153,021,002,200,204,074
51210 :055,002,208,244,152,162,065
51216 :021,160,002,032,189,255,163
51222 :169,160,133,178,096,083,073
51228 :065,086,069,032,065,076,165
```

```
51234 :076,032,070,082,079,077,194
51240 :032,072,069,082,069,063,171
51246 :032,040,089,047,078,041,117
51252 :095,032,121,199,032,198,217
51258 :192,169,027,160,200,032,070
51264 :199,194,032,157,202,201,025
51270 :089,208,007,162,000,160,184
51276 :064,076,091,200,024,165,184
51282 :253,105,064,170,165,254,069
51288 :105,000,168,165,253,133,144
51294 :251,165,254,133,252,032,157
51300 :249,200,169,251,032,216,193
51306 :255,176,011,032,183,255,250
51312 :208,006,032,003,201,076,126
51318 :195,194,032,003,201,032,007
51324 :231,255,173,054,002,201,016
51330 :068,240,013,169,132,160,144
51336 :194,032,199,194,032,157,176
51342 :202,076,195,194,169,000,210
51348 :032,189,255,169,015,162,202
51354 :008,160,015,032,186,255,042
51360 :032,192,255,162,015,032,080
51366 :198,255,160,000,032,207,250
51372 :255,201,013,240,007,153,017
51378 :000,002,200,076,170,200,058
51384 :169,095,153,000,002,032,123
51390 :204,255,169,000,160,002,212
51396 :032,199,194,162,015,032,062
51402 :201,255,169,073,032,210,118
51408 :255,169,013,032,210,255,118
51414 :032,231,255,076,140,200,124
51420 :032,121,199,032,249,200,029
51426 :032,198,192,169,000,166,215
51432 :253,164,254,032,213,255,123
51438 :176,136,076,003,201,169,231
51444 :004,141,136,002,000,169,184
51450 :000,141,021,208,169,147,168
51456 :076,210,255,169,001,141,084
51462 :021,208,169,147,032,210,025
51468 :255,032,082,196,032,007,104
51474 :192,076,195,194,248,169,068
51480 :000,141,000,001,141,001,052
51486 :001,224,000,240,021,202,206
51492 :024,173,000,001,105,001,084
51498 :141,000,001,173,001,001,103
51504 :105,000,141,001,001,076,116
51510 :031,201,216,173,001,001,165
51516 :009,048,141,002,001,173,178
51522 :000,001,041,240,074,074,240
51528 :074,074,009,048,141,001,163
```

```
51534 :001,173,000,001,041,015,053
51540 :009,048,141,000,001,096,123
51546 :056,165,045,233,002,133,212
51552 :045,165,046,233,000,133,206
51558 :046,169,001,133,097,169,205
51564 :008,133,098,169,000,133,137
51570 :057,133,058,160,000,177,187
51576 :097,200,017,097,240,027,030
51582 :160,002,177,097,133,057,240
51588 :200,177,097,133,058,160,189
51594 :000,177,097,072,200,177,093
51600 :097,133,098,104,133,097,038
51606 :076,117,201,024,165,057,022
51612 :105,001,133,057,165,058,163
51618 :105,000,133,058,032,198.176
51624 :192,160,000,132,098,160,142
51630 :000,024,165,045,105,037,038
51636 :145,045,200,165,046,105.118
51642 :000,145,045,200,165,057,030
51648 :145,045,200,165,058,145,182
51654 :045,200,169,131,145,045,165
51660 :200,132,097,164,098,132,003
51666 :098,177,253,170,032,022,194
51672 :201,164,097,173,002,001,086
51678 :145,045,173,001,001,200,019
51684 :145,045,173,000,001,200,024
51690 :145,045,200,169,044,145,214
51696 :045,200,132,097,164,098,208
51702 :200,152,041,007,208,213,043
51708 :132,098,164,097,136,169,024
51714 :000,145,045,160,000,177,017
51720 :045,072,200,177,045,133,168
51726 :046,104,133,045,230,057,117
51732 :208,002,230,058,164,098,012
51738 :192,064,208,143,160,000,025
51744 :152,145,045,200,145,045,252
51750 :024,165,045,105,002,133,000
51756 :045,165,046,105,000,133,026
51762 :046,076,094,166,032,149,101
51768 :193,173,045,002,010,109,076
51774 :045,002,168,162,000,185,112
51780 :227,202,157,035,203,200,068
51786 :232,224,003,208,244,032,249
51792 :149,193,173,045,002,010,140
51798 :109,045,002,168,162,000,060
51804 :177,253,029,035,203,145,166
51810 :253,200,232,224,003,208,194
51816 :243,096,173,056,002,073,235
51822 :001,141,056,002,096,032,182
51828 :198,192,160,000,162,060,120
```

```
51834 :169,003,133,097,177,253,186
51840 :157,227,202,200,232,198,064
51846 :097,165,097,208,243,138,058
51852 :056,233,006,170,016,232,085
51858 :160,062,185,227,202,145,103
51864 :253,136,016,248,096,032,165
51870 :228,255,240,251,096,013,217
```

Ultrafont +

Charles Brannon

This fast, feature-packed, machine language utility makes custom characters a breeze. Its unique features let you concentrate on your artwork instead of programming.

Anyone who has used graph paper to plot out characters, then tediously converted the rows into decimal numbers can appreciate a character editor. Instead of drawing and erasing on paper, you can draw your characters freehand with a joystick. "Ultrafont +" has been written to offer almost every conceivable aid to help you design whole character sets.

Typing It In

Ultrafont + is written entirely in machine language, giving you speed and efficiency that BASIC can't match. While this gives you a product of commercial quality, it does carry the liability of lots of typing. The program is actually rather short, using less than 4K of memory at hexadecimal location $C000 (49152), which is reserved for programs like this one. Therefore, you don't lose one byte of BASIC programming space.

However, 4000 characters require three times as much typing, since each byte must be represented by a three-digit number (000–255). With that much typing, mistakes are inevitable. To make things manageable, we've prepared Ultrafont + to be typed in using "MLX," the machine language editor. Full instructions are provided in Appendix D. So, despite the typing, rest assured that a few afternoons at the keyboard will yield a substantial reward.

Once you've entered, saved, and run MLX, answer the two questions, starting address and ending address:

Starting Address: 49152
Ending Address: 52505

After you've saved the program with MLX, you can load it with LOAD *"filename"*,1,1 for tape, or LOAD *"filename"*,8,1 for disk. **After it's loaded, enter NEW, then SYS 49152.**

The Display

At the bottom of the screen are eight lines of characters. These are the 256 characters you can customize, arranged in eight

32-character rows. A flashing square rests on the *at* symbol (@), the home position of the character set. Above the eight rows is the main grid, a blown-up view of ten characters. The bottom row of the screen is reserved for messages. The first time you SYS to Ultrafont +, you'll be asked whether you want to edit the uppercase/graphics character set, or the lowercase set.

About the Grid

The grid is like a large window on the character set. You see the first five characters and the five beneath them. A large red cursor shows you which character you're currently editing, and a smaller flashing square is the cursor you use to set and clear pixels in order to draw a character.

Moving Around

You can use the cursor keys (up, down, left, right) to move the large red cursor to any character you want to edit. If you move to a character not on the large grid (out of the window), the window automatically scrolls to make the character appear. You can also look at the bottom of the screen to move the larger cursor, since the flashing square on the character set moves with the main grid.

The HOME key moves the small cursor to the upper-left corner of the screen. If you press it twice, it takes you back to the top of the character set—to @.

A joystick plugged into port 2 moves the small cursor within the grid. If you move the cursor out of the current character, the red cursor jumps to the next character in whatever direction you want to move. The display at the bottom adjusts, and the grid scrolls as necessary. This means that you can ignore the traditional boundaries between characters and draw shapes as big as the entire character set (256 × 64 pixels—a pixel is a picture element, or dot). You still edit one character at a time or make a shape within a 2 × 2 box of characters. There is no wraparound for the cursor in the bottom section of the screen. When it hits an edge, it will go no further in that direction.

The joystick's fire button is used to set and clear points. If you press it when the cursor is resting on a solid square, the pixel is turned off. If the square is currently off, it's turned on. Holding down the button while you move the joystick keeps you in the same drawing mode. If you set a point, you will continue to draw as you move. If you clear a point, you can move around and erase points all over the screen.

If the drawing cursor is too fast or too slow, just press V to set the cursor velocity. Answer the prompt with a speed from 0 (slow) to 9 (too fast for practical use).

Manipulations

There are several functions that affect the current character (where the red box is). You can rotate, shift, mirror, reverse, erase, replace, and copy characters. The best way to learn is to play with the functions. It's really a lot of fun. The following keys control each function.

Function Keys

f1	Scrolls character right. All pixels move right. The rightmost column of pixels wraps around to the left.
f2	Scrolls character left. Wraparound is like f1.
f3	Scrolls character down. All pixels move down. The last row of pixels wraps around to the top.
f4	Scrolls character up. Wraparound is like f3.
R	Rotate. Rotates the character 90 degress. Press twice to flip the character upside down.
M	Mirror. Creates a mirror image of the character left to right.
CLR (SHIFT-CLR/HOME)	Erases the current character.
CONTROL-R or **CONTROL-9**	Reverses the character. All set dots are clear, and all empty dots are set. The bottom half of the character set is the reversed image of the top half.
CONTROL-back arrow (←)	Copies upper half of the character set, reverses it, and places it in the lower half. This way, you have to redraw only the normal characters, then use CONTROL–back arrow to create the reverse set.
F	Fix. Use this if you want to restore the normal pattern for the character. If you've redefined A and press F while the red cursor is on the character, the Commodore pattern for A will be copied back from ROM.
T	Type. This lets you try out your character set. The screen clears, with a copy of the character set provided for reference. You can type and

move the cursor around, just as in BASIC. This is handy for envisioning sample screens and fitting together multiple-character shapes. Press the RUN/STOP key to exit from Type and return to Ultrafont +.

Saving and Loading Character Sets

To save your creation to tape or disk, press S, then either T for tape or D for disk. When requested, enter the filename, up to 16 characters. Don't use the 0: prefix if you're using a disk drive (it's added for you). The screen clears, displays the appropriate messages, and then returns to the editing screen if there are no errors. If there *are* errors, such as the disk being full, Ultrafont + will read the disk error message and display it at the bottom of the screen,

Press a key after you've read the message and try to correct the cause of the error before you save again. The computer cannot detect an error during a tape SAVE.

To load a character set previously saved, press L and answer the TAPE OR DISK message. Enter the filename. If you're using tape, be sure the tape is rewound and ready. After the LOAD, you'll be returned to the editing screen; a glance is all it takes to see that the set is loaded. If an error is detected on a tape LOAD, you'll see the message ERROR ON SAVE/LOAD. Once again, if you are using disk, the error message will be displayed. Press a key to return to editing so that you can try again.

Copying and Moving Characters

You can copy one character to another with function keys 7 and 8. When you press f7, the current character flashes briefly, then is copied into a buffer. Ultrafont + remembers that character pattern. You can position the cursor where you want to copy the character before pressing f8. The memorized character replaces the character the cursor is resting on. You can also use the buffer as a fail-safe device. Before you begin to edit a character you've already worked on, press f7 to store it safely away. That way, if you accidentally wipe it out or otherwise garble the character, you can press f8 to bring back your earlier version.

Creating DATA Statements

A very useful command, CONTROL-D, allows you to create DATA statements for whatever characters you've defined. Ultrafont + doesn't make DATA statements for all the characters, just the ones you've changed. After you press CONTROL-D, Ultrafont + adds the DATA statements to the end of whatever program you have in BASIC memory. If there is no program, the DATA statements exist alone.

You can load Ultrafont +, enter NEW to reset some BASIC pointers, load a program you're working on, then SYS 49152 to Ultrafont + to add DATA to the end of the program. The DATA statements always start at line 63000, so you may want to renumber them. If you press CONTROL-D twice, another set of DATA statements will be appended, also numbered from line numbers 63000 and up. Since the keys repeat if held down, just tap CONTROL-D. If you hold it down, you may find a hundred DATA statements have been created! See the notes at the end of this article for more details on using DATA statements in your own programs.

Exiting Ultrafont +

After you create the DATA, you'll still be in Ultrafont +. If you want to exit to see the DATA statements or go on to other things, press CONTROL-X. The screen will reset to the normal colors and you'll see the READY. prompt. If you've made DATA, a LIST dramatically reveals it. It's best to enter the command CLR to make sure BASIC is initialized properly after creating DATA statements. One thing to watch out for: Don't use RUN/STOP–RESTORE to exit Ultrafont +. The program moves screen memory from the default area at address 1024, and the RUN/STOP–RESTORE combination does not reset the operating system pointers to screen memory. If you do press it, you won't be able to see what you're typing. To fix it, blindly type POKE 648,4 or SYS 49152 to reenter Ultrafont + so you can exit properly.

Reentering Ultrafont +

To restart Ultrafont + within the program, press SHIFT–RUN/STOP. After you've exited to BASIC, you can rerun Ultrafont + with SYS 49152. You'll see the character set you were working on previously, along with the message USE

ROM SET? (Y/N). Usually, Ultrafont + will copy the ROM character patterns into RAM where you can change them. If you press N, however, the set you were previously working on is untouched. Press any other key, like RETURN, to reset the characters to the ROM standard. You can copy either the uppercase/graphics set from ROM, or the lowercase set.

A Whole New World of Multicolor

You're not finished yet. There's yet another mode of operation within Ultrafont +, the multicolor mode. In multicolor mode, any character can contain up to four colors (one has to be used for the background) simultaneously. Multicolor changes the way the computer interprets character patterns. Instead of a one bit representing a solid pixel and a zero representing a blank, the eight bits are organized as four *pairs* of bits. Each pair can represent four possibilities: 00, 01, 10, and 11. Each pair is also a number in decimal from 0 to 3, and represents one of the four colors.

Ultrafont + makes multicolor easy. You don't have to keep track of bit pairs any more than you have to convert binary to decimal. Just press the f5 key. Presto—the whole screen changes. The normal characters are rather unrecognizable, and the drawing cursor is twice as wide (since eight bits have been reduced to four pixel-pairs, making each dot twice as wide). You have only four dots horizontally per character, but you can easily combine several characters to form larger shapes.

Multicolor redefines the way the joystick and fire button work. The fire button always lays down a colored rectangle in the color you're currently working with. That color is shown in the center of the drawing cursor. Press the number keys 1, 2, 3, or 4 to choose different colors to draw with. The number of the key is one more than the bit pattern, so color 1 is bit pattern 00, and color 4 is bit pattern 11. When you first SYS to Ultrafont +, the four colors show up distinctly on a color TV or monitor.

You can easily change the colors. Just hold down SHIFT and press the appropriate number key to change that number's color. You will see the message PRESS COLOR KEY. Now press one of the color keys from CONTROL-1 to CONTROL-8, or from Commodore-1 to Commodore-8. Hold down the CONTROL or Commodore key as you do this. Instantly, that color, and everything previously drawn in that color, is changed.

Three of the colors (including 1, the background color) can be any of the 16 colors. But because of the way multicolor works, color 4 (represented by bit pattern 11, or 3 in decimal) can only be one of the 8 CONTROL colors. Assigning it one of the Commodore logo colors just picks the color shown on the face of the color key. Incidentally, it's the color of bit pattern 3 (color 4) that changes according to the character color as set in color memory. The other colors are programmed in multicolor registers 1 and 2 (locations 53282 and 53283), so all characters share these two colors. When you want to vary a certain color without affecting the rest of the characters, you'll want to draw it in color 4.

Some of the commands in the multicolor mode aren't as useful as others. You have to press f1 and f2 twice to shift a character, since they only shift one bit, which causes all the colors to change. You can use CONTROL-R or CONTROL-9 (Reverse) to reverse all the colors (color 1 becomes color 4, color 2 becomes color 3, color 3 becomes color 2, and color 4 becomes color 1). R (Rotate) changes all the colors and is rather useless unless you press it twice to just turn the characters upside down. M (Mirror), works as it did before except that colors 2 and 3 are switched. And you can still copy characters using f7 and f8 (see above).

Returning to Normal

You can switch instantly back to the normal character mode by pressing f6. If you were drawing in multicolor, you can see the bit patterns that make up each color. Multicolor characters look just as strange in normal mode as normal characters look in multicolor.

If you changed colors in the multicolor mode, some of the colors in the normal mode may have been altered. You can change these colors just as you did in multicolor mode. Press SHIFT-1 to change the color of the empty pixels, and SHIFT-2 to change the color of the *on* pixels. Use SHIFT-4 to change the color of the eight rows of characters.

Notes: How to Use the DATA Statements

The DATA statements are created from lines 63000 and up, as many as necessary. Each line of data has nine numbers. The first number is the internal code of the character (the code you use when POKEing to the screen). It represents an offset into the table of character patterns. The eight bytes that follow are the decimal numbers for the eight bytes needed to define any character. Here's a sample program to read them and display them:

```
10 POKE 56,48:CLR                                  :rem 174
50 READ A:IF A=-1 THEN 70                          :rem 253
60 FORI=0 TO 7:READ B:POKE 12288+A*8+I,B:NEXT:GOTO
   50                                              :rem 228
70 PRINT CHR$(147);"{10 DOWN}":REM TEN CURSOR DOWN
   S                                               :rem 121
80 FOR I=0TO7:FORJ=0TO31:POKE 1028+J+I*40,I*32+J:P
   OKE55300+J+I*40,1:NEXT:NEXT                      :rem 14
90 POKE 53272,(PEEK(53272)AND240)OR 12:END :rem 15
```

You'll also need to add the following line to the end of your DATA statements:

```
63999 DATA -1
```

If you want to have your cake and eat it, too—that is, also have the normal ROM patterns—copy them from ROM down to RAM by adding:

```
20 POKE 56334,PEEK(56334)AND254:POKE 1,PEEK(1)AND
   {SPACE}251
30 FOR I=0 TO 2047:POKE 12288+I,PEEK(53248+I):NEXT
40 POKE 1,PEEK(1)OR4:POKE 56334,PEEK(56334)OR1
```

Quick Reference: Ultrafont + Commands

Cursor keys	Move to next character
HOME (CLR/HOME)	Moves the cursor to upper-left corner. Press twice to go back to start
V	Cursor velocity; answer from 0 (slow) to 9 (fast)
f1	Scrolls right with wraparound
f2(SHIFT-f1)	Scrolls left
f3	Scrolls down
f4(SHIFT-f3)	Scrolls up
R	Rotates 90 degrees; press twice to invert
M	Mirror image
CLR (SHIFT-CLR/HOME)	Erases current character
CONTROL-R, CONTROL-9	Reverse pixels
CONTROL-back arrow (←), CONTROL F	Copy first four rows of characters, reversed, to bottom four
F	Fix characters from ROM pattern
L	Load. Tape or Disk, *Filename*
S	Save. Tape or Disk, *Filename*
T	Typing mode: RUN/STOP to exit
f7	Memorizes character (keep)
f8 (SHIFT-f7)	Recalls character (put)
f5	Switches to multicolor character mode
f6 (SHIFT-f5)	Returns to normal character mode
CONTROL-D	Makes DATA statements
SHIFT-RUN/STOP	Restarts Ultrafont +
CONTROL-X	Exits Ultrafont + to BASIC

Ultrafont +

For mistake-proof program entry, be sure to use "MLX" (Appendix D).

```
49152 :076,019,197,000,001,003,040
49158 :004,000,001,003,004,000,018
49164 :173,048,002,072,173,045,013
49170 :002,141,048,002,141,079,175
49176 :002,032,047,193,104,141,031
49182 :048,002,169,100,133,252,222
49188 :169,000,133,251,133,167,121
49194 :169,216,133,168,169,008,137
49200 :141,040,002,169,002,141,031
49206 :042,002,169,005,141,041,198
49212 :002,174,003,192,173,079,171
49218 :002,205,048,002,208,002,021
49224 :162,002,142,080,002,160,108
```

```
49230 :000,177,253,170,173,063,146
49236 :002,240,003,076,233,192,062
49242 :169,207,145,251,138,010,242
49248 :170,176,008,173,080,002,193
49254 :145,167,076,112,192,173,199
49260 :004,192,145,167,200,192,240
49266 :008,208,221,024,165,251,223
49272 :105,008,133,251,133,167,149
49278 :165,252,105,000,133,252,009
49284 :105,116,133,168,024,165,075
49290 :253,105,008,133,253,165,031
49296 :254,105,000,133,254,056,178
49302 :238,079,002,206,041,002,206
49308 :173,041,002,208,156,056,024
49314 :173,079,002,233,005,141,027
49320 :079,002,056,165,253,233,188
49326 :039,133,253,165,254,233,227
49332 :000,133,254,206,040,002,047
49338 :173,040,002,240,003,076,208
49344 :056,192,206,042,002,173,095
49350 :042,002,240,030,169,008,177
49356 :141,040,002,024,173,079,151
49362 :002,105,032,141,079,002,059
49368 :024,165,253,105,248,133,120
49374 :253,165,254,105,000,133,108
49380 :254,076,056,192,096,134,012
49386 :097,169,000,141,043,002,174
49392 :006,097,046,043,002,006,184
49398 :097,046,043,002,174,043,139
49404 :002,169,207,145,251,200,202
49410 :169,247,145,251,136,189,115
49416 :003,192,145,167,200,145,092
49422 :167,200,192,008,208,215,236
49428 :076,117,192,169,000,141,203
49434 :026,208,165,001,041,251,206
49440 :133,001,096,165,001,009,181
49446 :004,133,001,169,001,141,231
49452 :026,208,096,169,000,133,164
49458 :254,173,048,002,010,133,158
49464 :253,038,254,006,253,038,130
49470 :254,006,253,038,254,169,012
49476 :112,005,254,133,254,096,154
49482 :032,047,193,160,000,177,171
49488 :253,073,255,145,253,200,235
49494 :192,008,208,245,032,012,015
49500 :192,096,169,102,133,252,012
49506 :169,218,133,168,173,058,249
49512 :002,174,063,002,240,002,075
49518 :009,008,141,080,002,169,007
49524 :132,133,251,133,167,162,070
```

```
49530 :008,169,000,133,097,160,177
49536 :000,165,097,145,251,230,248
49542 :097,173,080,002,145,167,030
49548 :200,192,032,208,240,024,012
49554 :165,251,105,040,133,251,067
49560 :133,167,165,252,105,000,206
49566 :133,252,105,116,133,168,041
49572 :202,208,216,096,032,169,063
49578 :203,173,044,002,141,024,245
49584 :208,169,200,013,063,002,063
49590 :141,022,208,169,000,141,095
49596 :032,208,141,033,208,032,074
49602 :094,193,173,058,002,174,120
49608 :063,002,240,002,009,008,012
49614 :141,134,002,165,209,133,222
49620 :243,024,165,210,105,116,051
49626 :133,244,164,211,177,209,076
49632 :073,128,145,209,177,243,175
49638 :072,173,134,002,145,243,231
49644 :032,228,255,240,251,170,132
49650 :164,211,201,133,208,006,141
49656 :238,032,208,238,033,208,181
49662 :201,134,208,008,177,209,167
49668 :141,082,002,076,026,194,013
49674 :201,135,208,012,173,082,053
49680 :002,145,209,104,173,134,015
49686 :002,072,162,029,177,209,161
49692 :073,128,145,209,104,145,064
49698 :243,138,032,210,255,032,176
49704 :225,255,208,165,032,201,102
49710 :203,169,000,141,134,002,183
49716 :169,012,141,032,208,076,178
49722 :169,196,032,023,193,169,072
49728 :112,133,252,173,083,002,051
49734 :133,254,162,008,169,000,028
49740 :133,253,133,251,168,177,167
49746 :253,145,251,200,208,249,108
49752 :230,254,230,252,202,208,184
49758 :242,165,252,201,128,240,042
49764 :007,169,208,133,254,076,179
49770 :072,194,032,035,193,162,026
49776 :004,189,006,192,157,002,150
49782 :192,202,208,247,096,169,208
49788 :112,133,252,169,116,133,015
49794 :254,169,000,133,253,133,048
49800 :251,168,162,004,177,251,125
49806 :073,255,145,253,200,208,252
49812 :247,230,254,230,252,202,027
49818 :208,240,096,032,047,193,202
49824 :160,000,177,253,010,008,000
```

```
49830 :074,040,042,145,253,200,152
49836 :192,008,208,242,076,012,142
49842 :192,032,047,193,160,000,034
49848 :177,253,074,008,010,040,234
49854 :106,145,253,200,192,008,070
49860 :208,242,076,012,192,032,190
49866 :047,193,160,000,177,253,008
49872 :133,097,200,177,253,136,180
49878 :145,253,200,200,192,008,188
49884 :208,245,165,097,136,145,192
49890 :253,076,012,192,032,047,070
49896 :193,160,007,177,253,133,131
49902 :097,136,177,253,200,145,222
49908 :253,136,016,247,200,165,237
49914 :097,145,253,076,012,192,001
49920 :032,047,193,160,000,169,089
49926 :000,133,097,162,008,177,071
49932 :253,010,102,097,202,208,116
49938 :250,165,097,145,253,200,104
49944 :192,008,208,233,076,209,182
49950 :200,032,047,193,160,008,158
49956 :169,000,153,048,002,136,032
49962 :208,250,169,007,133,097,138
49968 :152,170,169,000,133,007,167
49974 :177,253,074,145,253,038,226
49980 :007,202,016,251,166,097,031
49986 :165,007,029,049,002,157,219
49992 :049,002,198,097,165,097,168
49998 :016,224,200,192,008,208,158
50004 :215,136,185,049,002,145,048
50010 :253,136,016,248,076,012,063
50016 :192,032,047,193,160,000,208
50022 :152,145,253,200,192,008,028
50028 :208,249,076,012,192,120,197
50034 :169,127,141,013,220,169,185
50040 :001,141,026,208,169,177,074
50046 :141,018,208,169,027,141,062
50052 :017,208,169,146,141,020,065
50058 :003,169,195,141,021,003,158
50064 :088,096,173,018,208,201,160
50070 :177,208,039,169,242,141,102
50076 :018,208,173,044,002,141,230
50082 :024,208,173,022,208,041,070
50088 :239,013,063,002,141,022,136
50094 :208,173,057,002,141,033,020
50100 :208,169,001,141,025,208,164
50106 :104,168,104,170,104,064,132
50112 :169,177,141,018,208,169,050
50118 :158,141,024,208,173,032,166
50124 :208,141,033,208,169,200,139
```

```
50130 :141,022,208,238,037,208,040
50136 :169,001,141,025,208,076,068
50142 :049,234,085,064,000,064,206
50148 :064,000,076,064,000,076,252
50154 :064,000,076,064,000,076,002
50160 :064,000,064,064,000,085,005
50166 :064,000,000,000,085,080,219
50172 :000,064,016,000,064,016,156
50178 :000,064,016,000,064,016,162
50184 :000,064,016,000,064,016,168
50190 :000,064,016,000,064,016,174
50196 :000,085,080,000,000,000,185
50202 :000,255,255,255,000,001,024
50208 :001,001,000,255,001,000,034
50214 :000,255,001,000,000,255,037
50220 :001,018,085,076,084,082,134
50226 :065,070,079,078,084,032,202
50232 :043,032,086,046,050,146,203
50238 :095,069,082,082,079,082,039
50244 :032,079,078,032,083,065,181
50250 :086,069,047,076,079,065,240
50256 :068,095,018,084,146,065,044
50262 :080,069,032,079,082,032,204
50268 :018,068,146,073,083,075,043
50274 :063,095,070,073,076,069,032
50280 :078,065,077,069,058,095,034
50286 :069,078,084,069,082,032,012
50292 :067,079,076,079,082,032,019
50298 :075,069,089,095,085,083,106
50304 :069,032,082,079,077,032,243
50310 :083,069,084,063,032,040,249
50316 :089,047,078,041,095,018,252
50322 :085,146,080,080,069,082,176
50328 :067,065,083,069,032,079,035
50334 :082,032,018,076,146,079,079
50340 :087,069,082,063,095,169,217
50346 :045,160,196,133,251,132,063
50352 :252,160,040,169,032,153,214
50358 :191,103,136,208,250,177,223
50364 :251,200,201,095,208,249,112
50370 :136,132,097,152,074,073,090
50376 :255,056,105,020,168,162,198
50382 :024,024,032,240,255,160,173
50388 :000,177,251,032,210,255,113
50394 :200,196,097,144,246,096,173
50400 :133,251,132,252,160,040,168
50406 :169,032,153,191,103,136,246
50412 :208,250,162,024,160,000,016
50418 :024,032,240,255,160,000,185
50424 :177,251,201,095,240,006,194
```

```
50430 :032,210,255,200,208,244,123
50436 :096,174,076,002,240,008,088
50442 :160,000,200,208,253,202,009
50448 :208,250,096,173,002,221,198
50454 :009,003,141,002,221,173,059
50460 :000,221,041,252,009,002,041
50466 :141,000,221,169,100,141,038
50472 :136,002,169,147,032,210,224
50478 :255,169,000,141,134,002,235
50484 :169,008,032,210,255,160,118
50490 :000,152,153,128,099,200,022
50496 :016,250,168,185,224,195,078
50502 :153,128,099,200,192,023,097
50508 :208,245,160,000,185,247,097
50514 :195,153,192,099,200,192,089
50520 :032,208,245,169,156,141,015
50526 :044,002,169,012,141,032,238
50532 :208,169,128,141,138,002,118
50538 :032,113,195,169,048,141,036
50544 :076,002,169,011,141,057,056
50550 :002,169,007,169,000,141,094
50556 :048,002,141,045,002,141,247
50562 :063,002,173,006,192,009,063
50568 :008,141,058,002,173,004,010
50574 :192,141,034,208,173,005,127
50580 :192,141,035,208,032,012,000
50586 :192,032,094,193,169,203,013
50592 :205,011,192,240,017,141,198
50598 :011,192,162,208,142,083,196
50604 :002,032,060,194,032,012,248
50610 :192,076,198,197,169,126,112
50616 :160,196,032,173,196,032,205
50622 :228,255,240,251,201,078,163
50628 :240,029,169,145,160,196,111
50634 :032,173,196,032,228,255,094
50640 :240,251,162,208,201,076,066
50646 :208,002,162,216,142,083,003
50652 :002,032,060,194,032,012,040
50658 :192,032,169,196,169,142,102
50664 :141,248,103,169,143,141,153
50670 :249,103,169,003,141,021,156
50676 :208,169,024,141,000,208,226
50682 :169,000,141,016,208,169,185
50688 :051,141,001,208,169,176,234
50694 :141,003,208,169,053,141,209
50700 :002,208,169,000,141,029,049
50706 :208,141,023,208,141,038,009
50712 :208,169,003,141,028,208,013
50718 :169,000,141,059,002,141,030
50724 :060,002,173,000,220,072,051
```

```
50730 :041,015,073,015,141,061,132
50736 :002,104,041,016,141,062,158
50742 :002,032,228,255,240,006,049
50748 :032,197,199,076,038,198,032
50754 :032,005,197,173,062,002,025
50760 :208,003,032,088,199,173,007
50766 :062,002,073,016,141,075,191
50772 :002,173,061,002,240,204,254
50778 :174,061,002,189,023,196,223
50784 :172,063,002,240,001,010,072
50790 :024,109,059,002,141,059,240
50796 :002,024,173,060,002,125,238
50802 :034,196,141,060,002,174,209
50808 :059,002,016,027,162,000,130
50814 :142,059,002,173,048,002,040
50820 :041,031,240,015,206,045,198
50826 :002,162,007,173,063,002,035
50832 :240,002,162,006,142,059,243
50838 :002,174,059,002,224,040,139
50844 :144,022,162,039,142,059,212
50850 :002,173,048,002,041,031,203
50856 :201,031,240,008,238,045,163
50862 :002,162,032,142,059,002,061
50868 :172,060,002,016,026,160,104
50874 :000,140,060,002,173,048,097
50880 :002,201,032,144,014,056,129
50886 :173,045,002,233,032,141,056
50892 :045,002,160,007,140,060,106
50898 :002,172,060,002,192,016,142
50904 :144,026,160,015,140,060,249
50910 :002,173,048,002,201,224,104
50916 :176,014,024,173,045,002,150
50922 :105,032,141,045,002,160,207
50928 :008,140,060,002,173,059,170
50934 :002,172,060,002,074,074,118
50940 :074,192,008,144,002,105,009
50946 :031,109,045,002,141,048,122
50952 :002,041,224,074,074,105,016
50958 :176,141,003,208,173,048,251
50964 :002,041,031,010,010,010,124
50970 :105,053,141,002,208,169,192
50976 :000,105,000,133,097,173,028
50982 :060,002,010,010,010,105,235
50988 :051,141,001,208,173,059,165
50994 :002,010,010,010,038,097,217
51000 :105,024,141,000,208,165,187
51006 :097,105,000,141,016,208,117
51012 :173,048,002,205,081,002,067
51018 :240,009,032,012,192,173,220
51024 :048,002,141,081,002,076,174
```

```
51030 :038,198,032,047,193,173,255
51036 :060,002,041,007,168,173,031
51042 :059,002,041,007,073,007,031
51048 :170,232,134,097,056,169,194
51054 :000,042,202,208,252,174,220
51060 :063,002,208,048,133,097,155
51066 :173,075,002,208,022,169,003
51072 :000,141,064,002,141,038,002
51078 :208,177,253,037,097,208,090
51084 :008,169,001,141,064,002,013
51090 :141,038,208,165,097,073,100
51096 :255,049,253,174,064,002,181
51102 :240,002,005,097,145,253,132
51108 :032,012,192,096,133,098,215
51114 :074,005,098,073,255,049,212
51120 :253,166,097,202,133,097,100
51126 :173,066,002,074,042,202,229
51132 :208,252,005,097,145,253,124
51138 :076,012,192,141,065,002,170
51144 :174,225,199,221,225,199,163
51150 :240,004,202,208,248,096,180
51156 :202,138,010,170,189,006,159
51162 :200,072,189,005,200,072,188
51168 :096,035,133,137,134,138,129
51174 :077,082,147,018,145,017,204
51180 :157,029,070,135,139,049,047
51186 :050,051,052,019,136,140,178
51192 :033,034,035,036,086,083,043
51198 :076,024,004,006,131,084,067
51204 :005,178,194,156,194,229,192
51210 :194,200,194,255,194,030,053
51216 :195,096,195,073,193,083,083
51222 :200,105,200,127,200,149,235
51228 :200,173,200,246,200,025,048
51234 :201,042,201,042,201,042,251
51240 :201,042,201,065,201,090,072
51246 :201,112,201,130,201,130,253
51252 :201,130,201,130,201,204,095
51258 :201,002,203,142,203,162,203
51264 :203,035,204,122,194,074,128
51270 :200,167,193,216,200,162,184
51276 :255,154,032,129,255,076,209
51282 :019,197,173,060,002,041,062
51288 :007,133,097,056,173,060,102
51294 :002,233,008,056,229,097,207
51300 :141,060,002,076,169,200,236
51306 :173,060,002,041,007,133,010
51312 :097,024,173,060,002,105,061
51318 :008,056,229,097,141,060,197
51324 :002,076,169,200,173,059,035
```

```
51330 :002,041,007,133,097,056,210
51336 :173,059,002,233,008,056,155
51342 :229,097,141,059,002,076,234
51348 :169,200,173,059,002,041,024
51354 :007,133,097,024,173,059,135
51360 :002,105,008,056,229,097,145
51366 :141,059,002,104,104,076,140
51372 :119,198,032,047,193,032,025
51378 :023,193,160,007,024,173,246
51384 :083,002,101,254,105,143,104
51390 :133,252,165,253,133,251,097
51396 :177,251,145,253,136,016,150
51402 :249,032,035,193,076,012,031
51408 :192,173,063,002,208,003,081
51414 :076,012,192,032,047,193,254
51420 :160,007,177,253,162,004,215
51426 :074,008,074,102,097,040,109
51432 :102,097,202,208,245,165,227
51438 :097,145,253,136,016,234,095
51444 :076,012,192,169,016,141,082
51450 :063,002,169,001,141,029,143
51456 :208,032,012,192,032,094,058
51462 :193,169,050,141,065,002,114
51468 :032,043,201,173,059,002,010
51474 :041,254,141,059,002,076,079
51480 :169,200,169,000,141,063,254
51486 :002,141,029,208,169,001,068
51492 :032,055,201,032,012,192,048
51498 :096,173,063,002,208,001,073
51504 :096,056,173,065,002,233,161
51510 :049,141,066,002,170,189,159
51516 :003,192,141,038,208,096,226
51522 :173,059,002,013,060,002,119
51528 :208,003,141,045,002,169,128
51534 :000,141,059,002,141,060,225
51540 :002,032,012,192,076,169,055
51546 :200,032,074,193,032,074,183
51552 :193,032,047,193,160,000,209
51558 :177,253,153,067,002,200,186
51564 :192,008,208,246,096,032,122
51570 :047,193,160,000,185,067,254
51576 :002,145,253,200,192,008,152
51582 :208,246,076,012,192,169,005
51588 :110,160,196,032,173,196,231
51594 :032,228,255,240,251,162,026
51600 :000,221,218,232,240,008,039
51606 :232,224,016,208,246,076,128
51612 :169,196,056,173,065,002,049
51618 :233,033,168,138,153,003,122
51624 :192,192,003,240,010,192,229
```

```
51630 :000,240,022,153,033,208,062
51636 :076,199,201,174,063,002,127
51642 :240,002,041,007,141,058,163
51648 :002,153,003,192,032,094,156
51654 :193,032,012,192,076,169,104
51660 :196,169,241,160,201,032,179
51666 :173,196,032,228,255,056,126
51672 :233,048,048,248,201,010,236
51678 :176,244,133,097,056,169,073
51684 :009,229,097,010,010,010,081
51690 :010,141,076,002,076,169,196
51696 :196,067,085,082,083,079,064
51702 :082,032,086,069,076,079,158
51708 :067,073,084,089,032,040,125
51714 :048,045,057,041,063,095,095
51720 :160,000,140,078,002,169,045
51726 :164,032,210,255,169,157,233
51732 :032,210,255,032,228,255,008
51738 :240,251,172,078,002,133,134
51744 :097,169,032,032,210,255,059
51750 :169,157,032,210,255,165,002
51756 :097,201,013,240,039,201,067
51762 :020,208,013,192,000,240,211
51768 :209,136,169,157,032,210,201
51774 :255,076,010,202,041,127,005
51780 :201,032,144,194,192,020,083
51786 :240,190,165,097,153,000,151
51792 :002,032,210,255,200,076,087
51798 :010,202,169,095,153,000,203
51804 :002,152,096,032,231,255,092
51810 :169,082,160,196,032,173,142
51816 :196,032,228,255,240,251,026
51822 :162,001,201,084,240,011,041
51828 :162,008,201,068,240,005,032
51834 :104,104,076,169,196,141,144
51840 :077,002,160,000,169,001,025
51846 :032,186,255,169,100,160,012
51852 :196,032,224,196,032,008,060
51858 :202,208,007,173,077,002,047
51864 :201,084,208,237,173,077,108
51870 :002,201,068,208,069,169,107
51876 :083,141,020,002,169,048,115
51882 :141,021,002,169,058,141,190
51888 :022,002,160,000,185,000,033
51894 :002,153,023,002,200,204,254
51900 :078,002,208,244,200,200,096
51906 :200,173,065,002,201,083,150
51912 :208,026,152,072,160,002,052
51918 :162,020,032,189,255,169,009
51924 :015,162,008,160,015,032,092
```

```
51930 :186,255,032,192,255,032,146
51936 :231,255,104,168,136,076,170
51942 :246,202,160,000,185,000,255
51948 :002,153,021,002,200,204,050
51954 :078,002,208,244,152,162,064
51960 :021,160,002,032,189,255,139
51966 :169,160,133,178,096,032,254
51972 :095,202,032,169,203,169,106
51978 :000,133,253,133,251,169,181
51984 :112,133,252,162,255,160,066
51990 :119,169,251,032,216,255,040
51996 :176,011,032,183,255,208,125
52002 :006,032,201,203,076,169,209
52008 :196,032,201,203,032,231,167
52014 :255,173,077,002,201,068,054
52020 :240,015,169,063,160,196,127
52026 :032,173,196,032,228,255,206
52032 :240,251,076,169,196,169,141
52038 :000,032,189,255,169,015,218
52044 :162,008,160,015,032,186,127
52050 :255,032,192,255,162,015,225
52056 :032,198,255,160,000,032,253
52062 :207,255,201,013,240,007,249
52068 :153,000,002,200,076,093,112
52074 :203,169,095,153,000,002,216
52080 :032,204,255,169,000,160,164
52086 :002,032,173,196,162,015,186
52092 :032,201,255,169,073,032,118
52098 :210,255,169,013,032,210,251
52104 :255,032,231,255,076,061,022
52110 :203,032,095,202,032,169,107
52116 :203,169,000,162,000,160,074
52122 :112,032,213,255,176,137,055
52128 :076,201,203,169,004,141,186
52134 :136,002,000,120,169,000,081
52140 :141,026,208,169,255,141,088
52146 :013,220,169,049,141,020,022
52152 :003,169,234,141,021,003,243
52158 :169,000,141,021,208,169,130
52164 :147,088,076,210,255,169,117
52170 :147,032,210,255,032,113,223
52176 :195,169,003,141,021,208,177
52182 :032,012,192,032,094,193,001
52188 :076,169,196,248,169,000,054
52194 :141,000,001,141,001,001,255
52200 :224,000,240,021,202,024,175
52206 :173,000,001,105,001,141,147
52212 :000,001,173,001,001,105,013
52218 :000,141,001,001,076,232,189
52224 :203,216,173,001,001,009,091
```

```
52230 :048,141,002,001,173,000,115
52236 :001,041,240,074,074,074,004
52242 :074,009,048,141,001,001,036
52248 :173,000,001,041,015,009,007
52254 :048,141,000,001,096,096,156
52260 :056,165,045,233,002,133,158
52266 :045,165,046,233,000,133,152
52272 :046,169,024,133,057,169,134
52278 :246,133,058,169,000,141,033
52284 :079,002,133,251,133,253,143
52290 :169,112,133,254,173,083,222
52296 :002,133,252,032,023,193,195
52302 :160,000,177,251,209,253,104
52308 :208,062,200,192,008,208,194
52314 :245,238,079,002,024,165,075
52320 :253,105,008,133,253,133,213
52326 :251,165,254,105,000,133,242
52332 :254,109,083,002,105,143,036
52338 :133,252,173,079,002,208,193
52344 :213,169,000,168,145,045,092
52350 :200,145,045,024,165,045,238
52356 :105,002,133,045,165,046,116
52362 :105,000,133,046,032,035,233
52368 :193,076,051,165,160,000,021
52374 :024,165,045,105,041,145,163
52380 :045,200,165,046,105,000,205
52386 :145,045,200,165,057,145,151
52392 :045,200,165,058,145,045,058
52398 :200,169,131,145,045,174,014
52404 :079,002,032,223,203,200,151
52410 :173,002,001,145,045,200,240
52416 :173,001,001,145,045,200,245
52422 :173,000,001,145,045,200,250
52428 :132,097,160,000,132,098,055
52434 :177,253,170,032,223,203,244
52440 :164,097,169,044,145,045,112
52446 :200,173,002,001,145,045,020
52452 :173,001,001,200,145,045,025
52458 :173,000,001,200,145,045,030
52464 :200,132,097,164,098,200,107
52470 :192,008,208,214,164,097,105
52476 :169,000,145,045,160,000,003
52482 :177,045,072,200,177,045,206
52488 :133,046,104,133,045,230,187
52494 :057,208,002,230,058,076,133
52500 :091,204,013,013,013,013,111
```

Advanced Sound Effects on the 128

Philip I. Nelson

Here are some secrets to creating unusual sound effects with the Commodore 128's built-in synthesizer chip. Using the accompanying program, you can experiment with different sounds without programming.

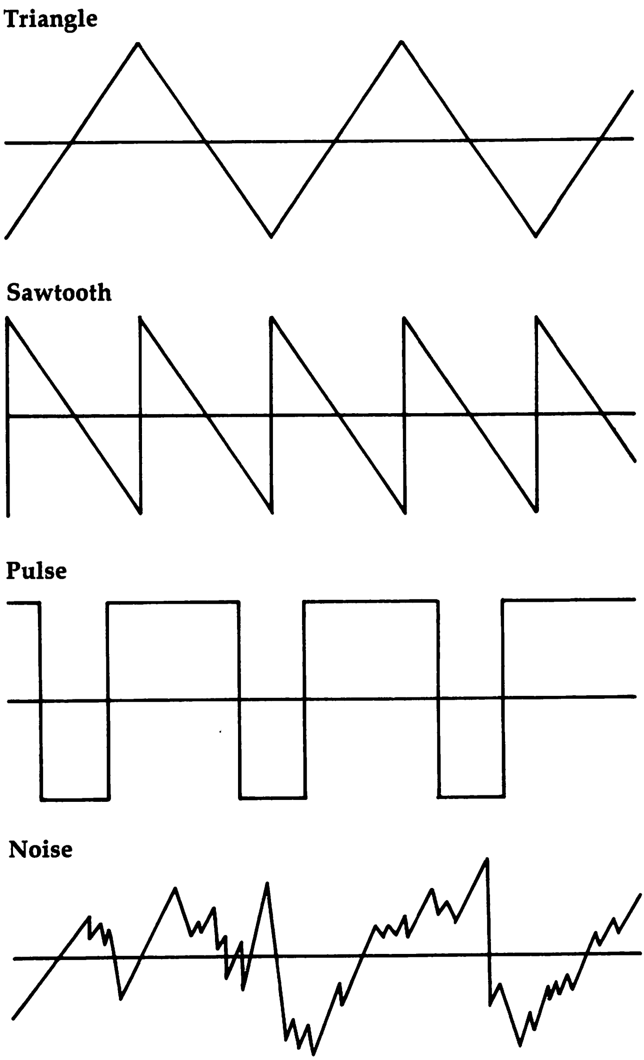
The Commodore 128's SID (Sound Interface Device) chip is capable of creating rich, extraordinarily complex sounds—but its power doesn't come without a price. There aren't any sound commands in Commodore BASIC 2.0, the BASIC available in the computer's 64 mode, so everything must be done with POKEs. It's tedious to look up all those POKE values and easy to get sidetracked, since you must define several *parameters* (controlling values) to make even a simple sound. Many programmers, including professionals, grow frustrated and settle for crude beeps and whooping noises, wasting the machine's classiest sound features.

The program at the end of this article is designed to help beginners learn about two of the SID chip's advanced sound effects: ring modulation and synchronization. It lets you produce a tone with two sound channels, and also switch either effect on and off just by pressing one of the 128's special function keys. Don't worry if the following explanations seem confusing at first; they'll make more sense after you've tried the program.

Independent Voices

Any sound can be visualized as a *waveform*, like the cross section of a ripple on a pond. When in the 64 mode, the Commodore 128 is capable of reproducing four different waveforms. Three of them (the triangle, sawtooth, and pulse waves) produce clear tones, and the fourth (the noise wave) makes a rushing or hissing sound. Figure 1 represents each of these waveforms. You can assign any one of the four waveforms to any of the 128's three sound channels, or *voices*.

Figure 1. SID Chip Waveforms



Each of the computer's three voices normally plays independently. That is, each voice sounds the same, no matter what the other two are doing. If you make voice 1 beep and voice 2 growl, voice 1 always makes the same beep even if you change voice 2's growl to a screech. For a simple analogy, picture each voice as playing through a separate channel, like the two channels on a home stereo system.

Ring modulation and synchronization go beyond this to create *interactive* effects, in which a parameter controlling one voice also affects the sound produced by a second voice. In both cases, the special effect is created by a difference in the frequencies (pitches) of the two voices.

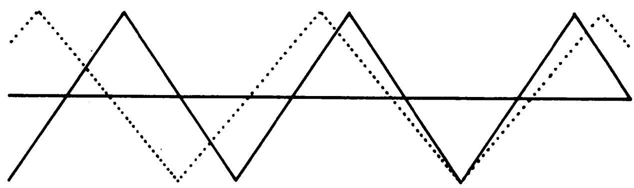
Synchronization

Synchronization is the simpler of the two effects. You could imagine it as mixing two voices in one channel so that their waveforms intermingle. The result is often a rhythmic or beating effect, produced as the peaks and valleys of the two waves move in and out of step with each other.

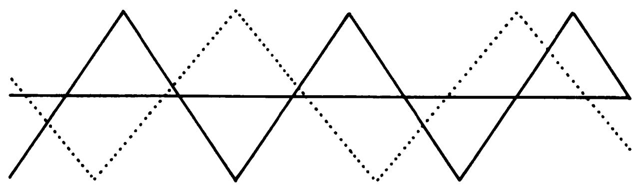
When the two waves are more nearly in step, their combined sound is more pronounced. When their peaks and valleys are more nearly opposed, they tend to cancel each other out, and the combined sound is quieter. Figure 2 shows a simplified diagram of both extremes.

Figure 2. Synchronization

Waves nearly in step



Waves far out of step



If you program both voices so that their frequencies are always identical, synchronization produces no audible effect.

In addition to the original tones each waveform produces by itself, synchronization adds nonharmonic *overtones* (also called *sidebands*). The overtones are entirely new waveforms which would not exist without synchronization. For instance, imagine someone pounding a huge gong. Gong sounds are full of nonharmonic overtones, which are created as different areas of the big, flexible metal plate vibrate in and out of phase.

In simplest terms, synchronizing two voices gives you both original tones plus new overtones. However, the original tones predominate.

Ring Modulation

Ring modulation is a special type of synchronization in which overtones almost completely suppress the original tones. What you're left with is a sound composed chiefly of nonharmonic overtones. The results are often surprising and bear little if any resemblance to so-called natural sounds.

Used with care, ring modulation can produce haunting, beautiful effects. However, it works through a complex interaction of two waveforms, largely suppressing what you'd hear without the feature. So it can be difficult to handle if you don't know how it works in the first place.

Experimenting with Effects

Let's hear how these effects sound. Type in the program "Sound Effects," save it, and type RUN. The program is set up with several default parameters, so to hear a quick example, just press RETURN at every prompt. The default parameters will be displayed in each case.

You should hear a flutey tone sweeping up the scale, over and over. To pause the tone during its upward sweep, press the CONTROL key. (Don't worry about accidentally hitting the RUN/STOP key; it's been disabled.)

To switch on synchronization, press the f7 function key. The f5 key switches on ring modulation, and the f3 key activates both effects at once.

When synchronization is selected, you'll hear the beating effect as the tone ascends in pitch and the two voices move in and out of phase with each other. Ring modulation creates a rich, spacey sound. Note that you can pause the tone with CONTROL while pressing a function key. As you'll hear, the sounds are far less exciting when both frequencies remain fixed. The most interesting effects are made by changing parameters in realtime.

In these two-voice effects, one of the voices is called the *carrier*; the other, the *program* voice. These terms are derived from electronics, meaning that the first voice *carries* the signal (produces the basic sound), and the second voice *programs* (modulates) it. In this example program, voice 1 produces the carrier tone, and voice 3 programs voice 1.

In both synchronization and ring modulation, it is the *frequency* of the program voice which affects the carrier voice. The other program voice parameters have no effect on the carrier (of course, they will affect the program voice if it is turned on).

Shifting Frequencies

Now that you've heard these special effects with the program voice set for a fixed frequency, let's try changing the frequency while the tone is being produced. To raise the frequency of the program voice, press either SHIFT key. To lower it, press the Commodore logo key (next to the SHIFT on the left side of the keyboard). The most pronounced effects are produced by decreasing the program frequency during a rising tone, and vice versa.

Now let's hear a descending tone. Press the f1 key to stop the sound, and enter the following values when prompted:

Rising/falling?	**F**
Carrier waveform	**T**
Program waveform	**(any waveform works)**
Hear program voice?	**N**
Program frequency	**9**
Starting frequency	**200**
Ending frequency	**5**
Loop rate	**6**

Experiment with the program for a while, trying out different parameters. For example, try producing the same sound with a smaller loop rate. Press f1 to enter edit mode, then press RETURN after the first seven prompts. Now enter 0.75 for the loop rate. Pressing RETURN at a prompt preserves the old value, so you need to type in only the parameters you want to change (however, you must always enter the loop rate for a falling tone).

When picking the waveforms, press T for a triangle wave, P for the pulse waveform, and so on. When you select a rising tone, the starting frequency must be smaller than the ending frequency. To create a falling tone, the first value must be larger than the second. If you make a mistake, use the INST/DEL key to back up. The program signals an error if you enter illegal values. If you accidentally type in a letter when a number is required, the computer prints ?REDO FROM START. No harm is done; just enter the number you want.

The loop rate controls how fast the carrier frequency is changed as the tone moves up or down the scale. It corresponds to the STEP value in the FOR-NEXT loop that creates the tone (see lines 13–17 in the program). The smaller the loop rate (fractions are allowed), the slower the frequency will change, and vice versa. When the starting and ending frequencies are far apart, you can specify a large value for the loop rate; however, if you specify a starting frequency that is close to the ending frequency, you must keep the loop rate small to avoid causing an error in the program.

Programming Your Own Sounds

You can use this program to start building a library of sound effects. Just play around until you find a sound you like, copy down the values from the screen, and plug them into your own program.

As you'll discover by experimenting, these special effects work well with certain combinations, and poorly (or not at all) with others. Ring modulation works only when you set the carrier voice to the triangle waveform. Synchronization works with any waveform, but synchronizing any frequency with the noise waveform (a nearly random combination of many frequencies) doesn't accomplish much. The sawtooth and pulse waves often sound similar.

Most of the time, you'll want to keep the program voice silent, using only its frequency to control the carrier (in which case its other parameters are irrelevant). However, you can press Y when prompted to hear the program voice. If you have trouble understanding how an effect works, try listening to the program voice for awhile.

Ring modulation and synchronization are most pronounced when the program frequency is considerably lower than the carrier frequency and remains fixed, as in the above examples. Changing the program frequency to a higher fixed value makes the two voices move in and out of phase more rapidly. Run the last example, and change the program frequency from 9 to 22. Now select synchronization, and you'll hear a sharp, *meow-meow* sound.

Controlling Voices with Voices

You can use ring modulation or synchronization with any of the 128's three voices, but the voice relationships are fixed: voice 1 modulates voice 2, voice 2 modulates voice 3, and voice 3 modulates voice 1.

Thus, if you want to synchronize or ring modulate voice 1, you must use voice 3 as the program voice, and so on. Again, it is the frequency of the program voice which affects the result. This simple tutorial program uses only the high-byte frequency register for each voice; of course, you can achieve much finer frequency control by using both the high and low bytes.

To select these special effects in BASIC, simply add 2, 4, or 6 to the normal POKE value for the waveform register of the voice you want to affect. For instance, POKE 54276,17 selects the triangle waveform for voice 1. POKE 54276,19 adds synchronization to the triangle wave (17+2=19). POKE 54276,21 enables a ring-modulated triangle wave; and POKE 54276,23 turns on both effects at once. Use POKE 54276,67 to select synchronization with the pulse waveform, and so forth.

Naturally, you can use these effects with more than one voice at a time. If you select synchronization in voices 1 and 3, then voice 1 will be affected by voice 3's frequency, and voice 3 will be affected by voice 2's frequency. However, because multivoice modulation creates so many overtones, it's easy for things to get out of hand. If you create a three-note musical chord with triangle waves in every voice, and then switch each to ring modulation, the result will be anything but musical.

Play with those frequencies for awhile, though, and you'll find you can push the *overtones* into complex chords. Such chords have a ringing, live sound and contain more than three notes. Interesting effects can also be created by tuning one or more voices slightly off-key.

Hints for Programmers

This program employs a few tricks you might find useful. Many programmers use a long series of individual POKEs to set up the SID chip at the beginning of a program. Line 1020 shows how to do this with a FOR-NEXT loop that READs the values from DATA statements and POKEs them into the SID chip. This makes your program easier for others to read and for you to modify. Note, however, that Commodore BASIC 2.0 (the BASIC used in the 64 mode) recommends POKEing attack/decay registers *before* waveform registers; the program follows this rule by POKEing the desired waveform values later on, in line 370.

To detect a single keypress, you can PEEK location 197 as is done in lines 14 and 15 (Z=197). Sometimes, however, you want to let the user do two things at once from the keyboard. In this program, for instance, you can select effects with a function key and simultaneously change the program frequency or pause the sound.

By PEEKing location 653, you can tell whether the CONTROL, SHIFT, or Commodore key is pressed with another key (see line 16; Y=653). Location 653 holds the following values when the indicated key is pressed:

1 = SHIFT
2 = Commodore
4 = CONTROL

You can also detect combinations of these keys. Location 653 contains a 3 when both SHIFT and the Commodore key are pressed, 5 when SHIFT and CONTROL are pressed, and so on. Checking for these keys gives you great flexibility in designing keyboard input. However, it's prudent to disable the RUN/STOP key when using them.

The program disables the RUN/STOP key in line 1010 with POKE 788,52. However, you can still exit the program by hitting RUN/STOP and RESTORE together. In the same line, POKE 657,128 prevents the computer from flipping the entire screen display from uppercase to lowercase if the SHIFT and Commodore keys are pressed simultaneously.

Sound Effects

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
Ø REM SOUND MODULATION DEMONSTRATOR          :rem 59
1 GOSUB 1ØØØ:GOTO1ØØ                         :rem 118
2 PRINTCHR$(145)C$:FORJ=1TO4ØØ:NEXT:PRINTER$CHR$(1
  45):RETURN                                 :rem 35
4 Z1=UN:ZZ=ED:RETURN                         :rem 1Ø9
5 ZZ=ZZ-LR:RETURN                            :rem 191
6 POKEW1,V1+TU:RETURN                        :rem 142
7 POKEW1,V1+FR:RETURN                        :rem 126
8 POKEW1,V1+SX:RETURN                        :rem 146
9 PF=PF+UN:IFPF>FFTHENPF=FF                  :rem 39
1Ø RETURN                                    :rem 65
11 PF=PF-UN:IFPF<UNTHENPF=UN                 :rem 126
12 RETURN                                    :rem 67
13 Z1=ZR:FORZZ=BGTOEDSTEPLR                  :rem 121
14 IFPEEK(Z)=NNTHENPOKEW1,V1:GOTO16          :rem 16
15 ONPEEK(Z)GOSUB1Ø,1Ø,6,4,8,7               :rem 2Ø2
16 ONPEEK(Y)GOSUB9,11,1Ø,5                   :rem 8
17 POKEH1,ZZ:POKEH3,PF:POKEBF,ZR:NEXT:IFZ1=UNTHEN1
   9                                         :rem 165
18 GOTO13                                    :rem 6
19 POKEH1,ZR:POKEH3,ZR:POKEW1,ZR:POKEW3,ZR:POKE198
   ,ZR                                       :rem 29
1ØØ PRINTFL$;:INPUTFF$                       :rem 137
11Ø IFFF$<>"R"ANDFF$<>"F"THENFF$="":GOSUB2:GOTO1ØØ
                                             :rem 184
12Ø PRINTUL$FF$:PRINTCV$;:INPUTVV$           :rem 238
13Ø IFVV$<>"T"ANDVV$<>"S"ANDVV$<>"P"ANDVV$<>"N"THE
    NGOSUB2:GOTO12Ø                          :rem Ø
14Ø FORJ=1TO4:IFVV$=VL$(J)THENV1=VC(J)       :rem 1Ø5
15Ø NEXT                                     :rem 213
16Ø PRINTUL$VV$:PRINTPV$;:INPUTVW$           :rem 32
17Ø IFVW$<>"T"ANDVW$<>"S"ANDVW$<>"P"ANDVW$<>"N"THE
    NGOSUB2:GOTO16Ø                          :rem 12
18Ø FORJ=1TO4:IFVW$=VL$(J)THENV3=VC(J)       :rem 112
19Ø NEXT                                     :rem 217
2ØØ PRINTUL$VW$:PRINTNF$;:INPUTYS$           :rem 9
21Ø IFYS$<>"Y"ANDYS$<>"N"THENGOSUB2:GOTO2ØØ
                                             :rem 158
22Ø IF YS$="N"THENV3=V3-UN                   :rem 16
23Ø PRINTUL$YS$:PRINTPF$;:INPUTPF            :rem 211
24Ø IFPF<UNORPF>FFTHENGOSUB2:GOTO23Ø         :rem 132
25Ø PRINTNL$PF:PRINTBG$;:INPUTBG             :rem 122
26Ø IFBG<ZRORBG>FFTHENGOSUB2:GOTO25Ø         :rem 119
27Ø PRINTNL$BG:PRINTED$;:INPUTED             :rem 111
28Ø IFED<ZRORBG=EDORED>FFTHENGOSUB2:GOTO27Ø
                                             :rem 1Ø7
```

```
290 IFFF$="R"ANDED<BGTHENGOSUB2:GOTO270    :rem 187
300 IFFF$="F"ANDED>BGTHENGOSUB2:GOTO270    :rem 169
310 PRINTNL$ED:PRINTLR$;:INPUTLR           :rem 148
320 IFLR<=ZRORLR>FFTHENGOSUB2:GOTO310      :rem 216
330 IFFF$="R"ANDLR>ED-BGTHENGOSUB2:GOTO310:rem 126
340 IFFF$="F"ANDLR>BG-EDTHENGOSUB2:GOTO310:rem 115
350 IFFF$="F"THENLR=-LR                    :rem 115
360 PRINTNL$ABS(LR):PRINTCHR$(158)A$:PRINTB$:PRINT
    F$:PRINTCHR$(158)A$                     :rem 63
370 POKEH3,PF:POKEW1,V1:POKEW3,V3           :rem 82
380 GOTO13                                  :rem 56
999 REM INITIALIZE                         :rem 129
1000 PRINTCHR$(147)CHR$(5)CHR$(142):POKE53281,0:PO
     KE53280,0:Z=197:BF=198:Y=653          :rem 188
1010 POKE657,128:POKE788,52:S=54272:VM=S+24:FORJ=S
     TOVM:POKEJ,0:NEXT                     :rem 146
1020 FORJ=STOVM:READQ:POKEJ,Q:NEXT          :rem 26
1025 FF$="R":BG=5:ED=125:LR=2:VV$="T":VW$="T":PF=1
     1:YS$="N"                             :rem 102
1030 ZR=0:UN=1:TU=2:FR=4:SX=6:NN=64:FF=255:H1=S+1:
     W1=S+4:H3=S+15:W3=S+18                 :rem 63
1040 R$=CHR$(18)                            :rem 51
1050 A$=R$+"{37 SPACES}"                    :rem 77
1060 PRINTA$                               :rem 185
1070 PRINTR$"{4 SPACES}SOUND MODULATION DEMONSTRAT
     OR{4 SPACES}"                          :rem 54
1080 PRINTA$                               :rem 187
1090 B$=R$+CHR$(158)+" F7=SYNCH F5=RING F3=BOTH F1
     =RESTART "+CHR$(159)                  :rem 108
1095 F$=R$+CHR$(158)+" CTRL=PAUSE COM=FREQ DN SHFT
     =FREQ UP "+CHR$(159)                  :rem 163
1100 C$=CHR$(158)+"{31 SPACES}"+R$+"ERROR "+CHR$(1
     59)                                   :rem 123
1105 ER$="{8 LEFT}{5 SPACES}"              :rem 235
1110 BL$=R$+CHR$(159)                       :rem 68
1115 UL$=CHR$(145):FORJ=1TO31:UL$=UL$+CHR$(29):NEX
     T:UL$=UL$+"{2 SPACES}"                :rem 104
1118 NL$=UL$+CHR$(157)                     :rem 165
1120 FL$=BL$+" RISING OR FALLING TONE? (R,F) "+CHR
     $(146)                                :rem 228
1130 BG$=BL$+" STARTING FREQUENCY{4 SPACES}(0-255)
      "+CHR$(146)                           :rem 80
1140 ED$=BL$+" ENDING FREQUENCY{6 SPACES}(0-255) "
     +CHR$(146)                            :rem 154
1150 LR$=BL$+" LOOP RATE{13 SPACES}(1-255) "+CHR$(
     146)                                  :rem 176
1160 CV$=BL$+" CARRIER WAVEFORM{4 SPACES}(T,S,P,N)
      "+CHR$(146)                          :rem 132
1170 PV$=BL$+" PROGRAM WAVEFORM{4 SPACES}(T,S,P,N)
      "+CHR$(146)                          :rem 162
```

```
1180 PF$=BL$+" PROGRAM FREQUENCY{5 SPACES}(1-255)
     {SPACE}"+CHR$(146)                        :rem 15
1190 NF$=BL$+" HEAR PROGRAM VOICE?{5 SPACES}(Y,N)
     {SPACE}"+CHR$(146)                        :rem 10
1200 FORJ=1TO4:READQ:VC(J)=Q:NEXT              :rem 85
1210 FORJ=1TO4:READQ$:VL$(J)=Q$:NEXT          :rem 203
1300 RETURN                                   :rem 164
2000 DATA 5,0,128,7,0,15,240:REMVOICE1         :rem 12
2010 DATA 0,0,0,0,0,0,0:REMVOICE2             :rem 251
2020 DATA 5,0,128,7,0,15,240:REMVOICE3         :rem 16
2030 DATA 0,0,0,15:REMFILTERS,VOLUME          :rem 148
2040 DATA 17,33,65,129:REMWAVEFORMS            :rem 17
2050 DATA T,S,P,N                             :rem 170
```

5 Utilities

5

NoZap
Automatic Program Saver

J. Blake Lambert

This short, useful disk routine automatically saves updated versions of the BASIC program you're working on. It also works with some ML assemblers, and is especially useful for those who live in areas where power dropouts frequently occur.

If you've ever been zapped by a power dropout or a loose power plug and have seen the ominous reset message, you know how it feels. The cost is high—your time and your work. It's easy to say *always make periodic backup copies as you type in or write programs.* But when the ideas are flowing, it's also easy to forget or procrastinate. "NoZap" does more than remind you—it does the SAVE for you, periodically and automatically.

NoZap is not a surge protector (it won't protect your computer from hardware damage resulting from a power spike), but it *will* protect you from momentary electric dropouts and loose connections that can cost you time and effort. Once you've run NoZap and entered a filename, it will save the current version of the program every ten minutes with an updated filename. You don't have to do anything—the operation is totally transparent. Every ten minutes, NoZap waits until you finish the line you're working on, and when you press RETURN to enter the line, it automatically saves.

NoZap even works with some programming utilities and typing aids. For example, it works with the "Automatic Proofreader," but not with "MLX." (Since MLX is a BASIC program, NoZap will back up the MLX program rather than the ML program you're entering.) It works with the DOS 5.1 Wedge, as well as with some assemblers, such as PAL and LADS.

NoZap keeps track of the size of the program you're working on, as well as automatically stamps a version number onto the beginning of the filename. NoZap can accommodate as many as 100 versions, numbered 01–99. After 99, the version number rolls over to 00.

There are a couple of limits which NoZap cannot work around: disk space and directory space. If not enough blocks are free, the program won't be saved. And the directory can't hold more than 144 filenames.

Using NoZap

After typing in NoZap, load and run it; it's a BASIC loader. The program POKEs a machine language program into the current top of BASIC memory and protects it from BASIC variables. NoZap uses memory locations 739–767, so avoid putting any ML routines there.

After you've run NoZap, the title line appears, then this prompt:

FILENAME?

Enter a filename (without quotes) from 0 to 14 characters long and press RETURN. Don't try to use a filename longer than 14 characters, as this can cause your computer to lock up. You don't need to include the version number, since NoZap adds that for you. Next, type NEW and press RETURN. From this point on, simply program as you normally would. NoZap is in charge of your SAVEs, although you may continue to use the normal SAVE command. The first time NoZap saves, it uses a version number of 01. For example, if you enter THOR as the filename, the first version will be 01THOR; the second, 02THOR; and so on. NoZap reports the disk status, but won't try to save again if there's an error.

Forced SAVEs and Toggling

Occasionally, you may want to save a new version before the next NoZap SAVE. Or you may want to turn off NoZap for awhile. To do so, use these commands:

SYS 739 (forced SAVE)
SYS 745 (toggle off and on)

Typing SYS 739 increments the version number and saves the program. NoZap resets its timer so the next SAVE will occur ten minutes later.

If you want to turn NoZap off, just enter SYS 745. This acts as a *toggle,* so if you SYS 745 again, NoZap restarts as if it had been run for the first time.

Zapping NoZap

NoZap has been written to prevent it from interfering with your programming— RUN/STOP–RESTORE *does not deactivate it*. To do that, turn the computer off, then on again, or SYS 64738.

There are also ways to trick NoZap to your advantage. For example, if you stop at 04THOR one evening, the next time you program, run NoZap and use the filename THOR again. To defeat SAVEs, open the gate on the disk drive (and remove the disk if you like). To bump the version number up, SYS 739 repeatedly until you reach the desired number. Leaving the gate open will also help you avoid saving something in memory that you don't wish to save (like the disk directory). You may have to initialize the drive (or turn it off and on) to get it to respond after this, since the drive protects itself by not repeatedly trying to operate with the gate open.

Wild Cards and Pattern Matches

Since the version numbers are at the beginning of the filename, you can list all the versions of THOR with

```
LOAD"$0:??THOR",8
LIST
```

or by using the wedge command:

```
@$0:??THOR
```

If the program name is long, you may want to use pattern matching as well. For example, versions of a filename such as THORSREVENGE could be viewed with the wedge command:

```
@$0:??THORS*
```

This is subject to the normal rules of pattern matching.

When you have a final version, you may want to do a normal SAVE of the program, using a unique name, like FINALTHOR. You can then scratch all the NoZap-saved versions of THOR with the following wedge command:

```
@S0:??THOR
```

Remember that it's usually best not to use pattern matching when scratching files so that you won't erase files accidentally.

How NoZap Works

NoZap takes advantage of the fact that many BASIC and Kernal routines are *vectored*. A vector is like a road sign that tells the computer the location of a routine. Since the vector is in RAM, it can be changed to point to your own routine, the same way a detour sign guides you when traffic is rerouted. A program that uses such a detour is called a *wedge*.

NoZap sets up a detour in the Main BASIC Loop, the part of BASIC that takes in program lines as they are entered (in direct mode). As a result, BASIC will take the NoZap detour each time you press RETURN. When you run NoZap and enter a filename, the name is placed in a filename buffer, just after the current version number. The vector at locations $302–303 (decimal 770–771), which points to the Main BASIC Loop, is altered, and one of the computer's internal timers is set to 0. It's this timer that NoZap checks as you enter each program line. The timer used is the TOD (time of day) clock at locations $DC08–DC0A (56328–56330). If the timer has not counted to ten minutes, NoZap sends the computer back to the Main BASIC Loop at $A483 (42115). This completes the NoZap detour.

Since NoZap wedges into the Main vector at $302–303, it is not compatible with programming utilities which use the same technique. You may have to experiment to find out which utilities will work with NoZap in place. Another source of conflict is programs that want to use the same section of memory.

Clock Strikes Ten

If the timer has counted far enough, NoZap continues, adding one to the version number in the filename buffer, then uses the Kernal SETNAM, SETLFS, and SAVE routines. NoZap determines which area of memory to save by looking at the pointers to the start and end of BASIC program text—$2B–2C (43–44) and $2D–2E (45–46), respectively. Then it checks the error channel and finishes the SAVE routine, returning to the Main Loop again.

The above description is brief, so use a machine language monitor to disassemble NoZap if you wish to look at all the details. In addition, the BASIC loader POKEs in two short routines. The first, which starts at location 739, sets the timer to trick NoZap into thinking the time is up. This forces an earlier SAVE.

The second routine is a NoZap pointer. Located at 745, the routine consists of a JuMP to the starting address of the NoZap initialization routine. When you run the BASIC loader, this address is placed in its correct form in addresses 746–747. This means that no matter where NoZap locates, you can toggle it on and off with SYS 745.

Customizing NoZap

After you've typed in, saved, *and tested* the BASIC loader, you may want to customize it to suit your preferences. One easy modification is to change the interval between SAVEs. While the normal value is 10 minutes, NoZap maintains a counter which allows you to use an interval of 20 minutes or more. To change the time between SAVEs to 20 minutes, for example, change the 1 in line 42 to a 2. Change it to 3 for 30 minutes, and so on. You must also increase the checksum number in line 102 by the same amount as you increase the counter value.

One side effect of changing the interval is that you must SYS 739 repeatedly to do a forced SAVE. For example, if you change the counter value to 2, you must SYS 739 twice to do a forced SAVE, and three times if the counter is set to 3. To avoid this problem, here's a simpler way to force a SAVE when the counter is set to 2 or higher:

POKE 750,1: SYS 739

It's even possible to force NoZap into starting at a version number other than 01. This is handy when you want to type in a program in several sessions. If you add the following four lines to NoZap, you can start at any version number from 00 to 99.

To use the addition, load but don't run NoZap. Add these lines and save under a new name (like NOZAPX, for extended) *before* running the new NoZap.

```
15 PRINT "RUN 200 TO ALTER THE VERSION NUMBER."
                                            :rem 215
200 INPUT "LAST VERSION NUMBER{3 SPACES}00{4 LEFT}
    ";N$:IFLEN(N$)<>2THEN200                 :rem 226
202 H$=LEFT$(N$,1):L$=RIGHT$(N$,1):IFH$<"0"ORH$>"9
    "ORL$<"0"ORL$>"9"THEN200                  :rem 80
204 POKE751,ASC(H$):POKE752,ASC(L$):NEW     :rem 17
```

To use the new version, just load and run it, and you'll be asked for the filename as usual. As before, don't include the version number. If you want to start at a number other than 01, type **RUN 200**. You'll see the prompt, **LAST VERSION NUMBER?**, which means you should enter *one less* than the number at which you wish to start. From that point on, use NoZap as you would normally. The program clears itself from BASIC memory, so if you toggle NoZap off and back on, you'll have to start at version 01 (unless you use the POKEs below or reload the extended version).

Here are the version number POKEs if you want to do it manually (*H* represents the ten's digit and *L* the one's digit):

POKE 751, ASC("*H*")
POKE 752, ASC("*L*")

If you've toggled NoZap off, for instance, and want to resume with version 35, you would

POKE 751, ASC("3")
POKE 752, ASC("4")

Remember that you're specifying the version *one less* than the number at which you want to start.

NoZap can be a lifesaver. It can take the worry out of losing files unexpectedly and let you concentrate on programming.

NoZap

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C)

```
1 REM NOZAP FOR THE 64                            :rem 207
2 POKE56,PEEK(56)-1:CLR:I=256*PEEK(56)+PEEK(55):S=
  I                                               :rem 216
4 DEFFNH(X)=INT(X/256)                             :rem 37
6 DEFFNL(X)=X-FNH(X)*256                          :rem 156
8 DATA169,16,141,10,220,96,76                     :rem 200
10 FORJ=739TO745:READK:POKEJ,K:NEXT               :rem 249
12 POKE746,PEEK(55):POKE747,PEEK(56)               :rem 87
14 PRINT "SYS 739=FORCED SAVE. SYS 745=TOGGLE."
                                                  :rem 241
16 READ A:B=B+A:IF A=256 THEN 102                 :rem 209
18 POKE I,A:I=I+1:GOTO 16                         :rem 141
20 DATA 173,236,2,73,1,141                        :rem 241
22 DATA 236,2,208,13,169,131                       :rem 90
24 DATA 141,2,3,169,164,141                        :rem 40
26 DATA 3,3,76,131,164,169                          :rem 1
28 DATA 48,141,239,2,141,240                       :rem 94
```

```
30 DATA 2,169,7,160,7,32                            :rem 149
32 DATA 30,171,32,249,171,160                       :rem 139
34 DATA 2,185,254,1,153,239                          :rem 48
36 DATA 2,240,3,200,208,245                          :rem 33
38 DATA 140,237,2,162,7,160                          :rem 43
40 DATA 7,142,2,3,140,3                              :rem 85
42 DATA 3,169,1,141,238,2:REM CHANGE THE 1 FOR A L
   ONGER INTERVAL                                   :rem 239
44 DATA 169,0,141,8,220,141                          :rem 38
46 DATA 9,220,141,10,220,173                         :rem 83
48 DATA 10,220,41,240,240,117                       :rem 129
50 DATA 206,238,2,208,233,238                       :rem 145
52 DATA 240,2,173,240,2,201                          :rem 27
54 DATA 58,208,20,169,48,141                        :rem 106
56 DATA 240,2,238,239,2,173                          :rem 49
58 DATA 239,2,201,58,208,5                            :rem 2
60 DATA 169,48,141,239,2,173                        :rem 105
62 DATA 237,2,162,239,160,2                          :rem 44
64 DATA 32,189,255,169,1,162                        :rem 109
66 DATA 8,160,0,32,186,255                            :rem 1
68 DATA 169,43,166,45,164,46                        :rem 118
70 DATA 32,216,255,169,141,32                       :rem 146
72 DATA 210,255,169,0,32,189                        :rem 101
74 DATA 255,169,15,162,8,160                        :rem 107
76 DATA 15,32,186,255,32,192                        :rem 106
78 DATA 255,162,15,32,198,255                       :rem 163
80 DATA 32,207,255,201,13,240                       :rem 133
82 DATA 6,32,210,255,56,176                          :rem 51
84 DATA 243,32,210,255,169,15                       :rem 149
86 DATA 32,195,255,32,204,255                       :rem 155
88 DATA 76,7,7,76,131,164                           :rem 222
90 DATA 18,78,79,90,65,80                           :rem 227
92 DATA 146,32,66,89,32,66                           :rem 15
94 DATA 76,65,75,69,32,76                           :rem 232
96 DATA 65,77,66,69,82,84                           :rem 239
98 DATA 13,70,73,76,69,78                           :rem 231
100 DATA 65,77,69,0,256                             :rem 112
102 IF B<>28715THENPRINT"ERROR IN DATA STATEMENTS.
    ":END                                           :rem 103
104 POKE S+32,FNL(S+210):POKES+34,FNH(S+210)
                                                    :rem 205
106 POKE S+58,FNL(S+83):POKES+60,FNH(S+83):rem 134
108 POKE S+205,FNL(S+67):POKES+206,FNH(S+67)
                                                    :rem 232
110 SYS745                                           :rem 49
```

Disk Directory Sort

N. A. Marshall

This short program can help you better organize your disks by alphabetically sorting each of your disk directories.

An alphabetized disk directory can be a timesaver, especially if you have a variety of disks. It's particularly helpful when you're looking for a filename in a long directory.

"Disk Directory Sort" is a short (35 lines) BASIC program. Type in the program and save it. To use it, first load it, then insert the 1541-format disk you wish to sort alphabetically. Type RUN, and the directory is read into memory and sorted. You will see the sort happening onscreen. Note that all deleted files are written to the end of the sort. After all files have been sorted, you're prompted to press the space bar to write the newly sorted directory (still sorted only in memory) back to disk. If you change your mind at this point, remove the disk before pressing the space bar. No damage is done, and your original directory remains intact.

Caution: The program reads the directory, alphabetizes it, and writes it back to disk. If you make any typing mistakes while entering it, the program could ruin the directories on your disks. There's a chance you would lose some programs. After entering and saving it, you should test it on a backup disk in case you incorrectly typed a line. (If you wouldn't mind the disk being run over by a lawn mower, it is okay to test this program with it.)

The program works on any size directory (up to 144 filenames are allowed on 1541-format disks). Here's a brief summary of the program routines:

Lines	Description
20–140	the sort
150–210	read in the file entries
220–290	write the directory
300–310	process the directory header
320–330	read a block
340–350	initialize the program

Disk Directory Sort

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
10 GOSUB340:GOTO150                                :rem 129
20 PRINT"{DOWN}SORTING":SK=K1:L%(K1)=K1:R%(1)=NF
                                                   :rem 176
30 L1=L%(SK):R1=R%(SK):SK=SK-1                     :rem 238
40 L2=L1:R2=R1:KE$=NS$(INT((L1+R1)/2))             :rem 116
50 KE$=MID$(KE$,31)+MID$(KE$,4,M%(INT((L1+R1)/2)))
                                                   :rem 127
60 IFMID$(NS$(L2),31)+MID$(NS$(L2),4,M%(L2))<KE$TH
   ENL2=L2+K1:GOTO60                                :rem 27
70 IFKE$<MID$(NS$(R2),31)+MID$(NS$(R2),4,M%(R2))TH
   ENR2=R2-K1:GOTO70                                :rem 61
80 IFL2>R2THEN110                                  :rem 248
90 N$=NS$(R2):H=M%(R2):NS$(R2)=NS$(L2):M%(R2)=M%(L
   2)                                               :rem 92
100 NS$(L2)=N$:M%(L2)=H:L2=L2+1:R2=R2-1:GOTO60
                                                    :rem 89
110 IFL2<R1THENSK=SK+1:L%(SK)=L2:R%(SK)=R1 :rem 23
120 R1=R2:IFL1<R1THEN40                            :rem 111
130 IFSKTHEN30                                      :rem 83
140 RETURN                                         :rem 117
150 NF=0:GOSUB300                                  :rem 228
160 GOSUB320:FORPP=1TO8:R$="":FL=0:M%(NF+1)=16:FOR
    X=1TO30:GET#5,I$                               :rem 169
170 IFI$=CHR$(160)ANDFL=0THENM%(NF+1)=X-4:FL=1
                                                   :rem 158
180 R$=R$+LEFT$(I$+C0$,1):NEXT:IFPP<>8THENGET#5,I$
    ,I$                                             :rem 70
190 X$=C0$:IFMID$(R$,1,1)=C0$THENX$=CHR$(255):PRIN
    TDD$;                                          :rem 138
200 NF=NF+1:NS$(NF)=R$+X$:PRINTMID$(R$,4,16):NEXTP
    P:IFYS<>255THEN160                             :rem 122
210 CLOSE5:GOSUB20                                  :rem 90
220 PRINT"{DOWN}PRESS SPACE BAR TO REWRITE DIRECTO
    RY"                                             :rem 62
230 GETA$:IFA$<>" "THEN230                         :rem 138
240 GOSUB300:NN=0                                  :rem 236
250 GOSUB320:FORPP=1TO8:NN=NN+1                    :rem 193
260 PRINT#5,MID$(NS$(NN),1,30);:IFMID$(NS$(NN),31)
    =CHR$(255)THENPRINTDD$;                        :rem 249
270 PRINTMID$(NS$(NN),4,16):IFPP<>8THENPRINT#5,C0$
    ;C0$;                                           :rem 25
280 NEXTPP:PRINT#15,"U2";5;0;LT;LS:IFYS<>255THEN25
    0                                              :rem 161
290 CLOSE5:END                                      :rem 87
```

```
300 OPEN5,8,5,"#":YT=18:YS=Ø:GOSUB32Ø:PRINT#15,"B-
    P";5;143:PRINTCHR$(14)                    :rem 193
31Ø PRINTRN$;:FORX=1TO24:GET#5,I$:PRINTI$;:NEXT:PR
    INTRF$:RETURN                             :rem 16Ø
32Ø PRINT#15,"U1";5;Ø;YT;YS:LT=YT:LS=YS:GET#5,T$,S
    $:YT=ASC(T$+CØ$)                           :rem 16
33Ø YS=ASC(S$+CØ$):RETURN                     :rem 25Ø
34Ø X=15Ø:DIM L%(X),M%(X),R%(X),NS$(X):K1=1:OPEN15
    ,8,15,"I":CØ$=CHR$(Ø):NF=Ø                :rem 141
35Ø DD$=":::::::::::::DELETED{+}":RN$=CHR$(18):RF
    $=CHR$(146):RETURN                        :rem 19Ø
```

Disk Defaulter

Eric Brandon

This useful utility saves typing for people who regularly use a disk drive instead of a cassette recorder. The machine language routine is in the form of an easy-to-use BASIC loader.

Faulty Default

When Commodore designed the operating system used in the 64, the designers assumed that most people would be using a cassette recorder for storage instead of the more expensive disk drive. That's why, when you type LOAD or SAVE, the computer responds by prompting PRESS PLAY ON TAPE or PRESS RECORD & PLAY ON TAPE. It *defaults* to the tape recorder.

Along the way, many 64 owners, and especially 128 owners, have opted to use a disk storage system. If you use a disk drive, though, you have to type the device number—,8—after each command (as in LOAD"*filename*",8). This can become bothersome after awhile.

"Disk Defaulter" is a short utility, written in machine language, that modifies the computer's operating system to recognize the disk drive as the default device instead of the cassette recorder. As long as the utility is activated, you no longer have to append ,8 to the LOAD, SAVE, and VERIFY commands.

To use Disk Defaulter, enter the program. When you type RUN, this BASIC loader will POKE the machine language into some free memory space and activate the utility. To turn it off (for instance, if you want to use cassette), press RUN/STOP–RESTORE. To turn it back on, type SYS 679.

To load machine language programs, you still must type LOAD "*filename*",8,1. Also, pressing SHIFT–RUN/STOP will not access the disk drive because it results in a MISSING FILENAME ERROR. But otherwise, all LOAD, SAVE, and VERIFY commands will refer to disk.

Programs that use the same area of memory as Disk Defaulter will interfere. One of these is the *PAL Assembler* for the Commodore 64. After saving and testing the original, adventurous programmers can remedy this interference with

PAL by changing the value 679 in lines 10, 1020, and 1030 to the value 703. Also, change the 188 in line 679 to 212, and the 195 in line 686 to 219.

Disk Defaulter

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
10 I=679                                          :rem 141
20 READ A:IF A=256 THEN 1000                      :rem 147
30 POKE I,A:I=I+1:GOTO 20                         :rem 130
679 DATA 169,188,141,48,3,169,2                    :rem 16
686 DATA 141,49,3,169,195,141,50                   :rem 54
693 DATA 3,169,2,141,51,3,96                      :rem 103
700 DATA 162,8,134,186,76,165,244                 :rem 100
707 DATA 162,8,134,186,76,237,245,256              :rem 53
1000 PRINT"{CLR}DISK DEFAULTER ACTIVATED" :rem 129
1010 PRINT"USE RUN/STOP-RESTORE TO DEACTIVATE"
                                                  :rem 184
1020 PRINT"TYPE SYS 679 TO REACTIVATE"              :rem 6
1030 SYS 679                                      :rem 105
```

UnNEW: Program Lifesaver

Vern Buis

If you have ever lost a BASIC program by accidentally typing NEW, then read on. This short machine language routine provides an easy means of recovering BASIC programs that have been "erased"—and it loads and executes in only ten seconds.

Sooner or later practically every programmer does it. Thinking a program has been saved, you type NEW to clear out the memory, and a split second after pressing RETURN, you wind up screaming.

But typing NEW does not really erase the program from memory. NEW just makes the computer (and the programmer) *think* the program is gone. As long as you don't start typing another program or switch off the machine, the program is still there. To get it back, all you have to do is fool the computer into remembering where in its memory the program begins and ends.

That's what "UnNEW" does. By loading and running this short machine language utility immediately after committing the grievous error, you can save your lost program, save your hours of work, and even save your sanity.

Entering UnNEW

UnNEW is listed as a BASIC loader, a BASIC program that creates a machine language program. Be sure to read the following special instructions before typing the program. The procedure is somewhat different from most and requires that certain steps be followed precisely.

First, if you are using tape instead of disk, enter line 60 as follows:

60 CLR:SAVE"UNNEW",1,1

After typing the listing, **do not run it.** Instead, save it on disk or tape with a filename such as UNNEW/BASIC. Do not use the filename UNNEW. This filename must be reserved.

Now enter RUN. The BASIC loader creates the machine language program and automatically saves it on disk or tape under the filename UNNEW. This is what you'll actually use to rescue lost programs; the BASIC loader can be set aside as a backup in case you need to create another copy. Now reset the computer by turning it off and back on.

Using UnNEW

To test UnNEW, you can load a short BASIC program and erase it with NEW. Recovering it is easy.

To load UnNEW from tape, enter LOAD"UNNEW",1,1
To load UnNEW from disk, enter LOAD"UNNEW",8,1

Remember the ,1 at the end of these commands. Either way, it loads pretty fast, because the program is short. Now, to activate UnNEW, enter

SYS 525
CLR

Incidentally, CLR means to type the keyword CLR followed by pressing the RETURN key, *not* to press the CLR/HOME key.

That's all there is to it. When you enter LIST, the BASIC program you thought was forever lost at sea is back, safe and sound.

UnNEW itself also remains in memory, but probably not for long. It's tucked away in memory which is unprotected (locations used by the input buffer and BASIC interpreter), so you'll have to load it again each time you want to use it. But unless you're either very unlucky or (shall we say) prone to inadvertent actions, UnNEW isn't something you should be needing often. If you find you often lose programs due to power failures, yanked-out power cords, or forgetfulness, see "NoZap: Automatic Program Saver" elsewhere in this book.

Why UnNEW Works

Instead of erasing the program in memory when you type NEW, the computer simply resets two key pointers in such a way that the operating system doesn't "see" that the program is still there. These pointers keep track of where in memory a BASIC program begins and ends. NEW moves the top-of-program pointer down to the bottom of BASIC memory, and the first two bytes of BASIC memory are set to zero. These first two bytes serve as a pointer to the address for the second line of BASIC code. When they are set to zero, the operating system believes that no program is in memory.

UnNEW works by skipping the first two bytes of BASIC memory (the address pointer) and the next two bytes (the BASIC line number). It scans upward for a zero byte—the end-of-line indicator. Upon finding the zero byte, the routine POKEs its address, plus one, into the second-line-of-BASIC address pointer. One of the erased pointers is thereby restored.

Next, UnNEW scans byte by byte through the BASIC memory area until it finds three consecutive zero bytes. This is the end-of-program indicator. Once it locates these zeros, the routine POKEs the address of the third zero, plus one, into the top-of-BASIC/start-of-variables pointer at locations 45–46. This completely restores the erased program.

For those who might want to relocate UnNEW to a safer memory area—to preserve it for frequent use or to combine it with other utility routines, the machine language program is written to be fully relocatable. It uses no absolute JMP or JSR instructions. The area used here was chosen to make it load easily and to minimize the danger of loading over a BASIC program.

UnNEW

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
10 I=525                                          :rem 131
20 READ A:IF A=256 THEN 40                         :rem 54
30 POKE I,A:I=I+1:GOTO 20                         :rem 130
40 POKE 43,525 AND 255:POKE 44,2                   :rem 96
50 POKE 45,578 AND 255:POKE 46,2                  :rem 109
60 CLR :SAVE "0:UNNEW",8                           :rem 79
70 REM FOR TAPE USE SAVE "UNNEW",1,1                :rem 3
60000 DATA 160,003,200,177,043,208,251,200,200,152
      ,160,000                                    :rem 251
60002 DATA 145,043,165,044,200,145,043,133,060,160
      ,000,132                                      :rem 8
60004 DATA 059,162,000,200,208,002,230,060,177,059
      ,208,245                                     :rem 26
60006 DATA 232,224,003,208,242,200,208,002,230,060
      ,132,045                                      :rem 6
60008 DATA 164,060,132,046,096,256                :rem 145
```

Function Key

Willie Brown

The function keys can be extremely useful if you know how to program them. This short utility program for the 128 allows you to define each function key and save your newly defined keyboard to tape or disk.

Turn your computer on, type some letters, and you'll see them appear on the screen. But press one of the function keys and you'll see nothing. They're mentioned almost in passing in most documentation. Often all that's said is that they *can* be programmed to perform many different functions. The question is, how do you program them?

The most common method of using the function keys is to set up a GET statement followed by an IF-THEN. The keys can be used in a program to start a game, change the border color, or almost any other function you can think of, as long as you type them in quote mode.

It would be nice, though, if they could be used outside a program, in direct (or immediate) mode. You might want f1 to LIST the program, f3 to run it, f5 to save, and so on—a collection of eight one-stroke commands.

"Function Key" lets *you* decide how you want to define the keys and use them.

Defining New Functions

After entering the program and saving it, type RUN. A short machine language program is then POKEd into memory. To turn it on, type SYS 52115.

The program is now activated. To assign a value to one of the function keys, simply type f*x*=(BASIC command), where *x* is a number from one to eight and any legal BASIC statement follows the equal sign. Press RETURN and the computer should respond with OK. If you get a SYNTAX ERROR, check the logic of the BASIC line. For example, f1=LIST defines the f1 key as LIST. Any time you press f1, LIST is printed on the screen. Of course, LIST won't be activated until you press RETURN. To activate the command without having to press RETURN, add a left arrow (the key directly above CONTROL) so that the syntax looks like f1=LIST←.

You can define all eight function keys with whatever commands you find most useful. But there are a few items to note. First, each key is limited to a maximum of 16 characters. If you exceed the limit, the extra letters will be ignored. Second, if you want a BASIC command to be executed, the last character *has to be a left arrow.* RUN/STOP–RESTORE resets the computer and eliminates the function key definitions. Simply use SYS 52115 to return to Function Key. Finally, this utility is disabled whenever you run a program. It works only in immediate mode. This allows you to use the function keys from within your program and still have your favorite commands available with one keystroke while editing the program.

Note that Function Key will not work with other programming utilities which use the same locations in memory, in other words locations 52115 and up.

Creating a Mini-Toolkit

It would be tedious to have to define all eight function keys every time you want to use this utility. You can create your own mini-toolkit with an f9 option, which allows you to save your function key definitions to tape or disk. You can then load your selected functions into memory at the beginning of a programming session.

When you have all the keys defined and want to keep them for future use, type f9=*filename*,8 (for disk) or f9=*filename*,1 (for tape), where *filename* is anything of your choice. Just don't put *filename* in quotes. If you want to save another set of function definitions, be sure to use a different filename.

To load the functions back into memory, type

LOAD "*filename*",8,1

for disk or

LOAD"*filename*",1,1

for tape. The secondary address of 1 is crucial: It tells the computer to load the program into the same area of memory it originally occupied. After the program is loaded, type NEW, then SYS 52115. The eight functions you previously saved will be available for use whenever you need them.

Function Key

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
10 I=51712:SH=INT(I/256):SL=I-SH*256          :rem 56
20 READ A:CK=CK+A:IF A=256 THEN 40            :rem 53
25 IF A<0 THEN 100                            :rem 99
30 POKE I,A:I=I+1:GOTO 20                    :rem 130
40 IFCK<>28195THENPRINT"ERROR IN DATA":STOP
                                             :rem 191
50 PRINT"USE SYS"SH*256+SL+403"TO START":END
                                             :rem 150
100 IF A<-255 THEN A=ABS(A+256)+SH:GOTO 30:rem 223
110 A=ABS(A+1)+SL:GOTO30                     :rem 116
49152 DATA 0,0,0,0,0,0                        :rem 27
49158 DATA 0,0,0,0,0,0                        :rem 33
49164 DATA 0,0,0,0,0,0                        :rem 30
49170 DATA 0,0,0,0,0,0                        :rem 27
49176 DATA 0,0,0,0,0,0                        :rem 33
49182 DATA 0,0,0,0,0,0                        :rem 30
49188 DATA 0,0,0,0,0,0                        :rem 36
49194 DATA 0,0,0,0,0,0                        :rem 33
49200 DATA 0,0,0,0,0,0                        :rem 21
49206 DATA 0,0,0,0,0,0                        :rem 27
49212 DATA 0,0,0,0,0,0                        :rem 24
49218 DATA 0,0,0,0,0,0                        :rem 30
49224 DATA 0,0,0,0,0,0                        :rem 27
49230 DATA 0,0,0,0,0,0                        :rem 24
49236 DATA 0,0,0,0,0,0                        :rem 30
49242 DATA 0,0,0,0,0,0                        :rem 27
49248 DATA 0,0,0,0,0,0                        :rem 33
49254 DATA 0,0,0,0,0,0                        :rem 30
49260 DATA 0,0,0,0,0,0                        :rem 27
49266 DATA 0,0,0,0,0,0                        :rem 33
49272 DATA 0,0,0,0,0,0                        :rem 30
49278 DATA 0,0,0,173,-1,-256                  :rem 87
49284 DATA 240,30,166,198,224,11              :rem 51
49290 DATA 176,24,168,185,-1,-256            :rem 103
49296 DATA 240,15,157,119,2,230              :rem 255
49302 DATA 198,238,-1,-256,173,-1             :rem 88
49308 DATA -256,41,15,208,3,141              :rem 246
49314 DATA -1,-256,108,-146,-257,32          :rem 180
49320 DATA 72,235,173,-1,-256,208             :rem 88
49326 DATA 34,165,157,240,30,165              :rem 51
49332 DATA 212,208,26,166,198,202            :rem 101
49338 DATA 189,119,2,201,133,144              :rem 51
49344 DATA 16,201,141,176,12,56              :rem 250
49350 DATA 233,133,10,10,10,10               :rem 174
49356 DATA 9,1,141,-1,-256,96                :rem 152
49362 DATA 166,122,189,0,2,201               :rem 199
```

```
49368 DATA 70,208,68,232,189,0                :rem 220
49374 DATA 2,201,49,144,60,201                :rem 197
49380 DATA 57,176,59,41,15,168                :rem 224
49386 DATA 185,-135,-257,168,232,189           :rem 11
49392 DATA 0,2,201,61,240,5                    :rem 40
49398 DATA 162,11,108,0,3,232                 :rem 149
49404 DATA 189,0,2,201,13,240                 :rem 140
49410 DATA 15,201,95,208,2,169                :rem 203
49416 DATA 13,153,-1,-256,200,152              :rem 77
49422 DATA 41,15,208,233,169,0                :rem 200
49428 DATA 153,-1,-256,160,107,32              :rem 86
49434 DATA 47,241,108,2,3,108                 :rem 151
49440 DATA -144,-257,208,251,232,189          :rem 247
49446 DATA 0,2,201,61,208,202                 :rem 139
49452 DATA 232,138,72,160,0,189                 :rem 3
49458 DATA 0,2,201,44,240,8                    :rem 47
49464 DATA 232,200,192,15,208,243              :rem 95
49470 DATA 240,182,192,0,208,4                :rem 201
49476 DATA 162,8,208,176,232,189               :rem 70
49482 DATA 0,2,201,49,240,8                    :rem 49
49488 DATA 201,56,240,4,162,9                 :rem 161
49494 DATA 208,160,41,15,170,152               :rem 48
49500 DATA 72,160,0,138,32,186                :rem 200
49506 DATA 255,104,168,104,170,152            :rem 147
49512 DATA 160,2,32,189,255,162                 :rem 0
49518 DATA -1,134,251,169,-256,133            :rem 144
49524 DATA 252,160,-257,169,251,162           :rem 202
49530 DATA -227,32,216,255,169,13              :rem 97
49536 DATA 32,210,255,76,116,164               :rem 53
49542 DATA 0,1,65,17,81,33                      :rem 1
49548 DATA 97,49,113,124,165,49                :rem 22
49554 DATA 234,162,-256,173,5,3                 :rem 0
49560 DATA 201,-256,240,17,141,-145           :rem 182
49566 DATA -257,173,4,3,141,-144               :rem 45
49572 DATA -257,169,-211,141,4,3               :rem 42
49578 DATA 142,5,3,173,21,3                    :rem 55
49584 DATA 201,-256,240,19,141,-147           :rem 192
49590 DATA -257,173,20,3,141,-146              :rem 90
49596 DATA -257,169,-130,120,141,20           :rem 190
49602 DATA 3,142,21,3,88,173                  :rem 102
49608 DATA 144,2,201,-256,240,19               :rem 41
49614 DATA 141,-170,-256,173,143,2            :rem 136
49620 DATA 141,-169,-256,169,-168,120          :rem 39
49626 DATA 141,143,2,142,144,2                :rem 195
49632 DATA 88,96,256                          :rem 246
```

Triple 64

Feemen Ng

This seven-line program creates three independent 12K blocks which can be accessed very simply. An excellent tool for program development and comparison.

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

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

Accessing Three Computers

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

Techniques and Applications

The most obvious use of Triple 64 is to partition the computer to hold three BASIC programs. These could be games, utilities, or applications—or any combination. And switching between them involves only a SYS and a single keypress. Each work area holds up to 12288 bytes, space enough for a fairly sophisticated program.

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

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

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

Triple 64

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
10 FORY=40004TO40071:READA:POKEY,A:NEXT     :rem 180
20 FORY=14336TO14338:POKEY,0:NEXT            :rem 29
30 FORY=26624TO26626:POKEY,0:NEXT:NEW        :rem 72
40 DATA174,125,156,165,45,157,129,156,165,46,157,1
   32,156,32,228,255,41,15,240                :rem 19
50 DATA249,201,4,176,245,170,142,125,156,189,125,1
   56,133,44,189,126,156,133,56               :rem 71
60 DATA189,129,156,133,45,133,47,133,49,189,132,15
   6,133,46,133,48,133,50,96,1                :rem 24
70 DATA8,56,104,152,3,3,3,8,56,104           :rem 174
```

Freeze

Dan Carmichael

Freezing a BASIC program, stopping it in midframe, is a handy feature, especially in game programs. Players get exhausted, want to answer the telephone, or make a sandwich, but don't want to give up that high score. "Freeze" lets you stop and start programs with single keypresses.

It's happened. You're playing a fast-action arcade game, and your hand is cramped from being too tightly wrapped around the joystick. Or your back is giving you spasms again. Or the phone rings and you just have to answer it. But you've got the highest score ever, and if you get up, the game will continue. Unfortunately, the joystick can't run itself, and you'll lose the game.

If you've placed "Freeze" in memory, however, you can stop the program at any time by pressing one key. Nothing will be lost; the program simply freezes. Anything on the screen still shows; it just doesn't move. Hitting another key unfreezes the program and restarts it. You can continue with the program from where you left off.

Freeze Keys

Type in Freeze and save it to tape or disk. The "Automatic Proofreader" in Appendix C makes it simple to enter the program correctly the first time.

After loading and running the program, you'll see a display list. You can customize Freeze by selecting your own key combination for freeze and unfreeze. If you want to use the default keys, just hit RETURN twice. The f1 key then freezes the action, and the f3 key restarts the program. To choose your own keys, enter the appropriate number before hitting RETURN.

The SYS command to access the routine also shows on the screen. Whenever you want to use Freeze, just enter SYS 679 in either direct mode or as a program line within your own program or game. If you use the last method, make sure that Freeze has been loaded into memory before you try to call it.

Once you've selected the two control keys, try the freeze function. Load and run a BASIC program. Let it run a bit, then hit the freeze key (f1 if you chose the default setting). The program immediately pauses. Press the unfreeze key (f3 if the default was used) to restart the program. That's it.

Interrupting Danger

Freeze uses a machine language interrupt by calling the IRQ interrupt vectors at $314–$315 (788–789 decimal). Because of this, if your program also uses interrupts, Freeze may not work. Programs which use machine language in other ways should still be able to access Freeze; it's only interrupts that interfere. Any completely BASIC program can call this routine. We've used this program at COMPUTE! to freeze programs so that we can take photographs of the monitor screen. We've had difficulties with only a few, and all of them used machine language interrupts.

Freeze

For mistake-proof program entry, be sure to use "Automatic Proofreader" (Appendix C).

```
10 FORA=679TO714:READB:POKEA,B:NEXT         :rem 212
20 PRINT"{CLR}{WHT}{DOWN}{15 RIGHT}64 FREEZE"
                                            :rem 186
31 PRINT"{YEL}{DOWN}KEY ASSIGNMENTS:":PRINT"{CYN}
   {DOWN}F1= 4{4 SPACES}F3= 5{4 SPACES}F5= 6
   {3 SPACES}F7= 3{6 SPACES}"               :rem 188
32 PRINT"{DOWN}£ = 48{3 SPACES}= = 53{3 SPACES}<
   {SPACE}= 47{3 SPACES}> = 44"             :rem 245
33 PRINT"{DOWN}← = 57{3 SPACES}↑ = 54{3 SPACES}+ =
    40{3 SPACES}- = 43"                     :rem 241
34 PRINT"{DOWN}? = 55{3 SPACES}CRSR{5 SPACES}CRSR"
                                            :rem 163
35 PRINT"{9 SPACES}UP = 7{3 SPACES}RIGHT = 2"
                                             :rem 63
36 PRINT"{DOWN}ENTER THE KEY YOU WISH TO FREEZE TH
   E C64":PRINT"{UP}WITH (SEE TABLE)"        :rem 43
40 INPUT"{3 RIGHT}4{3 LEFT}";K1:POKE715,K1:rem 255
45 PRINT"{DOWN}ENTER THE KEY YOU WISH TO UNFREEZE
   {SPACE}THE":PRINT"C64 WITH (SEE TABLE)" :rem 61
50 INPUT"{3 RIGHT}5{3 LEFT}";K2:POKE716,K2  :rem 4
60 PRINT"{DOWN} TO START PROGRAM{2 SPACES}* SYS679
    *[E7]"                                   :rem 36
100 DATA120,169,180,141, 20, 3, 169, 2     :rem 168
110 DATA141,21,3,88, 96, 165, 197, 205     :rem 191
120 DATA203, 2, 240, 3, 76, 49, 234, 32     :rem 73
130 DATA159,255,165,197,205,204, 2, 240     :rem 79
140 DATA243,76,190,2,234, 234, 234, 234     :rem 25
```

TurboDisk: High-Speed 1541 Disk Loader

Don Lewis

If you are frustrated by your slow 1541 disk drive, here is the solution. "TurboDisk" improves the speed of the 1541 by as much as 300 percent.

If you've ever used a really fast disk drive, you know that the Commodore 1541 drive leaves something to be desired—namely, speed. True, it's much faster than a normal Datassette, but it's still annoyingly slow compared with other floppy disk drives with high-speed parallel interfaces.

Now there's a stunning solution: "TurboDisk." Once you start using TurboDisk, you'll wonder how you got along without it. It turbocharges the loading process by a factor of three times or more. In fact, the longer the program, the more improvement you'll see!

TurboDisk requires no modifications to your disk drive or computer. It loads programs saved in the usual manner; no special Turbosave is required. It works with most BASIC and machine language programs, including the DOS Wedge. It doesn't compromise reliability, and you can switch it on or off at any time by typing a single command.

If you're still skeptical, give TurboDisk a trial—it delivers what it promises.

Preparing TurboDisk

You'll need to type in two programs to prepare TurboDisk: a BASIC program that creates a machine language file on disk (the actual TurboDisk utility), and a short two-line BASIC loader that calls up and activates TurboDisk.

Program 1 is the BASIC program that creates TurboDisk. Notice all the numbers in DATA statements; these represent the machine language portion of the utility. Be extra careful when typing these lines. We recommend using the "Automatic Proofreader" to prevent as many errors as possible (see Appendix C).

Save Program 1 on disk before running it for the first time. That way, if an error causes your computer to lock up, you can switch it off to clear the memory, reload the program, and search for the typing mistake. Otherwise you could lose all of your typing effort.

When Program 1 runs, it prints the message INSERT DISK AND HIT RETURN WHEN READY. Insert a formatted program disk and press RETURN. Program 1 creates a file on the disk with the name TURBODISK.OBJ and then prints the message, TURBODISK.OBJ CREATED. You'll probably want copies of TurboDisk on all of your program disks, so rerun the program as many times as necessary.

Program 1 will print an error message if it detects a disk error or a typing mistake in the DATA statements. In addition, the partially written TURBODISK.OBJ file will be scratched from the disk if an error is detected in the DATA.

Finally, you must type in Program 2 and save it on all your program disks with the filename TURBODISK. To load and run TurboDisk, all you have to do is enter **LOAD TURBODISK,8** and **RUN**. The short loader will call TURBODISK.OBJ off the disk, place it safely in high memory, and activate it automatically.

Turbocharged Loads

Once TurboDisk is activated, no special commands are necessary. Simply type **LOAD *"filename"*,8** or **LOAD *"filename"*,8,1** as usual. You'll be amazed at the difference.

One thing you will notice immediately is that the red light on the disk drive doesn't come on at all during a Turboload. Don't panic; this is normal. It's also normal for the screen to blank out as TurboDisk works. When the program is loaded, the screen reappears unaltered.

You may occasionally find it necessary to deactivate TurboDisk and use a normal LOAD instead. For example, 1541 disk drives are prone to head alignment problems, so if you have a disk formatted on a drive other than your own, you may find that your drive has difficulty loading programs from it. Since the Turboload routine gives up more easily on difficult LOADs, you may have to switch to the more forgiving standard LOAD to get the program into your computer. You can switch off TurboDisk at any time without erasing it from memory by entering SYS 49155. To reactivate TurboDisk, enter SYS 49152.

You'll also find it necessary to use the SYS to reactivate TurboDisk after pressing RUN/STOP–RESTORE. Using that key combination to reset the computer effectively disconnects TurboDisk.

There are a few cautions to observe. TurboDisk resides in the 4K block of free memory starting at address 49152 (hex $C000), so it's completely safe from BASIC. However, many machine language programs or subroutines also use this memory space and may overwrite TurboDisk. Don't attempt to use TurboDisk to load any program which occupies locations 49152–50431 ($C000–$C4FF).

TurboDisk speeds up LOADs—even LOADs from within programs, as are common in multipart programs—but it can't speed up SAVEs or VERIFYs. It also doesn't affect the speed of disk file handling with OPEN, PRINT#, GET#, and so forth. It's not compatible with certain features of some programs, such as saving text files with the *SpeedScript* word processor (available from COMPUTE! Publications), although you can use TurboDisk to load *SpeedScript* in the first place. It also may not work with some commercial software.

How TurboDisk Works

The machine language for TurboDisk is unusual in that only half of it works within your computer—the rest is actually executed within the 1541 drive itself. Unlike disk drives for most other computers, Commodore's are *intelligent* units, containing their own microprocessors, RAM, and ROM. This means that they can be programmed for special effects, like Turboloading.

During the brief delay you notice between the time you enter the LOAD command with TurboDisk and the time the drive starts spinning, 444 bytes of machine language code are transferred from the computer to the drive's RAM. This is the portion in the second set of DATA statements in Program 1. It's stored in locations 49664–50107 ($C200–$C3BB). This required transfer of data before each Turboload adds a certain amount of overhead time, which explains why TurboDisk gives less speed improvement for short programs.

TurboDisk operates by changing the ILOAD vector at locations 816–817 ($330–$331) to point to itself, bypassing the normal LOAD routines in ROM. (These locations are reset to their normal values during the RUN/STOP–RESTORE sequence, which explains why the program must be reactivated after that key combination is pressed.) TurboDisk first checks to see whether a disk directory (LOAD "$",8) or a VERIFY was requested. In either of these cases, control is returned to the ROM routines for normal processing. If a program LOAD was requested, the routine adds the filename to the code for the disk drive portion, then transfers that data to the drive's memory.

The portion of TurboDisk in the disk drive uses routines in the drive's ROM to locate the desired program and read it from the disk, sector by sector. To improve speed, drive ROM routines like the one that turns on the red light are omitted, and only the essential ones are used. The 256 bytes of data from each disk sector are transferred two bits at time to a 256-byte buffer within the computer. This buffer is at locations 50176–50431 ($C400–$C4FF).

TurboDisk machine language in the computer reads the incoming data from the serial port's DATA and CLK lines, instead of just the DATA line as in normal serial data transfers. Thus, TurboDisk temporarily converts your serial drive into a two-bit parallel drive. When the entire 256 bytes from a disk sector have been transferred into the computer's buffer, data from the buffer is added to the program in memory while the drive is reading the next sector from the disk.

Just How Fast Is It?

Despite a few limitations, TurboDisk is one of the most valuable general-purpose utilities a disk user can own. To discover exactly how fast it is, we ran tests with some programs recently published in COMPUTE! publications. The test results, shown below, demonstrate how TurboDisk yields the most improvement with medium to long programs. (Results with different disk drives may vary.)

Program	Blocks	Normal LOAD	Turboload	Factor
Acrobat	31	21 sec	7 sec	3.0
Space Caverns	17	13 sec	5 sec	2.6
64 Paintbox	45	31 sec	9 sec	3.4
Unicopy 64	8	7 sec	5 sec	1.4
SpeedScript	25	18 sec	6 sec	3.0
SpeedScript source code	122	75 sec	17 sec	4.4

Program 1. 64 TurboDisk Creator

```
100 PRINT"{CLR}"TAB(206)"{WHT}TURBODISK PROGRAM GE
    NERATOR":PRINT:PRINT                       :rem 2
110 PRINT"{CYN}INSERT DISK AND HIT {RVS} RETURN
    {OFF} WHEN READY":PRINT:PRINT             :rem 115
120 GET A$:IF A$<>CHR$(13) THEN 120           :rem 248
130 OPEN 2,8,2,"TURBODISK.OBJ,P,W":GOSUB 1000
                                              :rem 100
140 PRINT#2,CHR$(0)CHR$(192);                  :rem 78
150 FOR I=0 TO 435:READ A:CK=CK+A:PRINT#2,CHR$(A);
    :NEXT I                                   :rem 224
160 IF A<>96 OR CK<>55976 THEN PRINT"{RVS}ERROR IN
     DATA LINES 49152-49584":GOTO 300          :rem 23
170 FOR I=0 TO 75:PRINT#2,CHR$(234);:NEXT I
                                              :rem 116
180 CK=0:FOR I=0 TO 443:READ A:CK=CK+A:PRINT#2,CHR
    $(A);:NEXT I                               :rem 23
190 IF A<>160 OR CK<>45825 THEN PRINT"{RVS}ERROR I
    N DATA LINES 49664-50102":GOTO300          :rem 44
200 CLOSE 15:CLOSE 2:PRINT TAB(9)"{7}TURBODISK.OBJ
     CREATED":PRINT:PRINT TAB(10);             :rem 96
210 INPUT "ANOTHER COPY (Y/N)";A$:IF A$<>"Y" THEN
    {SPACE}END                                :rem 197
220 RUN                                       :rem 137
300 CLOSE 2:CLOSE 15:OPEN 15,8,15,"S0:TURBODISK.OB
    J":CLOSE 15:END                            :rem 45
1000 CLOSE 15:OPEN 15,8,15:INPUT#15,E,E$,T,S:IF E=
     0 THEN RETURN                             :rem 71
1010 PRINT"DISK ERROR"E": "E$;T;S             :rem 145
1020 CLOSE 15:OPEN 15,8,15,"I0:":CLOSE 15:END
                                              :rem 177
49100 REM ** 64 TURBODISK ML                  :rem 240
49152 DATA 24,144,24,169,165,141               :rem 50
49158 DATA 48,3,169,244,141,49                :rem 221
49164 DATA 3,160,0,185,41,192                 :rem 151
49170 DATA 240,6,32,22,231,200                :rem 184
49176 DATA 208,245,96,169,84,141               :rem 71
49182 DATA 48,3,169,192,141,49                :rem 220
```

```
49188 DATA 3,160,21,208,230,13          :rem 196
49194 DATA 84,85,82,66,79,68            :rem 142
49200 DATA 73,83,75,32,68,73            :rem 115
49206 DATA 83,65,66,76,69,68            :rem 135
49212 DATA 13,0,13,84,85,82              :rem 51
49218 DATA 66,79,68,73,83,75            :rem 137
49224 DATA 32,65,67,84,73,86            :rem 124
49230 DATA 65,84,69,68,13,0              :rem 64
49236 DATA 133,147,165,147,208,30       :rem 102
49242 DATA 160,0,177,187,201,36         :rem 253
49248 DATA 240,22,162,16,169,160         :rem 50
49254 DATA 157,172,195,202,16,250       :rem 102
49260 DATA 177,187,153,172,195,200      :rem 158
49266 DATA 196,183,144,246,176,11       :rem 114
49272 DATA 165,147,76,165,244,77         :rem 71
49278 DATA 45,87,0,0,32,169              :rem 65
49284 DATA 16,133,255,169,0,133           :rem 2
49290 DATA 251,169,194,133,252,169      :rem 164
49296 DATA 0,133,253,169,5,133          :rem 209
49302 DATA 254,165,186,32,177,255       :rem 110
49308 DATA 169,111,32,147,255,165       :rem 106
49314 DATA 253,164,254,141,128,192      :rem 152
49320 DATA 140,129,192,160,0,185         :rem 42
49326 DATA 125,192,32,168,255,200       :rem 100
49332 DATA 192,6,208,245,160,0          :rem 201
49338 DATA 177,251,32,168,255,200       :rem 106
49344 DATA 192,32,144,246,165,251       :rem 104
49350 DATA 105,31,133,251,165,252        :rem 89
49356 DATA 105,0,133,252,165,253         :rem 45
49362 DATA 105,32,133,253,165,254        :rem 97
49368 DATA 105,0,133,254,32,174         :rem 253
49374 DATA 255,198,255,208,180,165      :rem 170
49380 DATA 186,32,177,255,169,111       :rem 112
49386 DATA 32,147,255,169,85,32          :rem 19
49392 DATA 168,255,169,67,32,168         :rem 77
49398 DATA 255,32,174,255,120,169       :rem 115
49404 DATA 11,141,17,208,32,125         :rem 242
49410 DATA 193,44,0,196,48,83           :rem 164
49416 DATA 164,195,166,196,165,185      :rem 175
49422 DATA 240,6,172,2,196,174          :rem 207
49428 DATA 3,196,132,174,134,175         :rem 59
49434 DATA 162,4,173,0,196,240          :rem 204
49440 DATA 21,32,101,193,32,125         :rem 239
49446 DATA 193,173,0,196,48,50          :rem 218
49452 DATA 240,6,32,99,193,24           :rem 164
49458 DATA 144,240,162,2,160,0          :rem 197
49464 DATA 189,0,196,145,174,200         :rem 59
49470 DATA 232,236,1,196,144,244         :rem 52
49476 DATA 189,0,196,145,174,200         :rem 62
49482 DATA 32,112,193,24,72,169           :rem 6
```

```
49488 DATA 27,141,17,2Ø8,1Ø4,166         :rem 58
49494 DATA 174,164,175,88,96,169         :rem 87
495ØØ DATA 4,44,169,Ø,56,176             :rem 1Ø9
495Ø6 DATA 235,162,2,16Ø,Ø,189           :rem 2Ø4
49512 DATA Ø,196,145,174,2ØØ,232         :rem 42
49518 DATA 2Ø8,247,24,152,1Ø1,174        :rem 1ØØ
49524 DATA 133,174,165,175,1Ø5,Ø         :rem 48
4953Ø DATA 133,175,96,16Ø,Ø,173          :rem Ø
49536 DATA Ø,221,48,251,169,23           :rem 2Ø7
49542 DATA 141,Ø,221,173,Ø,221           :rem 185
49548 DATA 16,251,169,7,141,Ø            :rem 16Ø
49554 DATA 221,162,4,2Ø2,234,2Ø8         :rem 42
4956Ø DATA 252,162,4,173,Ø,221           :rem 196
49566 DATA 1Ø,8,1Ø,38,149,4Ø             :rem 1Ø7
49572 DATA 38,149,2Ø2,2Ø8,242,165        :rem 1Ø8
49578 DATA 149,73,255,153,Ø,196          :rem 2Ø
49584 DATA 2ØØ,2Ø8,2Ø4,96                :rem 221
496ØØ REM ** 1541 TURBODISK ML           :rem 86
49664 DATA 32,66,2Ø8,12Ø,169,21          :rem 4
4967Ø DATA 141,7,28,169,18,16Ø           :rem 215
49676 DATA 1,141,Ø,3,14Ø,1               :rem 246
49682 DATA 3,32,2Ø5,5,169,3              :rem 58
49688 DATA 133,6Ø,162,Ø,134,75           :rem 211
49694 DATA 24Ø,43,16Ø,Ø,177,59           :rem 215
497ØØ DATA 41,191,2Ø1,13Ø,2Ø8,25         :rem 34
497Ø6 DATA 2ØØ,2ØØ,2ØØ,185,169,6         :rem 42
49712 DATA 2Ø1,42,24Ø,66,2Ø1,63          :rem 244
49718 DATA 24Ø,4,2Ø9,59,2Ø8,7            :rem 167
49724 DATA 2ØØ,192,18,24Ø,53,2Ø8         :rem 47
4973Ø DATA 234,23Ø,75,166,75,224         :rem 56
49736 DATA 8,24Ø,7,189,11Ø,5             :rem 113
49742 DATA 133,59,2Ø8,2Ø6,173,Ø          :rem 2
49748 DATA 3,24Ø,6,172,1,3               :rem 3
49754 DATA 76,19,5,169,255,141           :rem 225
4976Ø DATA Ø,3,32,15Ø,5,169              :rem 51
49766 DATA 58,141,7,28,88,76             :rem 135
49772 DATA 69,217,2,34,66,98             :rem 13Ø
49778 DATA 13Ø,162,194,226,23Ø,59        :rem 113
49784 DATA 16Ø,Ø,177,59,141,Ø            :rem 16Ø
4979Ø DATA 3,2ØØ,177,59,141,1            :rem 156
49796 DATA 3,32,2Ø5,5,32,15Ø             :rem 1Ø4
498Ø2 DATA 5,173,Ø,3,2Ø8,245             :rem 1Ø1
498Ø8 DATA 169,58,141,7,28,96            :rem 182
49814 DATA 16Ø,Ø,185,Ø,3,133             :rem 95
4982Ø DATA 133,169,2,141,Ø,24            :rem 146
49826 DATA 173,Ø,24,41,4,24Ø             :rem 99
49832 DATA 249,169,Ø,141,Ø,24            :rem 155
49838 DATA 162,4,169,Ø,6,133             :rem 112
49844 DATA 42,1Ø,6,133,42,1Ø             :rem 94
4985Ø DATA 141,Ø,24,2Ø2,2Ø8,24Ø          :rem 24Ø
```

```
49856 DATA 72,104,72,104,169,0                :rem 210
49862 DATA 141,0,24,200,208,204               :rem 241
49868 DATA 96,172,1,3,132,7                   :rem 67
49874 DATA 173,0,3,197,6,8                    :rem 19
49880 DATA 133,6,40,240,16,169                :rem 209
49886 DATA 176,133,0,88,36,0                  :rem 119
49892 DATA 48,252,120,165,0,201               :rem 253
49898 DATA 1,208,78,169,238,141               :rem 25
49904 DATA 12,28,169,6,133,50                 :rem 159
49910 DATA 169,0,133,51,133,48                :rem 205
49916 DATA 169,3,133,49,32,82                 :rem 169
49922 DATA 6,80,254,184,173,1                 :rem 162
49928 DATA 28,153,0,3,200,208                 :rem 152
49934 DATA 244,160,186,80,254,184             :rem 115
49940 DATA 173,1,28,153,0,1                   :rem 48
49946 DATA 200,208,244,32,224,248             :rem 103
49952 DATA 165,56,197,71,240,4                :rem 221
49958 DATA 169,34,208,20,32,233               :rem 9
49964 DATA 245,197,58,240,4,169               :rem 25
49970 DATA 35,208,9,169,236,141               :rem 15
49976 DATA 12,28,96,24,105,24                 :rem 167
49982 DATA 133,68,169,255,141,0               :rem 13
49988 DATA 3,32,150,5,169,58                  :rem 124
49994 DATA 141,7,28,165,68,76                 :rem 183
50000 DATA 200,193,32,88,6,76                 :rem 146
50006 DATA 148,6,165,18,133,22                :rem 196
50012 DATA 165,19,133,23,165,6                :rem 194
50018 DATA 133,24,165,7,133,25                :rem 194
50024 DATA 169,0,69,22,69,23                  :rem 104
50030 DATA 69,24,69,25,133,26                 :rem 152
50036 DATA 32,52,249,162,90,32                :rem 198
50042 DATA 148,6,80,254,184,173               :rem 255
50048 DATA 1,28,217,36,0,208                  :rem 95
50054 DATA 6,200,192,8,208,240                :rem 192
50060 DATA 96,202,208,233,169,32              :rem 43
50066 DATA 208,170,169,208,141,5              :rem 46
50072 DATA 24,169,33,44,5,24                  :rem 99
50078 DATA 16,158,44,0,28,48                  :rem 109
50084 DATA 246,173,1,28,184,160               :rem 253
50090 DATA 0,96,160,160,160,160               :rem 239
50096 DATA 160,160,160,160,160,160            :rem 132
50102 DATA 160,160,160,160,160,160            :rem 120
```

Program 2. 64 TurboDisk Loader

```
10 IF A=0 THEN A=1:LOAD "TURBODISK.OBJ",8,1
                                              :rem 155
20 SYS 49152:NEW                              :rem 138
```

Appendices

A

A Beginner's Guide to Typing In Programs

What Is a Program?

A computer cannot perform any task by itself. Like a car without gas, a computer has *potential,* but without a program, it isn't going anywhere. Most of the programs published in this book are written in a computer language called BASIC. BASIC is easy to learn and is built into the Commodore 128 in both 128 and 64 modes.

BASIC Programs

This book includes programs for the Commodore 128 and 64. If you have a 128, note that these programs work only in 64 mode. To enter 64 mode, turn on the computer and type GO 64.

Computers can be picky. Unlike the English language, which is full of ambiguities, BASIC usually has only one right way of stating something. Every letter, character, or number is significant. A common mistake is substituting a letter such as *O* for the numeral *0*, a lowercase *l* for the numeral *1*, or an uppercase *B* for the numeral *8*. Also, you must be sure to enter all punctuation marks, such as colons and commas, just as they appear in the book. Spacing can be important. To be safe, type in the listings *exactly* as they appear.

Braces and Special Characters

The exception to this typing rule is when you see the braces, such as {DOWN}. Anything within a set of braces is a special character or characters that cannot easily be listed on a printer. When you come across such a special statement, refer to "How to Type In Programs" (Appendix B).

About DATA Statements

Some programs contain a section or sections of DATA statements. These lines provide information needed by the program. Some DATA statements contain actual programs (in machine language), while others may contain graphics codes. These lines are especially sensitive to errors.

If a single number in any one DATA statement is mistyped, your machine could lock up, or crash. The keyboard and RUN/STOP key may seem dead, and the screen may go blank. But don't panic. No damage has been done. To regain control, turn off your computer and then turn it back on. This will erase whatever program was in memory, *so always save a copy of your program before you run it.* If your computer crashes, you can load the program and look for your mistake.

Sometimes a mistyped DATA statement will cause an error message when the program is run. The error message may refer to the program line that READs the data. *However, the error is still most likely in the DATA statements.*

Get to Know Your Machine

You should familiarize yourself with your computer before attempting to type in a program. Learn the statements you use to store and retrieve programs from tape or disk. You'll want to save a copy of your program so that you won't have to type it in every time you want to use it. Learn to use your machine's editing functions. How do you change a line if you made a mistake? You can always retype the line, but you should at least know how to delete characters. Do you know how to enter reverse-video, lowercase, and control characters? It's all explained in your manual.

In order to insure accurate entry of each program line, we have included a checksum program. Please read "Automatic Proofreader" (Appendix C) before typing in any of the programs in this book.

A Quick Review

1. Type in the program a line at a time in order. Press RETURN at the end of each line. Use the INST/DEL key to correct mistakes.
2. Check the line you've typed against the line in the book. You can check the entire program again if you get an error when you run the program.

B

How to Type In Programs

Many of the programs in this book contain special control characters (cursor controls, color keys, reverse video, and so on). To make it easy to know exactly what to type when entering one of these programs into your computer, we have established the following listing conventions.

Generally, program listings will contain words within braces which spell out any special characters: {DOWN} would mean to press the cursor-down key, and {5 SPACES} would mean to press the space bar five times.

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

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

Rarely, you'll see a solitary letter of the alphabet enclosed in braces. These characters can be entered by holding down the CONTROL key while typing the letter in the braces. For example, {A} would indicate that you should press CONTROL-A.

Quote Mode

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

Once you press the quote (the double quote, SHIFT-2), you are in the quote mode. If you type something and then try to change it by moving the cursor left, you'll only get a bunch of reverse-video lines. These are the symbols for cursor left. The only editing key that isn't affected by quote mode is the INST/DEL key; you can still use INST/DEL to back up and edit the line. Once you type another quote, you are out of quote mode.

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

In order to insure accurate entry of each program line, we have included a checksum program. Please read "Automatic Proofreader" (Appendix C) before typing in any of the programs in this book.

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

When You Read:	Press:	See:	When You Read:	Press:	See:
{CLR}	SHIFT CLR/HOME		Ɛ1Ǝ	COMMODORE 1	
{HOME}	CLR/HOME		Ɛ2Ǝ	COMMODORE 2	
{UP}	SHIFT ↑CRSR↓		Ɛ3Ǝ	COMMODORE 3	
{DOWN}	↑CRSR↓		Ɛ4Ǝ	COMMODORE 4	
{LEFT}	SHIFT ←CRSR→		Ɛ5Ǝ	COMMODORE 5	
{RIGHT}	←CRSR→		Ɛ6Ǝ	COMMODORE 6	
{RVS}	CTRL 9		Ɛ7Ǝ	COMMODORE 7	
{OFF}	CTRL 0		Ɛ8Ǝ	COMMODORE 8	
{BLK}	CTRL 1		{F1}	f1	
{WHT}	CTRL 2		{F2}	SHIFT f1	
{RED}	CTRL 3		{F3}	f3	
{CYN}	CTRL 4		{F4}	SHIFT f3	
{PUR}	CTRL 5		{F5}	f5	
{GRN}	CTRL 6		{F6}	SHIFT f5	
{BLU}	CTRL 7		{F7}	f7	
{YEL}	CTRL 8		{F8}	SHIFT f7	
			←	←	
			↑	SHIFT ↑	

C

Automatic Proofreader

Charles Brannon

"Automatic Proofreader" will help you type in program listings without typing mistakes. It is a short error-checking program that hides itself in memory. When activated, it lets you know immediately after you type a line from a program listing if you have made a mistake. Please read these instructions carefully before typing any programs in this book.

Preparing the Proofreader

1. Using the listing below, type in the Proofreader. Be very careful when entering the DATA statements—don't type an *l* instead of a *1*, an *O* instead of a *0*, extra commas, and so on.
2. Save the Proofreader on tape or disk at least twice *before running it for the first time.* This is very important because the Proofreader erases part of itself when you first type RUN.
3. After the Proofreader is saved, type RUN. It will check itself for typing errors in the DATA statements and warn you if there's a mistake. Correct any errors and save the corrected version. Keep a copy in a safe place—you'll need it again and again, every time you enter a program from this book, *COMPUTE!'s Gazette,* or *COMPUTE!* magazine.
4. When a correct version of the Proofreader is run, it activates itself. You are now ready to enter a program listing. If you press RUN/STOP–RESTORE, the Proofreader is disabled. To reactivate it, just type the command SYS 886 and press RETURN.

Using the Proofreader

Some of the listings in this book have a *checksum number* appended to the end of each line, for example, **:rem 123. Don't enter this statement when typing in a program.** It is just for your information. The rem makes the number harmless if someone does type it in. It will, however, use up memory if you enter it, and it will confuse the Proofreader, even if you entered the rest of the line correctly.

When you type in a line from a program listing and press RETURN, the Proofreader displays a number at the top of your screen. *This checksum number must match the checksum number in the printed listing.* If it doesn't, it means you typed the line differently from the way it is listed. Immediately recheck your typing. Remember, don't type the rem statement with the checksum number; it is published only so you can check it against the number which appears on your screen.

The Proofreader is not picky about spaces. It will not notice extra spaces or missing ones. This is for your convenience, since spacing is generally not important. But *occasionally proper spacing is important, so be extra careful with spaces, especially within quote marks.*

Due to the nature of the checksums, the Proofreader will not catch all errors. Since 1 + 3 + 5 = 3 + 1 + 5, the Proofreader cannot catch errors of transposition. Thus, the Proofreader will not notice if you type GOTO 385 where you mean GOTO 835. In fact, you could type in the line in any order and the Proofreader wouldn't notice. The Proofreader should help you catch most typing mistakes, but keep this in mind if a program that checks out with the Proofreader still seems to have errors.

There's another thing to watch out for: If you enter a line by using abbreviations for commands, the checksum will not match up. But there is a way to make the Proofreader check the line. After entering the line, LIST it. This eliminates the abbreviations. Then move the cursor up to the line and press RETURN. It should now match the checksum. You can check whole groups of lines this way.

Special Tape SAVE Instructions

When you're through typing in a listing, you must disable the Proofreader before saving the program on tape. Disable the Proofreader by pressing RUN/STOP–RESTORE (hold down the RUN/STOP key and sharply hit the RESTORE key). This procedure is not necessary for disk SAVEs, *but you must disable the Proofreader in this way before a tape SAVE.*

SAVE to tape erases the Proofreader from memory, so you'll have to load and run it again if you want to type another listing. SAVE to disk does not erase the Proofreader.

Hidden Perils

Tape users have an additional problem to overcome. What if you type in a program in several sittings? The next day, you come to your computer, load and run the Proofreader, then try to load the partially completed program so that you can add to it. But since the Proofreader is trying to hide in the cassette buffer, it is wiped out!

What you need is a way to load the Proofreader after you've loaded the partial program. The problem is, a tape LOAD to the buffer destroys what it's supposed to load.

After you've typed in and run the Proofreader, enter the following three lines in direct mode (without line numbers) exactly as shown:

```
A$="PROOFREADER.T": B$="{10 SPACES}": FOR X=1 TO
  4: A$=A$+B$: NEXTX
FOR X=886 TO 1018: A$=A$+CHR$(PEEK(X)): NEXTX
OPEN 1,1,1,A$: CLOSE1
```

After you enter the last line, you will be asked to press RECORD and PLAY on your cassette recorder. Put this program at the beginning of a new tape. This gives you a new way to load the Proofreader. Anytime you want to bring the Proofreader into memory without disturbing anything else, put the cassette in the tape drive, rewind, and enter:

OPEN1:CLOSE1

You can now start the Proofreader by typing SYS 886. To test this, PRINT PEEK(886) should return the number 173. If it does not, repeat the steps above, making sure that A$ ("PROOFREADER.T") contains 13 characters and that B$ contains ten spaces.

You can now reload the Proofreader into memory whenever LOAD or SAVE destroys it, restoring your personal typing helper.

Automatic Proofreader

```
100 PRINT"{CLR}PLEASE WAIT...":FORI=886TO1018:READ
    A:CK=CK+A:POKEI,A:NEXT
110 IF CK<>17539 THEN PRINT"{DOWN}YOU MADE AN ERRO
    R":PRINT"IN DATA STATEMENTS.":END
120 SYS886:PRINT"{CLR}{2 DOWN}PROOFREADER ACTIVATE
    D.":NEW
```

```
886 DATA 173,Ø36,ØØ3,2Ø1,15Ø,2Ø8
892 DATA ØØ1,Ø96,141,151,ØØ3,173
898 DATA Ø37,ØØ3,141,152,ØØ3,169
9Ø4 DATA 15Ø,141,Ø36,ØØ3,169,ØØ3
91Ø DATA 141,Ø37,ØØ3,169,ØØØ,133
916 DATA 254,Ø96,Ø32,Ø87,241,133
922 DATA 251,134,252,132,253,ØØ8
928 DATA 2Ø1,Ø13,24Ø,Ø17,2Ø1,Ø32
934 DATA 24Ø,ØØ5,Ø24,1Ø1,254,133
94Ø DATA 254,165,251,166,252,164
946 DATA 253,Ø4Ø,Ø96,169,Ø13,Ø32
952 DATA 21Ø,255,165,214,141,251
958 DATA ØØ3,2Ø6,251,ØØ3,169,ØØØ
964 DATA 133,216,169,Ø19,Ø32,21Ø
97Ø DATA 255,169,Ø18,Ø32,21Ø,255
976 DATA 169,Ø58,Ø32,21Ø,255,166
982 DATA 254,169,ØØØ,133,254,172
988 DATA 151,ØØ3,192,Ø87,2Ø8,ØØ6
994 DATA Ø32,2Ø5,189,Ø76,235,ØØ3
1ØØØ DATA Ø32,2Ø5,221,169,Ø32,Ø32
1ØØ6 DATA 21Ø,255,Ø32,21Ø,255,173
1Ø12 DATA 251,ØØ3,133,214,Ø76,173
1Ø18 DATA ØØ3
```

D

MLX: Machine Language Entry Program

Charles Brannon

Remember the last time you typed in the BASIC loader for a long machine language program? You typed in hundreds of numbers and commas. Even then, you couldn't be sure if you typed it in right. So you went back, checked the lines, tried to run the program, crashed, went back again to proofread, corrected a few typing errors, ran again, crashed again, rechecked your typing....

Frustrating, wasn't it?

Now, "MLX" comes to the rescue. MLX makes it easy to enter all those long machine language programs with a minimum of fuss. It lets you enter the numbers from a special list that looks similar to DATA statements, and it checks your typing on a line-by-line basis. It won't let you enter illegal characters when you should be typing numbers. It won't let you enter numbers greater than 255. It will prevent you from entering the numbers on the wrong line. In short, MLX will make proofreading obsolete.

Tape or Disk Copies

In addition, MLX will generate a ready-to-use tape or disk copy of your machine language program. You can then use the LOAD command to read the program into the computer, just like you would with a BASIC program. Specifically, you enter LOAD *"filename"*,1,1 (for tape) or LOAD *"filename"*,8,1 (for disk).

To start the program, you need to enter a SYS command that tranfers control from BASIC to your machine language program. The starting SYS will always be given in the article which presents the machine language program in MLX format.

Using MLX

Type in and save MLX (you'll want to use it in the future). When you're ready to type in the machine language program, run MLX. MLX will ask you for two numbers: the starting address and the ending address. You'll get a prompt showing the specified starting address. Then type in the corresponding first line of the program.

Subsequent prompts will ask you to type in subsequent lines from the MLX listing. Each line is six numbers plus a checksum. If you enter any of the six numbers wrong or the checksum wrong, the computer will sound a buzzer and prompt you to reenter the entire line. If you enter the line correctly, a pleasant bell tone will sound and you may go on to enter the next line.

A Special Editor

You are not using the normal BASIC editor with MLX. For example, it will only accept numbers as input. If you make a typing error, press the INST/DEL key; the entire number will be deleted. You can press it as many times as necessary, back to the start of the line. If you enter three-digit numbers as listed, the computer automatically prints the comma and goes on to accept the next number. If you enter less than three digits, you can press either the space bar or the RETURN key to advance to the next number. The checksum automatically appears in reverse video for emphasis.

To make it even easier to enter these numbers, MLX redefines part of the keyboard as a numeric keypad (lines 581–584).

	U	**I**	**O**			**7**	**8**	**9**
H	**J**	**K**	**L**	**becomes**	**0**	**4**	**5**	**6**
	M	**,**	**.**			**1**	**2**	**3**

When testing it, I've found MLX to be an extremely easy way to enter long listings. With the audio cues provided, you don't even have to look at the screen if you're a touch-typist.

Done at Last!

When you get through typing, assuming you type your machine language program all in one session, you can then save the completed and bug-free program to tape or disk. Follow the instructions displayed on the screen. If you get any error messages while saving, you probably have a bad disk, a full disk, or a typo in MLX. Sorry, MLX can't check itself!

Command Control

What if you don't want to enter the whole program in one sitting? MLX lets you enter as much as you want, save the completed portion, and then reload your work from tape or disk when you want to continue. MLX recognizes these commands:

SHIFT-S: Save
SHIFT-L: Load
SHIFT-N: New Address
SHIFT-D: Display

Hold down SHIFT while you press the appropriate key. You will jump out of the line you've been typing, so I recommend that you type in the SHIFT key commands at a prompt. Use the Save command to store what you've been working on. It will write the tape or disk file as if you've finished. Remember what address you stop on. Then, the next time you run MLX, answer all the prompts as you did before and insert the disk or tape containing the stored file. When you get the entry prompt, press SHIFT-L to reload the file into memory. You'll then use the New Address command (SHIFT-N) to resume typing.

New Address and Display

After you press SHIFT-N, enter the address where you previously stopped. The prompt will change and you can continue typing. Always enter a New Address that matches up with one of the line numbers in the special listing or else the checksums won't match up. You can use the Display command to display a section of your typing. After you press SHIFT-D, enter two addresses within the line-number range of the listing. You can stop the display by pressing any key.

Tricky Stuff

You can use the Save and Load commands to make copies of the complete machine language program. Use the Load command to reload the tape or disk, then insert a new tape or disk and use the Save command to create a new copy.

One quirk about tapes made with the MLX Save command: When you load them, the message FOUND *filename* may appear twice. The tape will load just fine, however.

Programmers will find MLX to be an interesting program which protects the user from most typing mistakes. Some screen formatting techniques are also used. Most interesting is the use of ROM Kernal routines for loading and saving blocks of memory. Any error code for the SAVE or LOAD can be found in location 253 (an error would be a code less than ten).

I hope you will find MLX to be a true labor-saving program. Since it has been tested by entering actual programs, you can count on it as an aid for generating bug-free machine language. Be sure to save MLX; it will be used for future applications in other COMPUTE! books.

MLX

```
10 REM LINES CHANGED FROM MLX VERSION 2.00 ARE 750
   ,765,770 AND 860                             :rem 50
20 REM LINE CHANGED FROM MLX VERSION 2.01 IS 300
                                                :rem 147
30 REM LINE CHANGED FROM MLX VERSION 2.02 IS 763
                                                :rem 162
100 PRINT"{CLR}E6}";CHR$(142);CHR$(8);:POKE53281,1
    :POKE53280,1                                :rem 67
101 POKE 788,52:REM DISABLE RUN/STOP            :rem 119
200 PRINT"{2 DOWN}{PUR}{BLK} MACHINE LANGUAGE EDIT
    OR VERSION 2.03{5 DOWN}"                    :rem 239
210 PRINT"E5}{2 UP}STARTING ADDRESS?{8 SPACES}
    {9 LEFT}";                                  :rem 143
215 INPUTS:F=1-F:C$=CHR$(31+119*F)              :rem 166
220 IFS<256OR(S>40960ANDS<49152)ORS>53247THENGOSUB
    3000:GOTO210                                :rem 235
225 PRINT:PRINT:PRINT                           :rem 180
230 PRINT"E5}{2 UP}ENDING ADDRESS?{8 SPACES}
    {9 LEFT}";:INPUTE:F=1-F:C$=CHR$(31+119*F)
                                                :rem 20
240 IFE<256OR(E>40960ANDE<49152)ORE>53247THENGOSUB
    3000:GOTO230                                :rem 183
250 IFE<STHENPRINTC$;"{RVS}ENDING < START
    {2 SPACES}":GOSUB1000:GOTO 230              :rem 176
260 PRINT:PRINT:PRINT                           :rem 179
300 PRINT"{CLR}";CHR$(14):AD=S                  :rem 56
310 A=1:PRINTRIGHT$("0000"+MID$(STR$(AD),2),5);":"
    ;                                           :rem 33
315 FORJ=ATO6                                   :rem 33
320 GOSUB570:IFN=-1THENJ=J+N:GOTO320            :rem 228
390 IFN=-211THEN 710                            :rem 62
400 IFN=-204THEN 790                            :rem 64
410 IFN=-206THENPRINT:INPUT"{DOWN}ENTER NEW ADDRES
    S";ZZ                                       :rem 44
415 IFN=-206THENIFZZ<SORZZ>ETHENPRINT"{RVS}OUT OF
    {SPACE}RANGE":GOSUB1000:GOTO410             :rem 225
```

```
417 IFN=-2Ø6THENAD=ZZ:PRINT:GOTO31Ø          :rem 238
42Ø IF N<>-196 THEN 48Ø                      :rem 133
43Ø PRINT:INPUT"DISPLAY:FROM";F:PRINT,"TO";:INPUTT
                                             :rem 234
44Ø IFF<SORF>EORT<SORT>ETHENPRINT"AT LEAST";S;"
    {LEFT}, NOT MORE THAN";E:GOTO43Ø          :rem 159
45Ø FORI=FTOTSTEP6:PRINT:PRINTRIGHT$("ØØØØ"+MID$(S
    TR$(I),2),5);":";                          :rem 3Ø
451 FORK=ØTO5:N=PEEK(I+K):PRINTRIGHT$("ØØ"+MID$(ST
    R$(N),2),3);",";                           :rem 66
46Ø GETA$:IFA$>""THENPRINT:PRINT:GOTO31Ø      :rem 25
47Ø NEXTK:PRINTCHR$(2Ø);:NEXTI:PRINT:PRINT:GOTO31Ø
                                              :rem 5Ø
48Ø IFN<Ø THEN PRINT:GOTO31Ø                 :rem 168
49Ø A(J)=N:NEXTJ                             :rem 199
5ØØ CKSUM=AD-INT(AD/256)*256:FORI=1TO6:CKSUM=(CKSU
    M+A(I))AND255:NEXT                       :rem 2ØØ
51Ø PRINTCHR$(18);:GOSUB57Ø:PRINTCHR$(146);:rem 94
511 IFN=-1THENA=6:GOTO315                    :rem 254
515 PRINTCHR$(2Ø):IFN=CKSUMTHEN53Ø           :rem 122
52Ø PRINT:PRINT"LINE ENTERED WRONG : RE-ENTER":PRI
    NT:GOSUB1ØØØ:GOTO31Ø                     :rem 176
53Ø GOSUB2ØØØ                                :rem 218
54Ø FORI=1TO6:POKEAD+I-1,A(I):NEXT:POKE54272,Ø:POK
    E54273,Ø                                 :rem 227
55Ø AD=AD+6:IF AD<E THEN 31Ø                 :rem 212
56Ø GOTO 71Ø                                 :rem 1Ø8
57Ø N=Ø:Z=Ø                                   :rem 88
58Ø PRINT"{£}";                               :rem 81
581 GETA$:IFA$=""THEN581                      :rem 95
582 AV=-(A$="M")-2*(A$=",")-3*(A$=".")-4*(A$="J")-
    5*(A$="K")-6*(A$="L")                     :rem 41
583 AV=AV-7*(A$="U")-8*(A$="I")-9*(A$="O"):IFA$="H
    "THENA$="Ø"                              :rem 134
584 IFAV>ØTHENA$=CHR$(48+AV)                 :rem 134
585 PRINTCHR$(2Ø);:A=ASC(A$):IFA=13ORA=44ORA=32THE
    N67Ø                                     :rem 229
59Ø IFA>128THENN=-A:RETURN                   :rem 137
6ØØ IFA<>2Ø THEN 63Ø                          :rem 1Ø
61Ø GOSUB69Ø:IFI=1ANDT=44THENN=-1:PRINT"{OFF}
    {LEFT} {LEFT}";:GOTO69Ø                   :rem 62
62Ø GOTO57Ø                                  :rem 1Ø9
63Ø IFA<48ORA>57THEN58Ø                      :rem 1Ø5
64Ø PRINTA$;:N=N*1Ø+A-48                     :rem 1Ø6
65Ø IFN>255 THEN A=2Ø:GOSUB1ØØØ:GOTO6ØØ      :rem 229
66Ø Z=Z+1:IFZ<3THEN58Ø                        :rem 71
67Ø IFZ=ØTHENGOSUB1ØØØ:GOTO57Ø               :rem 114
68Ø PRINT",";:RETURN                         :rem 24Ø
69Ø S%=PEEK(2Ø9)+256*PEEK(21Ø)+PEEK(211)     :rem 149
691 FORI=1TO3:T=PEEK(S%-I)                    :rem 67
```

```
695 IFT<>44ANDT<>58THENPOKES%-I,32:NEXT    :rem 2Ø5
7ØØ PRINTLEFT$("{3 LEFT}",I-1);:RETURN      :rem 7
71Ø PRINT"{CLR}{RVS}*** SAVE ***{3 DOWN}" :rem 236
715 PRINT"{2 DOWN}(PRESS {RVS}RETURN{OFF} ALONE TO
     CANCEL SAVE){DOWN}"                   :rem 1Ø6
72Ø F$="":INPUT"{DOWN} FILENAME";F$:IFF$=""THENPRI
    NT:PRINT:GOTO31Ø                        :rem 71
73Ø PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR {RVS}D
    {OFF}ISK: (T/D)"                       :rem 228
74Ø GETA$:IFA$<>"T"ANDA$<>"D"THEN74Ø        :rem 36
75Ø DV=1-7*(A$="D"):IFDV=8THENF$="Ø:"+F$:OPEN15,8,
    15,"S"+F$:CLOSE15                      :rem 212
76Ø T$=F$:ZK=PEEK(53)+256*PEEK(54)-LEN(T$):POKE782
    ,ZK/256                                  :rem 3
762 POKE781,ZK-PEEK(782)*256:POKE78Ø,LEN(T$):SYS65
    469                                    :rem 1Ø9
763 POKE78Ø,1:POKE781,DV:POKE782,Ø:SYS65466:rem 68
765 K=S:POKE254,K/256:POKE253,K-PEEK(254)*256:POKE
    78Ø,253                                 :rem 17
766 K=E+1:POKE782,K/256:POKE781,K-PEEK(782)*256:SY
    S65496                                 :rem 235
77Ø IF(PEEK(783)AND1)OR(191ANDST)THEN78Ø   :rem 111
775 PRINT"{DOWN}DONE.{DOWN}":GOTO31Ø       :rem 113
78Ø PRINT"{DOWN}ERROR ON SAVE.{2 SPACES}TRY AGAIN.
    ":IFDV=1THEN72Ø                        :rem 171
781 OPEN15,8,15:INPUT#15,E1$,E2$:PRINTE1$;E2$:CLOS
    E15:GOTO72Ø                            :rem 1Ø3
79Ø PRINT"{CLR}{RVS}*** LOAD ***{2 DOWN}" :rem 212
795 PRINT"{2 DOWN}(PRESS {RVS}RETURN{OFF} ALONE TO
     CANCEL LOAD)"                          :rem 82
8ØØ F$="":INPUT"{2 DOWN} FILENAME";F$:IFF$=""THENP
    RINT:GOTO31Ø                           :rem 144
81Ø PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR {RVS}D
    {OFF}ISK: (T/D)"                       :rem 227
82Ø GETA$:IFA$<>"T"ANDA$<>"D"THEN82Ø        :rem 34
83Ø DV=1-7*(A$="D"):IFDV=8THENF$="Ø:"+F$   :rem 157
84Ø T$=F$:ZK=PEEK(53)+256*PEEK(54)-LEN(T$):POKE782
    ,ZK/256                                  :rem 2
841 POKE781,ZK-PEEK(782)*256:POKE78Ø,LEN(T$):SYS65
    469                                    :rem 1Ø7
845 POKE78Ø,1:POKE781,DV:POKE782,1:SYS65466:rem 7Ø
85Ø POKE78Ø,Ø:SYS65493                      :rem 11
86Ø IF(PEEK(783)AND1)OR(191ANDST)THEN87Ø   :rem 111
865 PRINT"{DOWN}DONE.":GOTO31Ø              :rem 96
87Ø PRINT"{DOWN}ERROR ON LOAD.{2 SPACES}TRY AGAIN.
    {DOWN}":IFDV=1THEN8ØØ                  :rem 172
88Ø OPEN15,8,15:INPUT#15,E1$,E2$:PRINTE1$;E2$:CLOS
    E15:GOTO8ØØ                            :rem 1Ø2
1ØØØ REM BUZZER                            :rem 135
```

```
1001 POKE54296,15:POKE54277,45:POKE54278,165
                                        :rem 207
1002 POKE54276,33:POKE 54273,6:POKE54272,5 :rem 42
1003 FORT=1TO200:NEXT:POKE54276,32:POKE54273,0:POK
     E54272,0:RETURN                    :rem 202
2000 REM BELL SOUND                      :rem 78
2001 POKE54296,15:POKE54277,0:POKE54278,247
                                        :rem 152
2002 POKE 54276,17:POKE54273,40:POKE54272,0:rem 86
2003 FORT=1TO100:NEXT:POKE54276,16:RETURN  :rem 57
3000 PRINTC$;"{RVS}NOT ZERO PAGE OR ROM":GOTO1000
                                         :rem 89
```

Index

To order your copy of the *Commodore 64/128 Collection* Disk, call our toll-free US order line: 1-800-334-0868 (in NC call 919-275-9809) or send your prepaid order to:

Commodore 64/128 Collection Disk
COMPUTE! Publications
P.O. Box 5058
Greensboro, NC 27403

All orders must be prepaid (check, charge, or money order). NC residents add 4.5% sales tax.

Send ____ copies of the *Commodore 64/128 Collection* Disk at $12.95 per copy.

Subtotal $__________

Shipping & Handling: $2.00/disk $__________

Sales tax (if applicable) $__________

Total payment enclosed $__________

□ Payment enclosed
Charge □ Visa □ MasterCard □ American Express

Acct. No. ______________________________ Exp. Date ______ (Required)

Signature ______________________________

Name ______________________________

Address ______________________________

City ____________________ State ________ Zip ________

Please allow 4-5 weeks for delivery.

456697B

Notes

Notes

Notes

Notes

Notes

Three Computers in One

On the heels of the Commodore 64's success, many wondered what Commodore would do next. Some said they would somehow repackage the 64. Some said they would bring out a CP/M machine. Others said they would use updated technology and introduce a machine with more memory. They were *all* right.

The Commodore 128 is really three computers. One is the proven 64, with its huge collection of software. Another is an advanced programmer's machine, with lots of free memory and a powerful, improved BASIC. And the other is a business-person's delight, with thousands of programs available for managing, planning, and charting.

COMPUTE!'s Commodore 64/128 Collection brings you ready-to-type programs for your Commodore 128 in 64 mode. Originally written for the 64, the programs have been tested on both the 128 *and* 64. In addition, you'll find discussions of the 128's unique features. Included are:

- Programming aids and utilities, including "TurboDisk," "Triple 64," and "NoZap"
- Educational and recreational games, like "Campaign Manager," "Heat Seeker," and "Mindbusters"
- A hands-on look at the 128
- An introduction to CP/M
- Information on BASIC 7.0
- Details on the 128's peripheral ports
- Suggestions for writing text-adventure games in BASIC

If you own a Commodore 64, you'll find here some of the best software ever collected.

If you just bought a 128, this book will provide you with an excellent library of programs and important information about your new computer.

ISBN 0-942386-97-3

COMPUTE!'s
Commodore 64/128 Collection

COMPUTE! Books