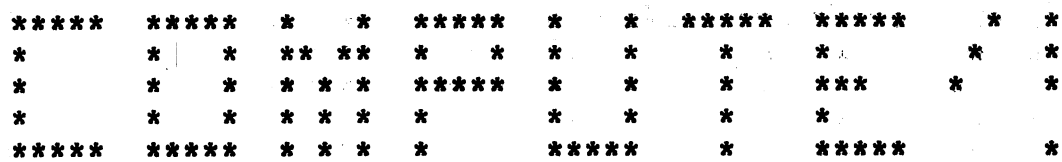

THE
COMPUTE!'S
GAZETTE
DISK
JUNE
1994



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

J U N E 1 9 9 4
================

(2) = Table Of Contents

Disk Magazine

PROGRAMS:
----

(4) = Grocery Lister

(6) = Disk Management Tool

(10) = Wizard's Demons

(12) = Word Master

(14) = Model Rocket Programs

(16) = Fast File Copier

(22) = File Drawer (PD)

(23) = Super Alarm III (PD)

(24) = Monitors

COLUMNS:
----

(27) = 64/128 View

(30) = Feedback

(33) = D'Iversions

(36) = Beginner Basic

(39) = Machine Language

(44) = Programmer's Page

(50) = G.E.O.S.

(53) = PD Picks

****

FEATURES:
----

(56) = Getting Started in Machine Language

(62) = Monitors: Gateway to Machine Language

To Load Disk:

Load "Menu",8,1 and press <Return> --- Commodore 64

****

Features:

GETTING STARTED WITH MACHINE LANGUAGE

By Jim Butterfield

So you've always wondered about machine language, but you're not sure how to start? This article should help. It explains the tools and many of the commands that you'll need. It even takes you through a simple program.

MONITORS--GATEWAY TO MACHINE LANGUAGE

By David Pankhurst

Before you can do any programming in machine language, you'll have to have a program called a monitor. This article explains many of the commands that you'll need in order to use a monitor. We've also included a monitor program on this disk for your convenience.

Columns:

64/128 VIEW by Tom Netsel
There's good news for Gazette Disk, but the news from Commodore is bleak.

FEEDBACK
Comments, questions, and answers.

D'IVERSIONS by Fred D'Ignazio
Book Learning vs. Experiential Learning.

MACHINE LANGUAGE by Jim Butterfield
Wordcount.

BEGINNER BASIC by Larry Cotton
BASIC Games and the Challenge.

PROGRAMMER'S PAGE by David Pankhurst
Screen Fun.

GEOS by Steve Vander Ark
In Search of the Perfect Driver.

PD PICKS by Steve Vander Ark
File Drawer and Super-Alarm III.

128 Programs:

Word Master 128 by D. G. Klich
A crossword puzzle utility.

64 Programs:

Grocery Lister by Tim Rich
Create a permanent grocery list of food categories and food items. Then select the items you want to purchase and indicate those for which you have coupons.

Disk Management Tool by Stephen A. Bakke
This disk utility lets you rename and delete files. It also lets you lock and unlock files and disks, restore deleted files, view the BAM, and more.

Wizard's Demons by Troy Heck
This arcade-style game puts you in the role of the wizard's apprentice, and you have to protect the wiz from being carried away by flying demons.

Word Master 64 by D. G. Klich
A crossword puzzle utility.

Three Model Rocket Programs by Burton L. Craddock
Altitude - A flight simulation program that calculates aerodynamic drag and theoretical height of one- to three-stage rockets.

Stability - This utility calculates the stability of rockets with up to three conical transitions and up to three sets of fins.

Tracking - This utility calculates the altitude of rockets using either one elevation device or two elevation-and-azimuth devices.

File Drawer 4.5 by Stan Takis
This shareware program, featured in "PD Picks," lets you create any number of self-contained database programs.

Super-Alarm III by Kirk Mook
This PD pick is a program that can turn your 64 into a device similar to a clock radio.

SuperMon by Jim Butterfield
Use this monitor program to get started with machine language.

Gazette, June 1994

GROCERY LISTER

By Tim Rich

Creating your weekly grocery shopping list by hand can often lead to omissions or errors. Grocery Lister (GL) allows you to create a permanent list of food categories and grocery items within those categories. From this list you can select items you wish to purchase as you think of them and then print your grocery list when you're ready to go shopping. You may also indicate those items for which you have a coupon, and GL will display this fact on your printed list.

GETTING STARTED

Grocery Lister is written entirely in BASIC. When you start the program, GL will automatically search the disk currently in the disk drive for a file called GL.LIST. If this file is not found, GL will create a file and initialize the categories and items in that file to a value of None. If the file is found, GL will load it into the program's memory just as it was last saved by you. After the file has been loaded, a menu will appear. This menu has four options.

INITIALIZE SHOPPING LIST

Option 1 allows you the choice of turning off all coupon indicators, turning off all buy-this-item indicators, turning off both indicators (allowing you to start a fresh shopping list), initializing all categories and items in the file to None, or pressing Return to go back to the menu.

MODIFY GROCERY LIST

Option 2 allows you to modify your shopping list. You can change the names of all categories and items, select an item for purchase, and add a coupon to an item. Additionally, upon exiting from this option, you are given the choice of saving all changes made to disk or not saving the changes.

Upon first choosing this option, a screen will be displayed showing all 15 food categories. From this screen press Return to go to the menu, use the cursor key to move the cursor, N to change the category name, or S to view the food items within that category. Press Return if you wish to save your changes to disk.

When viewing the items within a category, you'll see a screen showing all 30 of the food items within that category. From this screen press Return to go to the category screen. Again, use the cursor key to move down, press N to change the item name, C to toggle the coupon indicator, or B to toggle the buy-this-item indicator.

Toggling the coupon indicator will cause a dollar sign ($) to appear or disappear next to the item. Toggling the buy-this-item indicator will cause an asterisk (*) to appear or disappear next to the item.

PRINT/DISPLAY CURRENT SHOPPING LIST
Option 3 asks whether you wish to print your shopping list or display it on the screen. The shopping list will be printed/displayed one category at a time. Only those food items that were selected for purchase will be displayed, and those selected items that have a coupon will appear with a $ next to them.

EXIT
Pressing 4 gives you a prompt asking whether you really wish to exit.

RANDOM NOTES
The input/output device number is set in the variable DV in line 20 of the program. Currently the input/output device is set to 8; however, you may change it by simply changing the DV=8 statement in line 20 to DV=9 or whatever else you may want it to be.

It isn't necessary to enter values for the names of all 15 categories or for all 30 items within each category. The print/display option simply ignores categories and items that aren't being used.

This disk already has a sample grocery list on it for illustrative purposes only. You'll want to copy GL to a work disk and make your own customized shopping list.

Tim Rich lives in Austin, Texas.

Gazette, June 1994

DISK MANAGEMENT TOOL

By Stephen A. Bakke

Disk Management Tool (DMT) is written for the 64, 1541-compatible disk drive, and 1525-compatible printer. The printer is optional but strongly recommended.

The program provides a convenient method to perform several existing disk commands as well as a variety of additional and useful disk commands not normally available via direct keyboard command. Existing commands include Rename File, Delete File, and Validate Disk. New commands include Lock Disk, Unlock Disk, Lock File, Unlock File, Restore File, Rename Disk, View BAM, File Links, and File Length. The program also has the ability to print an expanded directory listing to the printer.

GETTING STARTED

For a long time I have believed that software should be written so that it is functional and easy to use. It also should require a minimum of documentation to make full use of its capabilities. I think this program fits that description. The program uses two scrolling menus: a Files menu and a Command menu. With the use of these two menus, selections are made in a friendly manner. Except for the Rename commands, the only keys used to navigate through your disks are the cursor up, cursor down, and Return keys.

COPYING A DISK

Before you begin, select a disk that contains a variety of programs and copy it. Hyperdrive from the April 1994 Gazette Disk works fine if you have a single drive; Double Dub 1541 from the October 1993 issue of Gazette works well if you have two drives. Commercial programs are also available to do this. The key is to copy all of the tracks and sectors, not just the active files. It is a very good idea to make a backup copy of any disk that you manipulate with this program. Do not use this program on commercial disks. Many commercial disks contain software protection schemes that this program was not designed to manipulate. (The Gazette Disk is not copy-protected.)

WALK THROUGH

Load DMT, replace this disk with your copy disk, and run the program. The initial screen displays headers for file type, track and sector of the first data block, filename, file length in blocks, and the load address. A short time after the initial screen is displayed, the disk name, ID, number of files, and blocks free and used are also displayed. The disk will remain busy for a few seconds longer while the load addresses for each of the files are read. Upon completion, the Files menu will be filled with information.

Locate DOS: 2A displayed in the upper right of the screen. The DOS type is always displayed as 2A in either normal or reverse video.

Reverse video indicates a locked disk, and normal video indicates an unlocked disk.

After the Files menu is filled with information, use the cursor up and down key to scroll through the list of files. Press the Return key to select a file. The Command menu is now visible. When the Command menu is visible, it is active. Otherwise, the Files menu is active. Use the cursor up and down key to scroll through the Command menu. After viewing each of the various commands, scroll to the top option, Go Files Menu, and press Return. In a flash, the Command menu is gone, and the Files menu is now active.

Select any file and press Return. The filename remains highlighted, and the Command menu appears. Select Lock File and press Return. The disk drive will spin, the command menu will soon disappear, and a greater than symbol (>) will appear to the left of the file type of the file you selected earlier. The > symbol indicates that this file is now locked.

DEFINITIONS

Locked files can be loaded and run, but they cannot be deleted from a disk using the normal Scratch command. This can be reversed with the Unlock File command. Locked disks also become read only. Locked disks respond to all disk commands, except Load and New, with a DOS error 73. This can be reversed with the Unlock Disk command.

Locking a disk makes it safe from most catastrophes except dog bites and the New (reformat the disk) command. You may want to continue using write-protect tabs, but if you're confident that you have a good disk-labeling system, that shouldn't be necessary.

Seven commands are file dependent, and all contain the word "file" in the name. These commands act upon the file selected in the Files menu. The remaining nine commands are not file dependent. They include commands such as Exit Program, Lock Disk, or View BAM and are not dependent upon the file selected in the Files menu.

RESTORING FILES

When you delete a file or a group of files from a disk, those files are not physically removed from the disk. Rather, those files are marked in the directory as deleted and those sectors are marked as free for use by other programs. If you save a new file to disk after a file has been deleted, some of the sectors marked as free may be overwritten by the new file. If this happens, the old file is no longer restorable.

When you delete a file or group of files from a disk and issue the Validate command, the disk controller can move sectors of the remaining files around on the disk. When this happens, some of the sectors previously used by the deleted files are overwritten. Once this happens, the old files are not restorable.

If, for example, you issue a Scratch command, OPEN15,8,15,"S0:DA*",

all files beginning with DA will be deleted (marked as deleted) from the disk. If you do not save any new files to this disk and you do not validate this disk, those files are restorable. To do this, load DMT, install a copy of the damaged disk, run DMT, select the file you want to restore and press Return, select the command Restore File and press Return. Repeat this procedure for each file you want to restore (only those that were deleted after your last Save or Validate).

When you've finished selecting files, issue the Validate command. This will verify the integrity of the BAM and verify the accuracy of the blocks free and used. After issuing the Validate command, no additional deleted files on this disk are restorable.

KEYBOARD ENTRY

Two commands require keyboard entry. These are the Rename Disk and Rename File commands. Rename File is file dependent, but Rename Disk is not. Keyboard entry is limited to a maximum of 16 characters. When a file is selected, enter the new name at the prompt near the bottom of the screen. If you make a mistake, use the Del key to remove it. Press the Return key when you've finished entering the new name.

INFORMATIVE COMMANDS

View BAM displays each sector in the Block Availability Map. Sector numbers appear on the left, and track numbers appear along the bottom. For more information about tracks and sectors, the best source available is the book that came with your disk drive.

Circles indicate used (full) sectors, pluses indicate free (empty) sectors that are free for use by future files saved to disk. File Links displays the track and sector sequence of the selected file. A comparison of the information revealed using these two commands, View BAM and File Links, can provide insight into how the disk controller manages the available space on the disk.

File Length displays the length of your files in bytes rather than blocks. This is a better measure of file length, since many files use only a small portion of the last sector. Program files also display the starting and ending address. This can be very informative, especially for assembly language programmers.

File length and file links are not displayed for relative files for several reasons. Relative files contain raw data, don't need to load into a specific place in memory, and use side sectors to store data.

The Print Listing command lists all of the information displayed in the Files menu to your printer. When a file is selected, you will be prompted to make the printer ready before printing begins. After printing is complete, the program will return control to the Files menu. If you don't have a printer, the program will return to the Files menu.

Stephen Bakke lives in Aurora, Colorado.

WIZARD'S DEMONS

By Troy Heck

An arcade-style game for the 64. Joystick required.

Wizard's Demons puts you in the role of a wizard's apprentice, whose master has just been placed under a curse by a sinister rival wizard. While your master hobbles around his courtyard in a mindless trance, demonic minions of his evil rival scream overhead, waiting for a chance to snatch your master and send him on a one-way trip to their summoner's lair.

You, the lowly apprentice, (and the few of your master's tricks that you are capable of using) are all that stand between your master and a tortuous death.

GAME OPTIONS

Wizard's Demons is written entirely in machine language, but it loads and runs like a BASIC program. After you load and run the program, an introductory screen with several options will appear. Pressing f1 or the joystick's fire button will begin the game.

Pressing f3 allows you to chose whether to begin play on wave 1, 51, 101, 151, or 201. These waves all begin with the same number and configuration of demons, but the speed of play increases when higher waves are selected.

Pressing f5 allows you to take extra teleport spells, which are explained below. The option programmed into f7 functions only when waves 51 or above are selected. When Normal is selected the time that your magic wand remains inoperable (as is explained below) will be exactly as it would if you had reached the level on your own.

Since this amount of time is too high for players who wish to dabble in higher speeds, I have included a Special selection, in which your magic wand is inoperable at speeds comparable to wave 1. It should be noted that the time your wand is inoperable is then increased the standard amount as you progress through each wave.

PLAYING THE GAME

After you have initiated gameplay, it isn't long before a demon will descend toward your pacing wizard. Your magic wand sits upon the top of the master's castle. You may maneuver it left or right with a joystick in port 2. The fire button will release an annihilation disk, which, upon hitting a demon, will cast it back to its own plane forever. Pulling down on the joystick will quickly shift your magic wand to the opposite side of the master's castle. There will be a brief pause after performing this option, but other than that, it may be done an indefinite number of times.

Try to annihilate the demons before they get into the castle. If they

do get in, the wizard will eventually collide with one and be carried aloft. When this happens, shoot the offending demon. If you are successful, your wizard will enter free fall; push up on the joystick as soon as possible to cast a feather-fall spell. This will cushion his fall. If you don't, cast the feather-fall spell before your wizard crashes into the ground, he will die and the game will be over.

If you are unable to annihilate a demon who has captured your wizard before the two of them disappear from the screen, then you must cast a teleport spell. These spells come from the master's spell book and are very scarce. You are given from three to nine depending on the option you selected in the beginning.

The teleport spell is cast automatically upon the loss of a wizard, and one spell will be deducted in the lower right-hand corner. When you lose a wizard and you have no spells left, the game ends.

Be careful not to allow your wand to collide with anything other than the wizard. This includes demonic lightning, which the demons emit periodically, or a demon descending or ascending out of the castle. If your wand is hit, its magic will be disrupted, and it will lose its integrity for a period of seven seconds at the first wave and one-tenth of a second additional penalty for each wave thereafter. When your wand has been disrupted, you can still move around, but you can't fire, which gives the demons a prime chance of stealing your wizard.

SCORING

Scoring is based upon the particular attack mode of a demon. If the demon is descending upon your wizard or stealing your wizard, you will be awarded 200 points. Demons that are circling are worth 100 points. The high score is updated at the end of every game.

If you wish to save your high score, exit out of the game with Stop/Restore, and save the program just as you would a BASIC program. The high score will be preserved the next time you play.

Troy Heck lives in Ortonville, Michigan.

Gazette, June 1994

WORDMASTER

By D. G. KLICH

A crossword puzzle utility for the 128 with a version for the 64.

If you work crossword puzzles as often as I do, this program will help you find many words that fit a pattern mask composed of the letters that you know. For example, if you are searching for a five-letter word that starts with CA and ends L, Word Master will check its list for all such words that match that pattern. Word Master uses 13 files of typical crossword entries (over 9000 words) each sorted in alphabetical order by word length (3-15 characters).

The program directions consist of four options. Option 1 lets you search for all words that satisfy the mask entry. The mask contains dashes for each of the unknown characters in the word. For the example mentioned above, you'd enter CA--L. The program would search its list of five-letter words and print to the screen all words that fit that particular pattern. "Camel" and "canal" would be two examples.

Since any letters, including the first, may be unknown, the entire file must be searched. Search time is a function of the file size, the largest file being the five-letter words, which currently contains 2607 words.

The second option offered lets you add additional words to the existing files. Update time is not as much a function of the size of the word list as it is the number and size of the files to be updated. The word list, containing up to 500 words of various sizes, is "bubble" sorted alphabetically by word size and then merged with its proper files, creating new, larger files.

The third option allows for the deletion of one word at a time from a file. You may want to use this to remove an incorrectly entered word. Again the process time is a function of the file size.

Finally, the fourth option displays all the words stored in a particular file. This feature is useful in case you would like to know why a word was not found, perhaps because it was misspelled.

Since Gazette Disk is write-protected, you cannot add or delete words when running the program from this disk. Make a backup copy of this disk or copy Word Master and the WORD sequential files to your own work disk.

The Word Master 128 program using graphics and the FAST/SLOW instructions was written in BASIC 7; a slower, less graphic version for 64 users is also included.

Donald Klich, the author of Stereogram 128, lives in Mt. Prospect,

Illinois.

Gazette, June 1994

THREE MODEL ROCKET PROGRAMS: ALTITUDE, STABILITY, TRACKING

By Burton L. Craddock

The three programs for aiding model rocketry on this disk are Rocket-Altitude, Stability, and Tracking. All three programs are compiled BASIC 64 programs. The programs load, save, and run like BASIC programs on either a 64 or a Commodore 128 in 64 mode.

The sequential file THRUST.DAT is the rocket motor's data file for Rocket-Altitude. Motor-Data is a BASIC program for creating a copy of the data file THRUST.DAT on a disk in device number 8. I used the MetaBASIC Plus READ command to display THRUST.DAT contents on the computer screen and to manually convert that display into data statements for Motor-Data by inserting a line number and the keyword "data" at the beginning of each line.

ROCKET-ALTITUDE

Rocket-Altitude is a flight simulation program of a model rocket with aerodynamic drag in perfectly vertical flight. Rocket-Altitude can simulate the flight of model rockets with one to three stages.

Before the ignition of a rocket stage, a menu of options is displayed onscreen. The f1 key selects metric units or English units. The metric system is the default unit of the first stage.

The f3 key selects a Summary report or Detail and Summary report. The Summary report is the default of the first stage. We'll go into detail about these reports later.

The f5 key selects current stage as the Last Stage or Not Last Stage. The default for all stages is Last Stage. If current stage is the third stage, the final stage-selection option will not be displayed, and the current stage is the last stage.

Press f7 to start the simulation for the current stage. At this point, you are asked to enter the rocket gross weight, the rocket body maximum diameter, and the drag coefficient. The input weight and diameter for the metric system is weight in grams and diameter in millimeters.

The input weight and diameter for the English system is weight in ounces and diameter in inches. The drag coefficient for a typical model rocket with a vertical takeoff is approximately 0.75. Some low-drag, high-performance model rockets have a drag coefficient as low as 0.25. If the rocket weight is more than 453.59 grams (16 ozs.), you will be prompted to enter the weight again.

You are then asked to select the model rocket motor type from a menu. The data for the different rocket motors is contained in sequential file THRUST.DAT. Make sure that the disk with this file is in drive 8. Device 8 must contain this file for the first motor selection.

DETAIL OUTPUT
The Detail output lists in 0.1-second intervals the time, altitude, velocity, acceleration, and weight of the rocket. If the simulation unit is metric, the altitude is in meters, the velocity is in meters per second, the acceleration is in meters per second squared, and the weight is in grams. If the simulation unit is English, the altitude is in feet, the velocity is in feet per second, the acceleration is in feet per second squared, and the weight is in ounces.

In this mode the scrolling output can be stopped by pressing the S key. A stopped detail output can be restarted by pressing the C key.

SUMMARY OUTPUT
The Summary output for the final stage of the rocket is the maximum altitude in meters and feet, the total time to peak altitude in seconds, the burnout altitude in meters and feet, and the maximum velocity in meters per seconds and in feet per seconds, and time of final stage burnout.

The Summary output for the other stages of the rocket is the burnout altitude in meters and feet, the maximum velocity in meters per second and in feet per second, and time of the stage burnout.

If this is not the rocket's last stage, the simulation will repeat the above process for the next stage. If this was the rocket's final stage, the simulation will display the message OTHER RUN (Y/N)?. Typing Y will run the program again for a new simulation. Type N to end the program.

ENTERING DATA
When entering numerical data, the only valid characters are those for legal floating point numbers and the Del key for deleting the last character. Numerical input must have at least one digit; otherwise, the Return key character will be ignored.

THRUST INFORMATION
The sequential file THRUST.DAT contains the data for different rocket motors. If you read this file with a sequential file reader, you'll see that each line contains information about one type of motor, with data items separated by commas.

The first item listed is the motor type, followed by a string in quotes that contains the thrust duration, the motor propellant mass in kilograms, and the motor thrust in Newtons in 0.1-second intervals starting at 0.1 seconds. For example, an A8 rocket motor would be listed as follows.

A8,"0.24, 0.00312, 12, 4"

The last line of data in THRUST.DAT is the word "end." Rocket-Altitude will read up to 19 rocket motor data from the file THRUST.DAT. The character limit for each line is a maximum of 80 characters.

THRUST.DAT contains data on eight rocket motors: 1/2A6, A8, B4, B6, B14, C6, C5-3S, and D12. You can create your own version of THRUST.DAT and add data for additional motor types that you may use. A program editor or a word processor with a Commodore ASCII sequential file format can be used to create a new version of THRUST.DAT.

As a convenience, I've included a short program called Motor-Data that can create and/or modify the THRUST.DAT file. Load this program and then list the DATA statements starting with line 2000. These statements contain the motor specifications for the eight motors listed above. You can add to this list or modify the ones included by adding to or changing the existing DATA statements. You can usually find the required motor specifications in manufacturers' catalogs and motor information sheets. When you run Motor-Data, it either creates or replaces the THRUST.DAT file on your work disk.

PROGRAM USES

If a model rocket's drag coefficient is unknown during a test flight but you can determine the rocket's peak altitude, you can determine an approximate value of the rocket's drag coefficient by using Rocket-Altitude. Run the program several times, experimenting with different values of the drag coefficient until the program's simulated maximum altitude equals the actual maximum altitude of the test flight.

After a rocket burns out, the final-stage motor usually releases a recovery device such as a parachute or streamer. Rocket-Altitude can help select the ideal ejection delay time for the rocket to release its recovery device. The last stage motor has a delay from the time that the motor's thrust ends to the time that the ejection charge ignites to release the recovery device. This usually occurs at or near maximum altitude. The difference between the peak altitude time and the last stage burnout time is the proper time delay.

If you enter 0 for the drag coefficient input, Rocket-Altitude will simulate the flight of a model rocket without the effect of aerodynamic drag.

Gazette, June 1994

THREE MODEL ROCKET PROGRAMS: ALTITUDE, STABILITY TRACKING

By Burton L. Craddock

STABILITY

The program STABILITY calculates the stability of model rockets with 0-3 conical transitions of the body and with 1-3 sets of fins. Each set of fins has either 3 or 4 fins. The program can handle the stability calculations for model rockets from one-stage models to three-stage models. Stability can be an aid in designing stable model rockets.

The first group of input data is the nose length and the nose base length. The nose base length is rocket diameter at the nose base. The rocket nose is the front part of the rocket. The rocket dimensions in this program are expressed in millimeters.

TRANSITIONS

If there are any transitions, enter the input data for 1-3 transitions. The input data for each transition is the transition front diameter, the transition rear diameter, the transition length, and the length between the nose tip to the transition front.

FINS

The next group of input data is the fin data for 1-3 set of fins. The data for each set of fins consists of the number of fins, radius of the body at the fins, and the fins' dimensions.

The following is a description of the fin dimensions for fins with four sides. These measurements are required for the program.

The fin root chord is the fin edge, or side, in contact with the rocket body. The fin tip chord is the fin edge with no endpoints in contact with the rocket body. The fin semispan is the perpendicular line distance from a line through the root chord and the tip chord rear endpoint. The midchord length of a fin is the distance from the midpoint of the fin root chord and the midpoint of the fin tip chord. The fin root to tip LE sweep is the distance parallel to the body of the root chord frontal endpoint and the tip chord frontal endpoint. The nose tip to fin chord distance is the distance between the nose tip and the root chord frontal endpoint.

For delta fins with three edges, the input for fin tip chord length is 0 and the input for the midchord length is the distance from the midpoint of the fin root chord and the fin tip corner.

CENTER OF GRAVITY

The next input data is nose tip to center of gravity distance. The center of gravity is found by balancing the rocket model with a loaded motor installed for each stage on a narrow edge.

NOSE SHAPE

The last input data is a menu selection of three nose shapes. The three nose shapes are ogive, cone, and parabola. An ogive nose shape is similar to a parabola but has a tip that is sharply pointed instead of rounded like a parabola.

READOUT

The output includes the normal force, the moment, and the center of pressure for the transitions, fins, and total rocket; the stability margin in distance from center of gravity and in calibers for the total rocket; and the stability as unstable, questionable, or stable.

After the output results, the user has the option of running the program again.

Gazette, June 1994

THREE MODEL ROCKET PROGRAMS: ALTITUDE, STABILITY, TRACKING

By Burton L. Craddock

TRACKING

The program Tracking calculates the altitude in meters of a model rocket using either a single elevation-angle-only tracking device or two-station elevation and azimuth angles tracking device.

The single elevation-only tracking device method assumes that the rocket flight is perfectly vertical with no change in the horizontal distance between the rocket and tracking device. Most model rocket flights will not have a perfectly vertical flight, however. The baseline for this method is the horizontal distance in meters between the launch pad and the tracker.

The two-station theodolite system (elevation and azimuth tracking) can accurately track a rocket altitude with a horizontal component in its motion. An azimuth is an angular measurement in the horizontal plane. The baseline for this method is the distance in meters between both theodolites. This method is used worldwide for contests and record-setting peak altitude measurements. This method will not work when the rocket is directly above the line through the baseline and both azimuth angle readings are either 0 or 360 degree.

The two theodolites are calibrated when both theodolites are directly aimed at each other along the baseline and the azimuth and elevation pointers on both theodolites read 0.

ONE STATION

Tracking starts with a menu selection. You press the f1 key for the single elevation-only tracking devise or press the f2 key for the two-station theodolite system.

After the tracking method has been selected, you are prompted to accept the current baseline by pressing the f5 key or to enter a new baseline by pressing f7.

The next steps for the single elevation-only tracking are the following steps. For the single-elevation method, you are then prompted to enter the elevation angle. The elevation is an angle between 0 and 90 degrees, measured at the rocket's peak altitude. After this figure has been entered, the computer calculates the rocket altitude. You then have the option of running the program again or quitting.

TWO STATIONS

If you have two-station theodolite tracking, you are prompted to enter the azimuth and the elevation angles of both stations. The azimuth angles are between 0 and 360 degree. The elevation angles are between 0 and 90 degrees.

The output provides altitudes calculated from each station, the

average of both altitude values, the percentage error from the average altitude, and the printout TRACK LOST for absolute percent errors greater than 10 percent or TRACK OK for absolute percent errors less than 10 percent.

Burton Craddock has a Bachelor of Science degree in computer science with a minor in physics from the Illinois Institute of Technology. He has been programming on the 64 since 1987. He lives in Chicago, Illinois.

Gazette, June 1994

average of both altitude values, the percentage error from the average altitude, and the printout TRACK LOST for absolute percent errors greater than 10 percent or TRACK OK for absolute percent errors less than 10 percent.

Burton Craddock has a Bachelor of Science degree in computer science with a minor in physics from the Illinois Institute of Technology. He has been programming on the 64 since 1987. He lives in Chicago, Illinois.

Gazette, June 1994

FILE DRAWER

By Stan Takis

File Drawer is a public domain database management program that is discussed in Steve Vander Ark's "PD Picks" column.

File Drawer creates a database that is self contained. That is, is creates a separate runnable program for each database. This program is saved back to disk whenever you modify it. To use File Drawer, load it from side 2 of Gazette Disk but do not run it. Insert a formatted work disk and then type RUN.

You'll be asked for a filename for your database and then you'll be asked how many fields you want. You are allowed up to 300 records with 8 fields each.

For example, you might have a databasae called Club. It could contain information abou members of your club. Some of the fields might be member name, address, city, state, telephone number, and date dues are due. In this example the program name would be CLUB and the numbr of fields would be six.

After creating this databsae, to access it again you would load and run CLUB, not File Drawer. Use File Drawer only to create a new database. Try it a few times and you'll soon see how it works.

Gazette, June 1994

Gazette, June 1994

SUPER-ALARM III

By Kirk Mook

Super-Alarm is a public domain program that is discussed in Steve Vander Ark's "PD Picks" column.

You can use Super-Alarm for a clock, an alarm clock, or a timer to start other computer programs. It also plays music files that have a .MUS extension.

There are no separate instructions for using this program; documentation is built in.

Gazette, June 1994

MONITORS--GATEWAY TO MACHINE LANGAUGE

By David Pankhurst

In this article, I'll be discussing monitors--but not the type that you're probably looking at right now to read this article. I'm talking about software you need for programming in machine langauge.

Monitors are a must for serious assembly language work. Commodore 128 owners have a monitor built into their machines, and it's summoned with the command MONITOR.

If you own a 64, you'll have to load a monitor just as you would any other program. There are several monitors available for the 64. Some popular ones are MicroMon, CBM Mon, and SuperMon. (Look for SuperMon on the flip side of this disk.) Depending on the type of monitor you have, once you've loaded it into memory, you start it with either a SYS command (MicroMon) or by typing RUN (SuperMon).

Unlike the free-form programming of BASIC, monitors expect input to follow strict guidelines. All commands are prefixed by a period, followed by a letter or character and a single space.
The various text and numbers following the command, called its parameters, are separated by a comma or space. If the data is numeric, it must be a four-digit hex number for addresses or a two-digit number for everything else. Pad with Os if necessary. Because of its banking system, the 128's monitor uses five-digit numbers for addresses. Mistakes are sometimes flagged with a question mark, or they may simply be ignored.

For programming flexibility, 64 monitors reside in various memory locations, typically at the top of BASIC or $C000-$CFFF. The latter can be a problem, since that area is a favorite for placing code. Monitors come in two versions for different locations. They may be automatically relocatable (like SuperMon), or else they will have instructions for relocating.

The commands for SuperMon are shared by all monitors, although syntax may differ somewhat. Here are some examples.

.A C000 LDA#$56 -- Assemble
After .A comes the address at which to assemble the code. The address is followed by the assembler mnemonic, followed by any appropriate data. Enter one instruction per screen line, and follow the syntax exactly. Include the dollar sign ($) when entering two- or four-digit hex numbers. If the code assembles correctly, you are prompted on the following line with the next available assembly address.

.D 5600 - Disassemble
The Disassemble command displays a screen full of assembly code, starting at the input address. Where data cannot be converted to a valid code, question marks are displayed.

.F D800 DBFF 00 - Fill
The first two parameters in the Fill command are the starting and ending addresses to fill, followed by the byte to store. (This example would set color memory to black.) Be careful with this command, because you can easily overwrite valuable places in memory such as the monitor itself or page 0.

.G C000 - GOTO
This runs your program, with the starting address as the only parameter. At the end of the program, an RTS instruction returns you to BASIC, and BRK starts the monitor.

.H A000 BFFF 53 51 - Hunt
This line hunts for matching characters in memory. The first two values are the memory range to examine, and the characters following are the bytes to look for. One or more bytes can follow, each with a space between. In this example, BASIC ROM is scanned for bytes $53 $51, which begin the keyword SQR.

.L "filename",08 - Load
The syntax for loading can differ widely from monitor to monitor. For SuperMon, "filename" will be reloaded from disk to the location from which it was saved. (See the section on Save.)

.M C000 C040 - Memory
This command displays the range of memory as hex bytes, eight per line. Cursoring up and changing values (with Return) pokes them to memory. The Run/Stop key can be used to stop a lengthy listing.

.R - Registers
All the CPU registers are displayed and can be changed by cursoring to them, editing, and pressing Return. These become the startup values used by GOTO.

.S "filename",08,C000,C200 - Save
The filename and device in a Save are identical to the Load command; the range of memory to save follows. When loading, the file is reloaded to this location in memory. The saved ending address should be one more than necessary; in this example, all data including that at $C1FF would be saved, but not $C200.

.T C300 C450 CD00 - Transfer
The first two parameters are the start and end of the memory block to transfer. The next number is the destination starting address. In this example, the byte at $C300 is copied to $CD00, the next at $C301 to $CD01, and so on. As in the Fill command, care is needed, since transferred data can overwrite important memory locations.

.X - Exit
You can quit the monitor with this command. To reenter the monitor, execute a BRK instruction (hex $00). This can be done by typing SYS 13 with some monitors. With SuperMon, type SYS 38893.

Coming next month:
"Starting Off in Assembly Language"
We'll take a look at assembly language mnemonics and go into more detail about programming at the machine level.

Remember, you can load and run SuperMon from the flip side of this disk.

Gazette, June 1994

64/128 VIEW: Where to from Here?

By Tom Netsel

If you've ever heard what the television networks charge to sponsors of major TV events, you know that there can be big bucks in advertising. Television networks command huge sums for a few seconds of air time because their programs deliver millions of viewers to the sponsor's message.

Magazines, with smaller audiences, can't attract the same fees as television, but corporate players still have to pay hefty prices for full-page ads in major publications. Newspapers and magazines earn the major portion of their profits from advertising. The income that subscriptions generate generally accounts for only a small portion of overall revenue.

When major software companies stopped buying ads in Gazette, we felt the pinch that was reflected in the Commodore market as a whole. Advertisers said they weren't getting the responses that they once were. Then, they started losing money on new Commodore titles. Commodore users weren't buying as they once were. Also, there were fewer Commodore users out there. Many of them were abandoning 8-bit technology in favor of more powerful machines whose prices had suddenly dropped.

A spiral was set into motion that still affects Commodore users today. Technology advanced and became cheaper, so many Commodore users switched to newer and more powerful machines. They stopped buying Commodore titles. Profits at software companies fel, so they dropped the Commodore line and followed the dollar to the PC market. Without Commodore titles to sell, these companies dropped their ads in publications such as Gazette. With Commodore titles becoming tougher to find, more computer users switched to the newer machines. Little by little, Commodore's downward spiral continued.

As I've mentioned before in this column, without the big software and hardware companies buying ads, Gazette has had to rely on subscribers to remain in business. Gazette hitched a ride with COMPUTE for a while, letting it absorb some of the operating costs.

A big jump in printing costs forced an end to that, and Gazette had to earn its own way or fold. It had to attract enough subscribers to cover all costs and provide the company with enough profit to make the venture worthwhile. Switching to disk seemed to be the answer. We still have ads, but their cost reflects Gazette's smaller size. (All the ads on Gazette Disk generate barely enough revenue to pay for one of the programs that I purchase each month.)

That's where we stood a few months ago when we switched Gazette format. Now, I'd like to pass along some news that I've recently learned. Management is pleased with the number of subscribers who've

made the switch with us from paper to disk, so it looks like Gazette Disk will be around for some time. I'd like to take this opportunity to thank all of you for your support. I hope you'll encourage other Commodore users to give us a try. I know the change has been awkward and not without its share of glitches, but I think Gazette's rolling again.

I wish I could say the same for Commodore. As I write this in early April, Commodore seems on the verge of bankruptcy. The price of its stock has dropped to well below a dollar per share. It's sunk so low, in fact, that trading of its shares has been halted.

According to its latest financial report, for the six-month period that ended December 31, 1993, Commodore's net loss was $17.9 million compared with a net loss of $96.0 million in the prior year. Sales for the six months were $152.7 million compared with $396.3 million in the year-ago period.

Sales over Christmas were down, hindered by the company's limited financial resources, which hampered the supply of products. Sales of the new Amiga CD32 game machine had problems due to a weak game-market environment in Europe. Sales of the Amiga 1200 strengthened, but not enough to make much of an impact on Commodore's overall financial picture.

Last year's net loss and the continuing losses for the first and second quarters of the current fiscal year have had a severe impact on the West Chester, Pennsylvania, company. Commodore is attempting to negotiate a restructuring plan with its creditors and suppliers. In the absence of additional resources and a restructuring, the company may become subject to filing for bankruptcy or face other liquidation proceedings.

A footnote to its financial statement reported that the Commodore's "financial position and operating results raise substantial doubts about the company's ability to continue as a going concern."

So what's this mean to us Commodore users? Well, in the past, companies such as Texas Instruments and Radio Shack discontinued lines of computers, turning those machines into "orphans." Now, it looks like the public--including Commodore owners--is going to drop its support of Commodore. I don't know if that makes Commodore an orphan or not, but I believe the Commodore community can stand on its feet and take what comes.

There's still a good solid core of Commodore users around the world that's still active. They use their machines most every day, they communicate with one another, they know where they can find products and service, and they know where they can turn for help. In an effort to make Gazette Disk more useful as a Commodore resource, we're now offering classified ads to individuals. If you're looking for a piece of hardware or software or if you have something to sell, you can now get your message to thousands of other Commodore users. Look for

details in the advertising section of this disk.

So what happens to us Commodore users if the parent company isn't around? We'll survive. It's a mature group and a resourceful one; it can stand on its own feet. As in any family, the death of a parent is a tragic event, but the family itself survives. And like the family it is, the Commodore community will survive.

Gazette, June 1994

FEEDBACK

BUYERS GUIDE UPDATE
Here's an update from the buyers guide that appeared on the March Gazette Disk. Dave's Computer Store has moved; its new address follows.

Dave's Computer Store
32400 Aurora Rd.
Solon, OH 44139
(216) 248-4514

INSTRUCTION PRINTOUTS
I like your disk, but you should have some way of printing out the documentation that accompanies the programs.
JACOB GOLDMAN
BROOKLYN, NY

Perhaps you've missed it, but we do have a way to print any of the text files on Gazette Disk. When you load a text file from the menu, the opening screen is a help menu. This explains how to change the text and background colors and how to print. You can change background color at any time you are reading an article or documentation by pressing the B key until you get the color you want. Change text color by pressing the T key.

To print with the default parameters, press P. To select your own parameters, hold down the Shift key and press P. You can bring up this help screen at any time by pressing H.

Remember that the documentation on Gazette Disk is in SpeedScript format, which consists of screen codes saved as PGM files. If you wish, you can load these files into SpeedScript for reading editing or printing. If you want a printout with different margin settings, for example, you could set those while in SpeedScript. If you have The Write Stuff, that word processor can load and display SpeedScript files.

COPYRIGHTS AND MORE 128 PROGRAMS
Your comments regarding Gazette material being copyrighted are a bit misleading when the disk contains public domain programs mentioned in Steve Vander Ark's column.

Stereogram 128 was worth the price of the March disk. I experienced this effect years ago as a young boy while gazing at the wallpaper in our living room and discovered that "unfocusing" my eyes to line up two identical flower patters resulted in a 3-D effect.

Do you want more 128 programs? Only one 128 program seems a bit stingy.
DICK ESTEL

FRESNO, CA

All programs and articles that we publish on Gazette Disk are copyrighted by our parent company with the exception of the public domain programs discussed in "PD Picks." Our copyright does not cover those programs. We offer these programs as a convenience to our readers who may not otherwise have access to them. Whenever we can locate the author of a PD program that Steve discusses in his column, we pay that author an honorarium for the use of that program but the author retains the copyrights.

Yes, we do want to publish more 128 programs. We have a large number of submissions waiting to be evaluated, but there are very few good 128 programs. Most of them are for the 64.

Since we've changed to disk format. I'd like to know how many of our current readers have and use 128s and how many of those have 80-column capability. we'll be publishing a new reader survey shortly, so we can better serve your computing needs. Meanwhile, it might help if you write "128" on the envelope of your 128-program submissions.

OUT OF BUSINESS
In your March buyers guide you list a company called Diskoveries of Waukegan, Illinois, as a seller of Commodore products. I mailed an order to this company in December, my check was cashed, but my order was never filled. I sent a letter in February but it was returned as undeliverable. I'd like to warn other Commodore users that this company apparently is no longer in business.
JULIAN KOCZUR
BUFFALO, NY

IT'S YOUR PUTT
The Putt-Putt game on the April disk is a good one. I compiled it with Blitz! and it plays more smoothly and quickly. Other subscribers with compilers might want to try it.
DON RADLER
CAPE CORAL, FL

The April PD-Pick Putt-Putt has a bug in it. You will notice that the speed of the ball is displayed with a range of from 0-5. As written, a selection of 0 is not allowed. There are in fact, ten speeds available, ranging from 0-9. Here is a solution.

In line 2950, change SPEED (0-5) to SPEED (0-9).

In line 2960, change IF SP>0 AND SP<9 to IF SP=>0 AND SP<10.

In line 2970, change SPEED IS FROM 0 to 5 to SPEED IS FROM 0 to 9.
BOB MARKLAND
NEWCASTLE, WY

Since Putt-Putt is an entertaining public domain program, but bear in

mind that Gazette does not check PD programs as closely as it does the one's purchased for publication. While the instructions say that speeds range from 0 to 5, the code in line 2960 would seem to indicate that a ten-point range was planned at one time. It will give you more power. Thanks for your comments.

HIDDEN SYNTAX ERROR
When I load and run SpeedCheck. DM a menu appears which has five items on it. If I press choices 1 or 4, list the words or list to printer, I get a syntax error in 260. Listing the line reads as follows.

```
260 W$=W$+CHR$(K): IFSTORFTHENRETURN
```

I am not a programmer, so I cannot fix this line to get my dictionary to print to screen or printer. Can you help?
RAYMOND FABER
PHILADELPHIA, PA

When programming in Commodore BASIC, it usually doesn't matter if you run words and commands together without spaces. The Commodore interpreter can usually figure things out. On occasion, unexpected problems do pop up. That's what's happening in the command that follows the colon.

We read that code as IF ST OR F THEN RETURN, but the 64 doesn't see it that way. By running the ST and OR together (STOR), the 64 sees the BASIC word TO as being in the middle of an IF/THEN statement. To fix the problem, simply insert a space between the ST and the OR.

VIDEO REQUEST
Can you help me locate the company that sells VideoByte and VideoMate?
KEN HARRISON
NOVA SCOTIA, CANADA

Contact The Soft Group in Montgomery, Illinois. The telephone number is (708) 851-6667.

If you have a question or comment, send it to Gazette Feedback, 324 West Wendover Avenue, Suite 200, Greensboro, North Carolina 27408.

Gazette, June 1994

D'IVERSIONS:
The Original Reality Divide: Book Learning vs. Experiential Learning

By Fred D'Ignazio

When I was a kid, I thought we all experienced the same reality. Then I got to know my father. My father had been raised in an Italian ghetto in the Great Depression. He grew up a savvy, street-smart kid who knew how to read and write but never searched for reality between the covers of a book. My father would look at me, studying and reading books all the time, and he'd just shake his head in dismay.

To my father, reality was something you collided with each morning when you fell out of bed and put your warm toes on the cold, hard floor. Only by being out in the world, striving, stumbling, and enduring life's hard knocks could you come to know what reality was. I, on the other hand, believed in the formula of "reality through reading." I thought you could learn about reality by reading about great ideas and great lives, by thinking great thoughts, and dreaming great dreams.

That was then. Now I have a hunch that reality is a lot more complicated than either of us realized. Reality is neither my father's objective concept that you just have to crawl through--or survive--to learn, nor is it my conceptual reality of ideas. Instead, there could be lots of different coexisting realities. For example, a totally different reality could be nearby as I write this column. It could reside in the mind of the person sitting next to me on a bus as we glide down a nighttime highway into cyberspace.

WHAT IS THE FLAVOR OF YOUR REALITY BUBBLE?
A person makes his or her reality. That's the reality-bubble idea. Each of us is surrounded by our own reality bubble. It's an invisible aura that we create through a combination of our unique nature and nurture--partly from life experience; partly from the person we're becoming; and partly from the thoughts, emotions, and experiences that we're living through this very moment.

A reality bubble is not a mystical or transcendental concept. You sense it when you come near a person and really look at that person closely or listen seriously. It's that slight "bump in the road" or discontinuity you feel whenever your peach-flavored bubble collides with someone's watermelon bubble. Your bubble pops, and you realize that not everyone thinks the way you do or sees things the same way you do. It's sort of a weird feeling, where for a moment you question the other person's sanity or humanity--or begin questioning your own. Then the next moment you realize this person is experiencing life in ways that are completely alien to you. And you are an alien to him or her. That person is inside a reality bubble, and you're inside a different reality bubble of your own.

STRONG BUBBLES, WEAK BUBBLES

One of the first things we notice about other people is their force of personality. We experience this force when we collide with their reality bubble. The stronger the other person's reality bubble, the more forceful the collision. If the other person's reality bubble is especially strong or our bubble is particularly weak, it's not long before our bubble shatters, and we suddenly start perceiving reality through the other person's bubble. We begin thinking and acting more like the other person than ourselves. This effect is especially dramatic with extraordinary persons. Adolph Hitler, Winston Churchill, and Mahatma Gandhi had bubbles that were so strong that they popped the bubbles of entire peoples and societies.

RABBITS IN THE REALITY TUNNEL

Is it really that simple? Reality bubbles may explain a lot, but are people really that self-contained, that isolated and alone? Maybe bubbles is the wrong concept; perhaps it should be reality tunnels. We could be reality rabbits, constructing warrens and mazes of reality tunnels that we inhabit socially with other like-minded rabbits.

Like-minded people seek out and attract like-minded people. We all are drawn powerfully to people like us (PLUs). We look for PLUs and then latch onto them. We like to work with them, play with them, live near them, hang out with them. If we all traveled around together, I'd call it a giant, multiperson reality bubble. But we are rarely in the same place our PLUs are or traveling at the same pace or same direction. Instead, many of our closest PLUs are scattered across space and across time.

THE PLU MAP OF THE WORLD

So instead of a bubble or tunnel, maybe reality takes the form of a yellow brick road--a personal system of highways and byways through Oz. We and our fellow PLUs have mapped this system and travel it carefully to make sure we don't veer off onto any soft shoulders; miss important exits; or find ourselves on strange, unmapped back roads. We all realize there are evil witches lurking around the next corner and lions, tigers, and bears in the dark forest just beyond the boundary lines of our road. So we cling to our map tightly because it's a security blanket, a beacon, a ticket for safe passage through a frightening, violent, chaotic, and puzzling larger world.

THE ALCHEMISTS OF CYBERSPACE

But now we are entering cyberspace, a world in which fabricated, artificial realities will become more real than physical realities. What kind of road map will we carry to guide us safely through cyberspace? What kind of fellow travelers will we find there?

Back in the 1950s and 1960s, it was easy to recognize the early travelers in cyberspace: white males wearing dark suits or lab coats. In the early years before the seas of cyberspace began to rise to envelop human consciousness and the human imagination, cyberspace was limited, finite, more like a puddle. The only people playing in the puddle were grownups, mostly men. These were the early alchemists of cyberspace. They cast arcane hexes using magical terms like CPU and

RAM and wrote long formulas called algorithms for the transmutation of ordinary reality into data, the elementary particles of cyberspace.

They always spoke in capital letters. Their company names were capital letters like IBM and RCA, and their computer names were written in capital letters like UNIVAC.

Then in the 1970s and 1980s, the cozy, clubby world of the cyberspace alchemists suddenly burst. With the advent of low-cost personal computers, all sorts of riffraff were making their way into cyberspace: old people, young people, women (and girls!), white people, people of color, Catholics, Latinos, and atheists. All ventured into cyberspace, and they took their collective consciousness with them. Cyberspace, that once small puddle, began to swell and grow. Instead of a tight, technical pool of white, male, middle-class thought, cyberspace has evolved into a rainbow-colored, multicultural soup.

Gazette, June 1994

BEGINNER BASIC: BASIC Games and the Challenge

By Larry Cotton

Thanks for all your responses to the challenges I threw out a few months back. This month we'll look at how two programmers approached the program of writing a BASIC simulation of the card game Sets.

Granted, these programs were not written by tyros--far from it. But there are certain concepts and techniques common to both efforts that illustrate solid, well-structured programming.

What sorts of games can be written in BASIC? Can an arcade-style, blast-the-aliens-at-warp-speed game be written in BASIC? Yep, but the aliens would probably attack at the speed of cold molasses, and anyone playing the game would probably nod off waiting to see the results of a joystick nudge. Arcade-style games are usually written in the native tongue of the 64, machine language, which doesn't require every command, such as POKE or PRINT, to be interpreted.

Could a nice, slow-paced game like chess be written in BASIC? Again, the answer is probably yes (at least in a rudimentary form), but you'd not only doze off waiting for the computer to analyze every move, you could probably turn off the monitor and get a good night's sleep.

So fast games are impractical, and slow games that require the computer to do a lot of thinking in the form of mathematical calculations and/or evaluations of numerous logical statements are out. So what's left?

That depends on your definition of a game. In last December's issue, a BASIC program written by William F. Snow, described as "an enjoyable way to practice spelling words," could have been considered a game, even though it was designed primarily for teaching fifth graders to spell.

The preceding July issue featured another "game" by the same author. It cleverly disguised an educational program about learning the names and capitals of the 35 countries in the Americas as a Scud missile-annihilating game.

Obviously, many educational games are perfectly suited to BASIC. They do only a modicum of mathematical manipulations, display sparse graphics, and can easily keep up with the player's activities. The answers to the drills (spelling, geography, math, whatever) are usually coded into BASIC's DATA statements. There's relatively little for the computer to do; it spends most of its time figuratively tapping its foot.

The card game Sets could be described as an educational game: It presents 12 randomly arranged groups of three figures and challenges the player to quickly pick a set. In the process, the player soon

learns what a legal set consists of and what it doesn't.

My original definition of a set, which was deliberately somewhat vague, required that it contain three groups of three figures which may share four characteristics: quantity, color, shape, or fill. Characteristics of the figures within a given group must be either identical or completely different.

The computer's basic task is to present the figures randomly, wait for the user to pick a set, then decide whether or not the chosen set is legal. All of this can be accomplished rather expeditiously in BASIC.

Of the submissions, I chose one from Arthur Moore of Orlando, Florida, and another from James de Weese of Downey, California, as excellent examples of BASIC programming and creative solutions to the Sets challenge.

Arthur's game more closely paralleled the actual card game. After a setup time of about eight seconds, his cards were neatly laid out like playing cards in a 3 x 4 grid pattern, lettered A - L. The figures he chose were diamonds, hearts, and solid circles on backgrounds that were either red, blue, or green. For fill, Arthur used a plain, checked, or striped background pattern.

After the player selects three cards, the computer decides whether the chosen sets are valid. Arthur's program gave the computer the ability to show sets of two as well as three cards.

James took a few liberties with the rules, simplifying the game to be playable by young children. His version also randomly displayed 12 cards lettered A - L but could be played by up to five--including the computer. James used up to six figures, shapes, and colors. His setup time was almost nil.

James's game required that a set consist of only one shared characteristic. In other words, if all three chosen cards were red, whether or not they shared the same figure or number of figures, the set was legal. One small quibble: If the computer played, it always picked color as the set's common characteristic.

Both programs featured excellent error-trapping and user-friendliness. Basic instructions were displayed onscreen, and if a player pressed the wrong key, nothing happened. Lacking in both programs (but a rule in the real game) was the ability to pick three cards whose characteristic(s) of the figures within a given set could be completely different.

Here are a couple of highlights of Arthur and James's programming techniques, in their own words, but edited for brevity:

Arthur: "My approach to tackling the Sets problem was to use BASIC's AND statement. Each card has a 12-bit value attached to it. The 12 bits are broken into four sets of 3 bits, which contain the

characteristics of each card. To see what characteristics any two cards (share), all that's necessary is to AND the values together."

James: "The computer uses Z to loop through the card variables, X and A to loop through the cards, and W to keep count of its selections. For each card variable Z, it chooses the cards one by one as base cards; for each base card, it seeks NC-1 other cards that have the same value of the current variable. X references the base cards and A references the other cards being compared with it. The last card on the grid need not be considered as a base card, since by the time X reaches it, all the other cards have already been compared with it. The routine abandons its nested loops whenever W indicates that the necessary number of cards has been selected, and, in any case, CV is set to the current variable If CV=TV upon return, no set was found."

As I said, these programs weren't written by beginners, but it goes to show that BASIC isn't just for beginners either.

Gazette, June 1994

MACHINE LANGUAGE: Wordcount

By Jim Butterfield

In this session, we'll do a simple word-counting program. It will run on the VIC-20, 64, or 128; with one minor change it will run on earlier Commodore 8-bit machines. The coding will be slanted for a particular technique: reducing the number of branches.

No matter what programming language you use, it's always advisable to keep to a minimum the number of program transfers: branches, jumps, gotos, and so on. Such transfers are a major cause of program bugs. Methods such as structured programming are designed to keep program transfers to a minimum. By the way, subroutine calls and returns are not considered part of the problem.

Branches and jumps are unavoidable in machine language programs, and they're the only way to set up program loops. They're usually also needed to implement a program decision, the equivalent of IF/THEN statements. But sometimes we can manage to code decisions without making use of a branch instruction.

Wordcount does just this in counting words. At first, the task seems impossible without a branch decision of some sort. As we scan through the text, we must add 1 to the word count only when we see a switch from a nontext character to a text character. How can we possibly detect this without a decision?

The programming shown below illustrates how it can be done. Using branches would produce faster code, but our emphasis in this lesson is on technique. The program leaves a trail of information, mostly in the carry flag. Once you have seen this sort of thing (although perhaps not as extreme an example as this one), it's surprising how often you can find a comparable task that will let you avoid branches.

THE PROGRAM

We'll skip quickly over the first part of the program, which sets the word-counter variable to 0. That's three bytes at hexadecimal 2100 to 2102. It also sets the text-type flag at 2103 to 0, signaling nontext. When we see this flag switch to text mode, we must add 1 to the word counter.

Let's move along to where we connect to the text file. This file will have been opened as logical file 1 by the BASIC program. At address $200B, we hook it to the input stream with the following instruction.

```
LDX #$01 : JSR $FFC6
```

At 2010, we have our main loop which reads the file. First, we get a character from the file.

```
JSR $FFE4
```

Exactly what constitutes a text character, anyway? Certainly the numerics and upper- and lowercase alphabetics can be counted in, but punctuation is tricky. For ASCII and Commodore-ASCII files (PETASCII), we'll use the following simple rule: Any characters from hex 30-7F (decimal 48-127) and from hex B0-FF (decimal 176-255) will be considered text. We can streamline this rule by knocking off the high bit (bit 7) using AND; the text characters will all be above hex 30.

Here goes. At $2013, AND #$7F gets rid of the character's high bit. Then CMP #$30 will test the character for text status. If the character is in text mode, the carry flag will be set; otherwise, it will be clear.

Now we get tricky. ROR A (ROtate the A register Right) moves that carry flag into the high bit of the A register. The high bit will be set for a text-mode character, clear otherwise. The other part of A will contain our original input character, somewhat mangled, but we don't care about that. Only the high bit is of interest. Then, TAX (Transfer Accumulator to index X) copies the A value into register X as well.

At 2019, the EOR (Exclusive OR) command comes into play. You may recall that address $2103 contains the previous text-mode status. Is it the same as the new status? If so, we do nothing. EOR $2103 substitutes for a comparison. If the two modes are the same, the high bit of A will be 0. Remember that the high bit is all we care about. We may be passing through a word of text, or we may be sliding over a series of spaces. Either way, EOR 1 with 1 or EOR 0 with 0, we don't want to do any counting. On the other hand, if the mode is changing and that means we're starting or ending a word, the high bit of A will be 1.

We may now log the revised status with STX $2103. Again, only the high bit will interest us. Next time around, $2103 will be used to check against the following character.

Back to that high bit of A. It will be set at 1 if we are at either the start or the end of a word, but we don't want to count both of these occurrences. We must count only when the word starts, not when it ends. We can fix this situation easily with an AND $2103. We just stored the new character's status in the high bit of $2103, remember? If we're starting a word, the high bit of A will remain on. If we're at the end of a word, A will be switched off.

Our high bit is now set only at the start of a word. We flip this bit into the carry flag with ASL (Shift Left) A. Other bits may move around within A, but the high bit, now moved to the carry flag, is the only one we will use.

Let's add 0 to the counter. Huh? How will that help? Here's the trick. If the carry flag is on, we will actually add a value of 1 (0 plus the carry). The carry flag will be on only when we find the start of a

word in our text stream. So we'll count the words without a single branch.

For ease in output, the program adds in decimal mode. That saves work when we output the final value. You'll see the add-0 code at $2023 to 2035.

At $2026, we test to see if we're at the end of the file and loop back if we're not, using this line.

```
LDA $90 : BEQ $2010
```

When we reach the end of the file, we must disconnect from it. BASIC will close the file later; we disconnect with a simple statement.

```
JSR $FFCC
```

You'll find some short code at $203D to output the six-digit count; this includes a subroutine at $2049 to split apart the two decimal digits in each byte and print them.

Is all of this subtle programming worth the trouble? Well, the program works well; it's compact and reasonably fast. Of course, you might prefer to stay with the more popular branch-a-lot programming methods, but it's still fascinating to see that there's another way of tackling decisions.

The program on the flip side of this disk is called Wordcount. It consists of a BASIC loader that pokes the machine language data into memory and a file reader that opens the sequential file whose words you want to count. At the filename prompt, be sure that the disk with that file is in the drive.

The following is the program's source code. You can list Wordcount on the flip side to see its BASIC loader.

```
100 ; wordcount source listing.
110 ;       jim butterfield 1984
120 ; the following defines the 3-byte
130 ; counter, and the 1-byte char-type
140 ; flag.  they can, of course, be
150 ; placed in other locations.
160 count = $2100
170 ctype = count+3
180 ; here comes the program.
190 *=$2000
200 ; clear the three-byte counter
210 ; also the text-mode flag at 2013
220   lda #$00
230   tax
240 cloop sta count,x
250       inx
```

```
260       cpx #$04
270       bne cloop
280 ; connect input stream to file 1
290       ldx #$01        ; logical file 1
300       jsr $ffc6       ; chkin
310 ; main loop.  ead char, analyze, count
320 mloop jsr $ffe4   ; get char
330       and #$7f    ; strip high bit
340       cmp #$30    ; test if text char
350       ror a       ; c flag to high a
360       tax         ; make copy in x
370       eor ctype   ; match prev char
380       stx ctype   ; stash new char type
390       and ctype   ; flag start-word
400       asl a       ; move status to cflag
410 ; add zero (plus carry) to counter
420 ; do this in decimal mode
430       sei         ; for pet/cbm
440       sed         ; set decimal mode
450       ldx #$00
460 ; loop to add to 3-byte counter
465         ror a     ; preserve c flag
470 aloop   rol a     ; restore c flag
475         lda count,x
480         adc #$00
490         sta count,x
495         ror a     ; preserve c flag
500         inx
510         cpx #$03
520         bne aloop
530 ; clear decimal mode
540       cld         ; clear decimal mode
550       cli         ; for pet/cbm
560 ; test end of file, loop back
570 ; for pet/cbm, change $90 to $96
580       lda $90     ; check st variable
590       beq mloop   ; if not end, loop
600 ; end of file; disconnect input stream
610     jsr $ffcc
620 ; print the (decimal mode) counter
630     ldx #$02            ; three times
640 ploop lda count,x   ; get two digits
650       jsr prtwo     ; print them
660       dex           ; count down
670       bpl ploop
680 ; finished.  return to basic.
690     rts
700 ; subrtn to print two decimal digits
710 ; first extract and print high nybble
720 prtwo pha         ; save the byte
730       lsr a       ; move high nybble..
740       lsr a       ; .. down to ..
750       lsr a       ;   .. low nybble
```

```
760         lsr a
770         ora #$30    ; change to ascii
780         jsr $ffd2   ; print it
790         pla         ; restore the byte
800 ; now extract and print low nybble
810         and #$0f    ; get low nybble
820         ora #$30    ; change to ascii
830         jsr $ffd2   ; print it
840         rts         ; return from sub
850 ;       end
```

Gazette, June 1994

PROGRAMMER'S PAGE: Screen Fun

By David Pankhurst

Screen stuff is what we're going to look at this month. Although some of the programs submitted by readers have practical use, we're going to ignore that aspect in favor of nifty effects, funny displays, and amusing tricks.

STATIC TV

The latest thing in the IBM world is turning your $400 monitor into a $150 color television. We can do that on the Commodores without spending a cent! Hong Pham of Antigonish, Nova Scotia, Canada, wrote a program that makes your computer imitate late night TV. Save hundreds of dollars! (To stop it, press Run/Stop and Restore.)

```
10 REM  STATIC TV
20 REM  BY HONG PHAM
30 REM
40 V=53248:S=54272:Z=16384:E=Z+111
50 FORI=ZTOE:READA:POKEI,A:CK=CK+A:NEXT
60 IFCK<>14422THENPRINT"ERROR IN DATA STATEMENTS!":STOP
70 FORI=STOS+24:POKEI,0:NEXT: POKE56334,0
80 POKEV+32,0:POKEV+33,0:POKEV+34,11:
POKEV+35,12:POKEV+24,29:POKEV+22,216:SYSZ
90 DATA162,25,189,86,64,157,255,211,
202,208,247,169,9,157,0,216,157,0,217,157
100 DATA0,218,157,232,218,232,208,241,
169,4,141,45,64,160,4,205,27,212,240,251
110 DATA173,27,212,157,0,0,232,208,242,
238,45,64,136,208,236,169,48,141,75,64
120 DATA160,8,205,27,212,240,251,173,
7,212,77,18,208,157,0,0,232,208,239,238
130 DATA75,64,136,208,233,240,224,0,8,
0,0,129,16,255,0,26,0,0,129,16,255,255
140 DATA255,0,0,129,16,255,0,0,0,15
```

BORDER FLASH

The following is practical, but don't worry. You'll enjoy it, too. James Jones of Klondike, Texas, wanted to have the computer do something to let you know that it's still working during long pauses. The result is a little interrupt patch that repeatedly flashes the border.

The great thing about this program is that it runs in the background, a constant reminder your computer has important things to do besides waiting for you! SYS 686 starts it and SYS 679 ends it. Line 70 is a ten-second demonstration of the routines. Line 50 is the count in sixtieths of a second, so poking 30 to 708 would make the border flash every half a second.

```
10 REM  BORDER CHANGE
20 REM  BY JAMES JONES
30 REM  SYS 686 STARTS; SYS 679 ENDS
40 C=0:FORI=679TOI+36:READ D:POKE I,D:C=C+D:NEXT
50 POKE708,45:REM # JIFFIES TILL CHANGE
60 IF C<>4135 THENPRINT"ERROR IN DATA STATEMENTS":STOP
70 SYS 686:FORI=1TO10000:NEXT:SYS679
80 DATA120,169,49,160,234,208,5,120,
169,187,160,2,141,20,3,140,21,3,88,96,206
90 DATA203,2,208,8,238,32,208,169,60, 141,203,2,76,49,234,1
```

SCREEN WIPES

Although machine language can do some fancy things, let's not forget BASIC. Brian Kissinger on Evansville, Indiana, presents six ways to clear the screen. The following program demonstrates each of these screen wipes by filling the screen with text. You then choose a number from 1-6 to see each in action.

The program is simply a vehicle to demonstrate the effects. In reality, each wipe is a one-line routine. Lines 40, 50, 60, 70, 80, and 90 can each be popped into your own programs for a special effect. Note, however, that after clearing, each routine then prints the clear screen code 147. Make sure you include this code to reset the screen properly.

To see other effects, simply type RUN again and select a different number.

```
10 REM  SIX SCREEN WIPES
15 REM  BY BRIAN KISSINGER
20 PRINT"[CLR]";:FORJ=0TO29:PRINT"THIS IS A VERY CLUTTERED SCREEN
";:NEXT
25 PRINT"[HOME][RVS ON]ENTER 1-6[RVS
OFF]":WAIT198,7:GETA$:ONASC(A$)-48GOTO40,50,60,70,80,90:GOTO25
35 REM 1 - WIPES SCREEN UPWARD
40 FORJ=24TO0STEP-1:POKE781,J:SYS59903: NEXT:PRINTCHR$(147):END
45 REM 2 - WIPES SCREEN DOWNWARD
50 FORJ=0TO24:POKE781,J:SYS59903: NEXT:PRINTCHR$(147):END
55 REM 3 - WIPES TOP/BOTTOM TO CENTER
60 FORJ=0TO12:POKE781,J:SYS59903:POKE
781,24-J:SYS59903:NEXT:PRINTCHR$(147):END
65 REM 4 - WIPES CENTER TO TOP/BOTTOM
70 FORJ=12TO0STEP-1:POKE781,J:SYS59903:
POKE781,24-J:SYS59903:NEXT:PRINT"[CLR]":END
75 REM 5 - WIPES LIKE CLOSING BLINDS
80 FORJ=0TO4:FORX=0TO4:POKE781,X*5+J:
SYS59903:NEXT:NEXT:PRINTCHR$(147):END
85 REM 6 - WIPES LIKE OPENING BLINDS
90 FORJ=1TO5:FORX=1TO5:POKE781,X*5-J:
SYS59903:NEXT:NEXT:PRINTCHR$(147):END
```

If you add a a line such as 100 GOTO 10, you can run the program

repeatedly, trying out different effects by selecting different number 1-6.

VIRUS SIMULATOR

S. Bala Gangdher Rao of India has created a screen effect that makes you think your computer has a virus. It randomly sends each character flying down to the bottom, taking others with it. For best effect, run this one with lots of text on the screen. Trust me, the screen eventually will be cleared, although you may have to wait a while for it.

```
10 REM  VIRUS SIMULATOR
20 REM  BY S. BALA GANGADHER RAO
30 Y=PEEK(646)
40 B=1024+RND(1)*999:C=B+54272
50 X=PEEK(B):IFX=32THEN40
60 POKEB,32:B=B+40:C=C+40
70 IFB<2024THENPOKEB,X:POKEC,Y:GOTO60
80 GOTO 40
```

FAKE SCREEN

On the subject of screen tricks, I thought I'd put one of mine in here. When you run it, you get the initial startup screen, complete with flashing cursor. Everything looks normal, and you can enter text normally. The difference is that nothing happens when you press Return following a command. You can have hours of fun confusing your friends with this one. For more fun, add the following line.

```
65 IF RND(1)<.3 THEN 60
```

When printing, this will skip about 30 percent of the characters typed. This is guaranteed to make people think their computer is broken!

```
10 REM C64 SCREEN SIMULATOR
20 REM
30 PRINT"[CLR][DOWN]     **** COMMODORE 64 BASIC V2 ****"
40 PRINT"[DOWN] 64K RAM SYSTEM  38911 BASIC BYTES
FREE":PRINT"[DOWN]READY.":POKE205,2
50 POKE204,0:POKE207,1:POKE788,52
60 POKE205,1:POKE204,0:WAIT198,7:GETX$: POKE205,2:WAIT207,1:POKE204,1
70 PRINTX$;:GOTO60
```

BOXES

For pure mindless entertainment, I like random patterns onscreen. Greg Waggoner, of Olney, Texas, sent in a routine to create a box. It's quite practical if you need to draw boxes. It's fast, compact, and robust. But we aren't concerned with practical here. It also makes a great screen display. In this demo, lines 430-510 draw boxes of random sizes and colors. Lines 480 and 500 are to fix two problems that can

crop up: a box being displayed in the background color (making it invisible) and a box touching the bottom right corner (which would scroll the screen).

```
100 REM  ML BOX DRAWING
110 REM  BY GREG WAGGONER
120 REM
130 REM ----------FORMAT----------------
140 REM   SYS 49152,X1,Y1,X2,Y2
150 REM
160 REM  WHERE X1 AND Y1 ARE THE UPPER-
170 REM  LEFT COORDINATES. X1 RUNS FROM
180 REM  0-36, AND Y1 FROM 0-21. X2 AND
190 REM  Y2 ARE THE BOX LENGTH AND
200 REM  HEIGHT, RESPECTIVELY. THEY ARE
210 REM  DESCRIBED AS FOLLOWS:
220 REM    X2 > 1 AND X2 < 39-X1
230 REM    Y2 > 1 AND Y2 < 24-Y1
240 REM
250 DATA32,241,183,142,167,2,32,241,183
260 DATA 142,168,2,32,241,183,142,169,2
270 DATA 32,241,183,142,170,2,174,168,2
280 DATA172,167,2,24,32,240,255,169,176
290 DATA 32,210,255,162,0,169,96,32,210
300 DATA 255,232,236,169,2,208,245,169
310 DATA 174,32,210,255,162,0,142,171,2
320 DATA 238,168,2,238,171,2,174,168
330 DATA 2,172,167,2,24,32,240,255,169
340 DATA 98,32,210,255,162,0,169,29,32
350 DATA 210,255,232,236,169,2,208,245
360 DATA 169,98,32,210,255,174,171,2
370 DATA 236,170,2,208,209,238,168,2
380 DATA 174,168,2,172,167,2,24,32,240
390 DATA 255,169,173,32,210,255,162,0
400 DATA169,96,32,210,255,232,236,169,2
410 DATA 208,245,169,189,32,210,255,96,
420 FORI=49152TO49298:READD:CK=CK+D: POKEI,D:NEXT
430 PRINT"[CLR]":C=PEEK(53281)AND15: IFCK<>20639THENPRINT"ERROR IN
DATA":STOP
440 X1=INT(RND(1)*37)
450 X2=INT(RND(1)*(36-X1))+2
460 Y1=INT(RND(1)*22)
470 Y2=INT(RND(1)*(21-Y1))+2
480 IFX1+X2>37ANDY1+Y2>22THEN440
490 SYS49152,X1,Y1,X2,Y2
500 X=INT(RND(1)*16):IFX=CTHEN500
510 POKE646,X:GOTO440
```

As I mentioned, this routine is also practical. The values used in the SYS command are the coordinates of the top left corner and the length and width of the box. Be careful not to use invalid range values (see lines 100-240), or unpredictable results can occur.

CHARACTER EFFECTS

Another great screen effect is from David Gunderson of McCook, Nebraska. He has written three routines that move the character set to RAM and then randomize it, erase it, and restore it. The effects are impressive. What this program does is show the effects off by randomly calling each of the three. A random delay is inserted at line 240, to make the display more pleasing. Play with the fractions on lines 210 and 220 to determine the percentage of time each routine is called. And to really see the effects, make sure there's a lot of text on screen.

```
100 REM  CHARACTER EFFECTS
110 REM  BY DAVID GUNDERSON
120 REM
130 REM SYS 49152 MAKES TEXT REAPPEAR
140 REM SYS 49269 SCRAMBLES TEXT
150 REM SYS 49361 MAKES TEXT DISAPPEAR
160 POKE52,48:POKE56,48:CLR:REM SAVE MEM FOR CHARACTERS
170 S=49152:IFPEEK(S+287)*PEEK(S+2)= 21120THEN190
180 FORK=STOK+287:READX:C=C+X:POKEK,X: NEXT:IFC<>40352THENPRINT"DATA
ERROR":STOP
190 POKE53272,12+(PEEK(53272)AND240)
200 SYS S
210 IFRND(1)<.4THENSYS S:GOTO240
220 IFRND(1)<.6THENSYS S+117:GOTO240
230 SYS S+209
240 FORI=1TORND(1)*1999:NEXT:GOTO210
250 :  MAKE TEXT APPEAR
260 DATA173,14,220,41,254,141,14,220
270 DATA169,255,141,14,212,141,15,212
280 DATA169,128,141,18,212,169,5,141
290 DATA233,3,134,252,134,254,160,0
300 DATA169,48,133,253,169,208,133,255
310 DATA162,0,173,27,212,141,233,3
320 DATA73,255,141,234,3,165,1,41
330 DATA251,133,1,177,252,45,234,3
340 DATA141,232,3,177,254,45,233,3
350 DATA13,232,3,145,252,209,254,240
360 DATA2,162,1,165,1,9,4,133
370 DATA1,200,192,0,208,204,230,253
380 DATA230,255,165,253,201,56,208,194
390 DATA224,0,208,180,173,14,220,9
400 DATA1,141,14,220,96
410 :  MAKE TEXT SCATTER
420 DATA173,14,220,41,254,141,14,220
430 DATA169,5,141,233,3,169,255,141
440 DATA14,212,141,15,212,169,128,141
450 DATA18,212,162,250,160,0,169,48
460 DATA133,253,132,252,173,27,212,41
470 DATA7,141,232,3,238,232,3,169
480 DATA0,42,206,232,3,208,250,141
490 DATA233,3,177,252,77,233,3,145
```

```
500 DATA252,200,192,0,208,222,220,253
510 DATA165,253,201,56,208,214,232,224
520 DATA0,208,201,173,14,220,9,1
530 DATA141,14,220,96
540 :  MAKE TEXT DISAPPEAR
550 DATA173,14,220,41,254,141,14,220
560 DATA169,250,141,233,3,169,255,141
570 DATA14,212,141,15,212,169,128,141
580 DATA18,212,160,0,169,48,133,253
590 DATA132,252,162,0,173,27,212,141
600 DATA233,3,177,252,45,233,3,145
610 DATA252,240,2,162,1,200,192,0
620 DATA208,234,230,253,165,253,201,56
630 DATA208,226,224,0,208,212,173,14
640 DATA220,9,1,141,14,220,96
```

Virus Simulator is the program that loads from the "Programmer's Page" menu. Be sure to run this one again when there's a lot of text on the screen.

All of the programs mentioned in this column, however, can be found on the flip side of this disk. Load and run them as you would any BASIC program. The filenames are as follows.

STATIC TV	SCREEN WIPES
BOXES	FAKE SCREEN
BORDER FLASH	CHAR CHANGER
VIRUS SIM	

If you have a programming tip or trick that you think others might enjoy, send it on disk to Programmer's Page, COMPUTE's Gazette Disk, 324 West Wendover Avenue, Suite 200, Greensboro, North Carolina 27408. We pay $25-$50 for each tip that we publish.

Gazette, June 1994

GEOS: In Search of the Perfect Driver

By Steve Vander Ark

My printer vanished! It was my beautiful 24-pin Epson LQ-500 printer. Well, maybe it didn't exactly vanish, but it disappeared from my computer room and turned up next to the PC clone that my wife uses. She teaches kindergarten, and she uses that computer for all her schoolwork. The printouts she gets from GEOS Ensemble and the Epson are stunning, and she likes for her letters to parents and whatnot to look as perfect as possible. I gave the good printer to her.

Suddenly, I found myself looking for a printer. I had forgotten how confusing that can be. I was glad I had read as much as I had about printers. I learned one thing quite a few years ago: I wanted Epson compatibility. That narrowed the field somewhat, so when a used Epson turned up for a good price, I snapped it up. I am now the proud owner of an Epson Spectrum LX-80 9-pin printer.

Now that I had my "new" printer, I rediscovered another interesting problem: I needed a GEOS printer driver for it. Of course, I wanted the very best one there was. I know that many of you have wrestled with this problem because I've fielded quite a few questions about it. But I haven't had to think about my own printer drivers for a long time. I checked my original system disk, and sure enough, there was a driver for the LX-80. That was the easy answer; I loaded it into my system.

Let me just stop a minute here and talk about printer drivers. This is a subject many people find confusing, so let me try to explain just what those files are and how they work.

Printer drivers are little programs, usually only one or two kilobytes in length, to which GEOS refers every time you tell the system to print something. Every printer type has its own set of codes that tell the machine what to do: skip a line, print the next characters in italics, switch to uppercase. In other words, these codes tell the printer how to turn out a document the way you want it.

GEOS needs to know these codes if it's to create the page as you've designed it, and it gets them from the printer driver program. The printer driver you use must have the right codes for your printer, or you're likely to be disappointed with what the printer prints--if it prints at all.

The printer driver is one of those things that you need to set up when you start using GEOS for the first time. You choose Select Printer from the GEOS menu on the deskTop and then indicate your printer from the file requester box. The driver for your printer is then installed. This means that it will automatically be in effect every time you boot GEOS from that boot disk. If you are using GEOS 64, you have to have your printer driver available on any disk from which you plan to print, but not so with GEOS 128. From then until you change it, GEOS

will use the codes in that printer driver to run your printer.

As I said, there was a driver for my LX-80 on my GEOS boot disk. So I installed it, and I was ready to print. That was OK for starters. Often, there are better drivers to be had on GEnie and QuantumLink, drivers that create better-looking printouts.

I decided to start searching. I hoped to find at least a double-strike driver, maybe even a Lasermatrix or quadruple-strike driver. If I was really lucky, I would find a driver by George Wells, the printer driver wizard on Q-Link. The drivers that George created for 24-pin printers are clearly the best around. But I knew that the LX-80 was an old model printer, and I was afraid that the snazzy drivers wouldn't work with it.

To my delight, I found out that my LX-80 is compatible with the FX-80 printer, which is something of a standard for Epson-compatible 9-pin printers. There are a lot of great drivers around for the FX-80. I logged onto Q-Link and started searching the GEOS libraries. It didn't take long to track down quite a few possibilities. Files that small take almost no time to download from network, so I downloaded them all. I wanted to find the absolute best printout I could.

I started with double-strike drivers. They often have a DS tacked onto the name. That means that each pattern of dots is printed twice. On the second printing, the printhead is offset just a little so the patterns blend together to help fill in the jaggies.

Double-strike drivers offer quite an improvement over the normal kind. I also found some quadruple-strike drivers (yep, they have QS tacked onto their names) that really filled in the rough spots. These were very nice drivers, and I was almost ready to stop there.

I looked at some even fancier drivers. I found interpolating drivers, such as the Lasermatrix drivers (they have LM on the end of the name). These drivers really do a nice job of printing but tend to round things out quite a bit. I even found drivers that printed a full page in less than a full page of space. This means that the dots are pressed much closer together. The print quality of such a compressed image is superb, but I really needed something that would give me full pages.

After poking around for some time, I found a wonderful driver written by--you guessed it--George Wells. This driver, called EP8PIN3PASS, produces a curved line with barely a jiggle. Unfortunately, It tends to add squiggles to italics, but since I seldom use italics, I figured that was acceptable. (I prefer to switch to an italics font instead of letting the computer italicize text.) In the end, EP8PIN3PASS is the driver that I installed for most of my work.

Obviously, if all I wanted were great printouts from geoWrite only, the best printer driver would be GEOS LQ. This driver comes in the Perfect Print package distributed by Creative Micro Designs. GEOS LQ

offers extremely clean geoWrite printouts, much better than anything else for GEOS.

Perfect Print disks also include fine printer drivers for other GEOS applications. These are called High Quality drivers. (Guess what they have on the end of their names? You got it: HQ.) As far as I can tell, the printouts from the HQ drivers are as good as EP8PIN3PASS.

Once I had settled on a printer driver, I discovered something else. This printer does labels. That may seem obvious to you, but my 24-pin printer, for all its wonderful features and beautiful built-in fonts, wouldn't do labels. Now I have all sorts of possibilities available to me. I wonder what GEOS might be able to do with labels. Next month I'll let you know some of things I've discovered.

Gazette, June 1994

PD PICKS: File Drawer and Super-Alarm III

By Steve Vander Ark

I talked to my editor the other day. He talked to me, really. In his gentle way he pointed out that it had been a heck of a long time since I wrote about anything other than games in this column. You have to know Tom Netsel; he is the nicest guy in the Commodore world. So what he was really saying was, "Lose the joystick and get to work!"

He didn't have to tell me twice. I grabbed my QuantumLink disk and logged on. I pointedly avoided all those game libraries and headed straight for the applications section. Now I had a lot of decisions to make. After all, if you want to do some serious work with a Commodore, there are literally hundreds of programs on Q-Link to help you out. The applications library includes separate sections for different categories, such as Home Applications, Business Programs, and so on. I decided that I'd start with Home Applications and see what I could turn up.

What I discovered was that I had a lot more choices to make. Under the heading of Home Applications I found areas with names like Budgets and Checkbooks, Mailing Lists and Labels, and Database Managers. There is a lot of great stuff there. I decided to check out the databases; that seemed like a fairly straightforward task. Turns out that even this choice left me with plenty of files to wade through. The database library is huge.

I downloaded a few and tried them out. Several were designed for a specific task like keeping track of videotapes or baseball cards. I'm as lazy as the next guy, so finding a database that's been designed for a specific job sounds pretty good. For this column, however, I decided to go with a more general database, one that lets users design things the way they want.

Once I narrowed the field of my search, I decided to look for something a little out of the ordinary. One excellent way to track down good programs on Q-Link is to look for asterisks after the names. The librarians of the various areas add these to indicate a program that they particularly like. In some libraries, the graphics libraries for example, you'll find files marked with two, three, or even four asterisks. You know that a file with three or four stars will be spectacular. The application libraries, on the other hand, stick with a single star. Anyway, tucked away in the Home Applications library is a file with an intriguing name and a star. I just had to download it and see what it did. It certainly lives up to its billing, and I think you'll like this and the other program on this month's disk.

Here are the details on these two excellent programs.

FILE DRAWER
By Stan Takis

QuantumLink filename: file drawer 4.5. Uploaded by StanT2.
GEnie file number 14456.
Shareware: $2.00.

Yes, the shareware fee on this program is a mere two bucks. Don't let that fool you into thinking this is a lightweight program, though. If you need to create a simple database and want speed and some neat features, that two bucks will buy you the best deal in town.

When you start File Drawer, a text prompt asks you to decide how many fields you need. Fields are the various spaces or categories into which you want to put information. You are then asked to give them names.

For the database I was designing, a listing of GEOS programs, I created five fields: title, author, associated application, file type, and description. Once the program received that information from me, there was a brief pause while File Drawer created the database. A few seconds later there was a new program on my disk.

File Drawer creates a separate program, a new database, which you can run on its own. So load File Drawer and save it to another formatted disk before you try to use it. It won't work on the write-protected Gazette Disk. You can create a different self-contained database for each application that you have.

File Drawer's menu of commands is basic but complete. You can add new information, naturally, or edit existing data. You can sort by any of the fields alphabetically (not numerically) and print out selected portions of the database as well. You can't add a whole new field, though, so it's important that you plan ahead and decide what fields you'll be needing. Aside from that minor limitation, you'll find that File Drawer will nicely keep track of just about anything you can think of.

SUPER-ALARM III
By Kirk Mook
Q-Link filename: super-alarm iii. Uploaded by Layoric.
GEnie file number 14457.

There's practically nothing the good old Commodore can't do. Here's a great example for you. Super-Alarm is an alarm clock that'll give your old clock radio a run for its money.

Every alarm clock has a required annoying wake-up sound, and Super-Alarm has that. If you crank up the volume on your monitor or TV, this alarm is guaranteed to set you bolt upright in bed.

Super-Alarm also has the option of displaying the time on the screen. Unfortunately, it displays in 24-hour military format only. But, hey, if it couldn't do that much, I wouldn't bother writing about it.

The real charm of this program is its music option. You can tell

Super-Alarm to play any music file you want at wake-up time. It recognizes files with the .MUS extension, the kind normally referred to as SID files. For someone like me who collects SID files by the hundreds, that's pretty cool. Some of my favorite versions of songs are those my Commodore plays. Super-Alarm will also function as a player for your music collection, playing all the .MUS files on the disk one after the other.

If that weren't enough, you can also instruct Super-Alarm to start any program that you want at any time. If you know how to program the sound chip, for example, you could write your own alarm routine, complete with custom sound effects.

There's no reason why you have to use this program only to drag yourself out of bed. There are plenty of ways you could use a timer to start a program. If you run a part-time BBS, for example, you could use Super-Alarm to start it up at a given time, even if you weren't home.

One last note. Unlike many public domain or shareware programs, Super-Alarm includes documentation that's built right into the program. You can read it from the menu. I sure like that, especially when I come back to use a program after a month or so, and I've forgotten half the commands and can't find my notes.

Well, there you have it, two perfectly nice application programs. Nothing blowing up; no weird creatures flying all over the screen. And you know, there are really a lot of other great files in these libraries. I think I'll make Tom happy and write up a couple more for next month. If you have any special requests for different types of programs, drop me a line in care of Gazette. See you next month.

Gazette, June 1994

GETTING STARTED IN MACHINE LANGUAGE

By Jim Butterfield

Programs containing machine language are a mystery to the beginning computer experimenter. It seems as if you can't list such programs and the rules for writing them are not widely known, and yet these programs often run at dazzling speed.

Machine language and its cousin, assembly language, aren't hard. But you'll need to learn new skills, such as how to deal with binary and hexadecimal numbers. You'll need new tools, principally a machine language monitor program. And you'll need new information, not only about machine language itself but also about how to tap into the workings of your computer.

Several benefits come from machine language skills. You can write programs that run amazingly fast. You can do things that BASIC cannot. Perhaps most importantly, you'll get an insight into the workings of your computer.

THE TOOLS

To begin, you'll need a machine language monitor (MLM) program. This program helps you to examine and change memory, and it even has a tiny assembler and disassembler built in. (More about these later.) If you have a Commodore 128 or the rarer Plus/4, there's a machine language monitor built into it. Just type MONITOR to get there.

A machine such as the Commodore 64 needs to have the MLM program loaded. Several such programs are available on bulletin boards and user groups. You'll find a monitor that I wrote called SuperMon on this disk.

You also need reference material: information about the instructions, about the addressing modes, and about how to call the Commodore operating system. Some of this material is provided here. You may also be able to find books that will serve as reference data.

A working knowledge of number systems, especially hexadecimal and binary, is valuable. Many machine language instructions deal directly with the bits of memory: the shifts and rotates and the logical operations. You'll understand these better if you dig into number systems.

As for hexadecimal: For the moment, you can start with the idea that the dollar sign ($) signals a hex number and that the letters A to F are valid numeric digits. So $AC is just as legitimate a number as 45, and $BEAD could refer to a legal address somewhere in memory.

Bring your wits, too. You're learning new skills, and you'll need patience and attention to detail.

REGISTERS
The working part of your microprocessor chip contains storage areas called registers. Three of these are available to hold data. These three registers are called A, X, and Y; each can hold a single byte.

The A register is sometimes called the accumulator, since the processor does much of its arithmetic there. The X and Y registers are often called index registers. The contents of index registers are often used to tweak an address, allowing an instruction to reach any one of a range of memory locations.

Most of a computer's work takes place in the registers. You bring the data in from memory, work on it, and then take it back out and store it.

FIRST DATA INSTRUCTIONS
You can load data into A, X, or Y. The instructions are abbreviated LDA (Load A), LDX, and LDY. You can load information from anywhere in memory or just supply a value (immediate addressing).

You can store data in memory, copying it from A, X, or Y. The instructions to do this are STA (Store A), STX, and STY.

The A register is a handy place for addition and subtraction. ADC is Add-with-Carry; SBC is Subtract. Values in X or Y can be bumped up or down. INX (Increment X) and INY make the register's contents one higher; DEX (Decrement X) and DEY make the contents one lower. Keep in mind that a register only holds a value that ranges from 0 to 255 (hex 00 to FF). If you go past the limit, you wrap around to the other end.

FLOW CONTROL
A machine language program executes instructions in the order that they are stored in memory. You can change this with a Branch or Jump that takes you to a new location.

There are eight Branch instructions, all conditional. Depending on the results of a test, these instructions will cause your program to hop forward or back by up to about 128 bytes. If the test fails, the Branch won't take place, and the program continues with the next instruction. The branch instructions check flags, whose conditions are set by previous instructions.

Jump (JMP) will transfer your program's execution point to anywhere in memory.

Return from Subroutine (RTS) takes a program back to wherever it was called from. In your main machine language program, it will take you back to the BASIC program that called it.

Jump Subroutine (JSR) also takes you anywhere, but it leaves a link that allows the subroutine to return and pick up where it left off. It's like BASIC's GOSUB and RETURN commands. JSR is an important

command for calling your computer's operating system. For example, JSR $FFD2 asks the system to output whatever character is currently stored in A.

SYSTEM CALLS

There are dozens of system calls provided by the operating system. To begin, you need learn only two: address $FFD2 (CHROUT) and address $FFE4 (GETIN).

A call to $FFD2 causes whatever is in the A register to be sent to the output, normally the screen. When you make a call to this address, you get a bonus in that the contents of all three data registers are carefully preserved. A, X, and Y will not be changed.

A call to $FFE4 causes a character to be drawn from the input stream, normally the keyboard buffer. The character will be placed in the A register. The subroutine returns immediately. If no character is received, A gets a value of 0. This subroutine might change the contents of all three data registers. The incoming data will arrive into A, but X and Y might change from their previous values, too.

Eventually, you can expand your catalog of calls to six, as shown in the accompanying table. These six will do almost all the work you are likely to need.

LET'S DO IT

We can walk through the creation of a simple program, step by step. As we do so, I'll try to make holistic comments on the coding, the tools, and the computer system.

Be sure your machine language monitor is installed and ready to go. Have it active by typing MONITOR, or SYS 8, or whatever the documentation calls for. If you're using SuperMon, simply load it and type RUN.

When you start the MLM, you'll see a register display on your screen. You should recognize three registers: AC, XR, and YR. Their contents, shown below the titles, don't matter at the moment.

Right now, our objective is to write a program that lets us touch a key on the keyboard and have that key echo many times to the screen.

We'll put this program in memory starting at address 8192, or hexadecimal $2000. There should be memory space available there on almost any model of Commodore 8-bit computer.

FIRST STEP

First, our program needs to get a key from the keyboard. Recall the subroutine call to system address $FFE4, GETIN. The coding is JSR $FFE4, Jump Subroutine to address $FFE4, and we want to put it at address $2000. To do this, enter the following line.

```
A 2000 JSR $FFE4
```

The A stands for Assemble. That is, change this mnemonic instruction into real machine code. With some monitors, such as the one built into the 128, the moment you press Return, the line will automatically change to the following.

```
A 2000 20 E4 FF JSR $FFE4

A 2003
```

The A 2000 is familiar, but the next part is new. The 20 E4 FF is the instruction as it sits in memory. Three bytes, three numbers, and they mean JSR $FFE4. If you puzzle over it awhile, you might figure out that 20 (hex, of course) must means JSR, and the address has turned itself around. Don't worry, the computer knows what it's doing.

The computer has also written part of the next line for you. To save you from arithmetic mistakes, it has counted off those three bytes and is printing for you the correct address for the next instruction, complete with A for Assemble.

Your first instruction has been placed into memory. What do we need to do next?

ANYBODY HOME?

When the program runs, the first instruction asks for an input character. If there's one waiting, it will be placed in the A register; if no characters are waiting, A will contain a 0 byte. Let's test for this; if A contains 0, we'll go back and try again. Here are the next two lines.

```
A 2003 CMP #$00
A 2005 BEQ $2000
```

CMP stands for Compare (the A register). The # symbol means actual value, so the first instruction is comparing the A register against the actual value of 0.

On the next line, BEQ is Branch if Equal. It's testing the result of the comparison. If the contents of A equal 0, the program will branch back to address $2000 and try again for an input character.

If the contents of A don't equal 0, the program has a valid character. Our next task is to print that character several times. As your screen will show you, this code will start at address $2007.

START THE PRINT LOOP

I suggest that we print the character 40 times. That's a nice round number that matches the width of the screen. If you happen to be working with an 80-character screen, you can substitute 80 if you like.

We can use X or Y to do the counting, so I'll pick X. The first thing

to do is to set the counter (Load X) with a value of 0.

```
A 2007 LDX #$00
```

Note that once again we use the # character to say that we want the actual value 0 to be loaded, not the contents of address 0. This is called immediate mode addressing.

The character is still in A, so we can proceed to print it. The call is to CHROUT at $FFD2. No need to worry about disturbing the count value in X; this subroutine will preserve the register values.

```
A 2009 JSR $FFD2
```

By the time our running program gets to $200C, the character has been printed once. Count it by adding 1 to the contents of the X register.

```
A 200C INX
```

If the count in X has not yet reached 40, we can go back and print some more. We must test X for 40 ($28 in hex). That's a Compare-X instruction, of course, and loop back with a Branch not Equal.

```
A 200D CPX #$28
A 200F BNE $2009
```

Can you see why we must branch back to address $2009 and not to $2007 or $200C?

As I said, that $28 is the hexadecimal equivalent of decimal 40. Most MLM assemblers will allow you to code CPX #+40, where the + symbol indicates a decimal value. Try it and see if it works on your machine.

By the time the program gets to $2011, it has received a character and printed it 40 times. You might like to do it again, but we must have a way to stop.

Let's do it this way. If the character we have just printed is not an asterisk (value $2A), we'll go back and do it all again. But if it is an asterisk, the program will terminate with an RTS and go back to BASIC.

Here's the instruction. Compare A with value $2A, Branch-not-Equal to $2000.

```
A 2011 CMP #$2A
A 2013 BNE $2000
A 2015 RTS
```

That's the end of our program, but the MLM doesn't know that. It will prompt you again with a line reading A 2016. Just strike Return to cancel it.

Before you leave the MLM, you might like to look over your program one last time. Use the command D for Disassemble.

D 2000 2015

You'll see the whole program listed. If anything looks wrong, you can simply move the cursor up and type over it to correct the problem.

BACK TO BASIC
Leave the machine language monitor by typing X. You'll see the familiar READY response after you exit the monitor.

The program is in memory, ready to go. The BASIC SYS command is something like a Subroutine call. To activate it, use the following command.

SYS 8192

Everything will seem quiet at first, but try typing a few words on the keyboard. How's that for speed typing? This example should work, with slight variations depending on the monitor you use, on most Commodore machines.

CONCLUSION
Machine language may be new territory for you, but it doesn't have to be mysterious. Each step is logical and simple. Take a little time, give it some effort, and you too can do it.

Gazette, June 1994

MONITORS--GATEWAY TO MACHINE LANGAUGE

By David Pankhurst

In this article, I'll be discussing monitors--but not the type that you're probably looking at right now to read this article. I'm talking about software you need for programming in machine langauge.

Monitors are a must for serious assembly language work. Commodore 128 owners have a monitor built into their machines, and it's summoned with the command MONITOR.

If you own a 64, you'll have to load a monitor just as you would any other program. There are several monitors available for the 64. Some popular ones are MicroMon, CBM Mon, and SuperMon. (Look for SuperMon on the flip side of this disk.) Depending on the type of monitor you have, once you've loaded it into memory, you start it with either a SYS command (MicroMon) or by typing RUN (SuperMon).

Unlike the free-form programming of BASIC, monitors expect input to follow strict guidelines. All commands are prefixed by a period, followed by a letter or character and a single space.
The various text and numbers following the command, called its parameters, are separated by a comma or space. If the data is numeric, it must be a four-digit hex number for addresses or a two-digit number for everything else. Pad with 0s if necessary. Because of its banking system, the 128's monitor uses five-digit numbers for addresses. Mistakes are sometimes flagged with a question mark, or they may simply be ignored.

For programming flexibility, 64 monitors reside in various memory locations, typically at the top of BASIC or $C000-$CFFF. The latter can be a problem, since that area is a favorite for placing code. Monitors come in two versions for different locations. They may be automatically relocatable (like SuperMon), or else they will have instructions for relocating.

The commands for SuperMon are shared by all monitors, although syntax may differ somewhat. Here are some examples.

.A C000 LDA#$56 - Assemble
After .A comes the address at which to assemble the code. The address is followed by the assembler mnemonic, followed by any appropriate data. Enter one instruction per screen line, and follow the syntax exactly. Include the dollar sign ($) when entering two- or four-digit hex numbers. If the code assembles correctly, you are prompted on the following line with the next available assembly address.

.D 5600 - Disassemble
The Disassemble command displays a screen full of assembly code, starting at the input address. Where data cannot be converted to a valid code, question marks are displayed.

.F D800 DBFF 00 - Fill
The first two parameters in the Fill command are the starting and ending addresses to fill, followed by the byte to store. (This example would set color memory to black.) Be careful with this command, because you can easily overwrite valuable places in memory such as the monitor itself or page 0.

.G C000 - GOTO
This runs your program, with the starting address as the only parameter. At the end of the program, an RTS instruction returns you to BASIC, and BRK starts the monitor.

.H A000 BFFF 53 51 - Hunt
This line hunts for matching characters in memory. The first two values are the memory range to examine, and the characters following are the bytes to look for. One or more bytes can follow, each with a space between. In this example, BASIC ROM is scanned for bytes $53 $51, which begin the keyword SQR.

.L "filename",08 - Load
The syntax for loading can differ widely from monitor to monitor. For SuperMon, "filename" will be reloaded from disk to the location from which it was saved. (See the section on Save.)

.M C000 C040 - Memory
This command displays the range of memory as hex bytes, eight per line. Cursoring up and changing values (with Return) pokes them to memory. The Run/Stop key can be used to stop a lengthy listing.

.R - Registers
All the CPU registers are displayed and can be changed by cursoring to them, editing, and pressing Return. These become the startup values used by GOTO.

.S "filename",08,C000,C200 - Save
The filename and device in a Save are identical to the Load command; the range of memory to save follows. When loading, the file is reloaded to this location in memory. The saved ending address should be one more than necessary; in this example, all data including that at $C1FF would be saved, but not $C200.

.T C300 C450 CD00 - Transfer
The first two parameters are the start and end of the memory block to transfer. The next number is the destination starting address. In this example, the byte at $C300 is copied to $CD00, the next at $C301 to $CD01, and so on. As in the Fill command, care is needed, since transferred data can overwrite important memory locations.

.X - Exit
You can quit the monitor with this command. To reenter the monitor, execute a BRK instruction (hex $00). This can be done by typing SYS 13 with some monitors. With SuperMon, type SYS 38893.

Coming next month:
"Starting Off in Assembly Language"
We'll take a look at assembly language mnemonics and go into more detail about programming at the machine level.

Remember, you can load and run SuperMon from the flip side of this disk.

Gazette, June 1994