Commodore 64: AUTOSWAP – RUN Two Programs Simultaneously!

The Transactor®

CDC 00594

January 1985

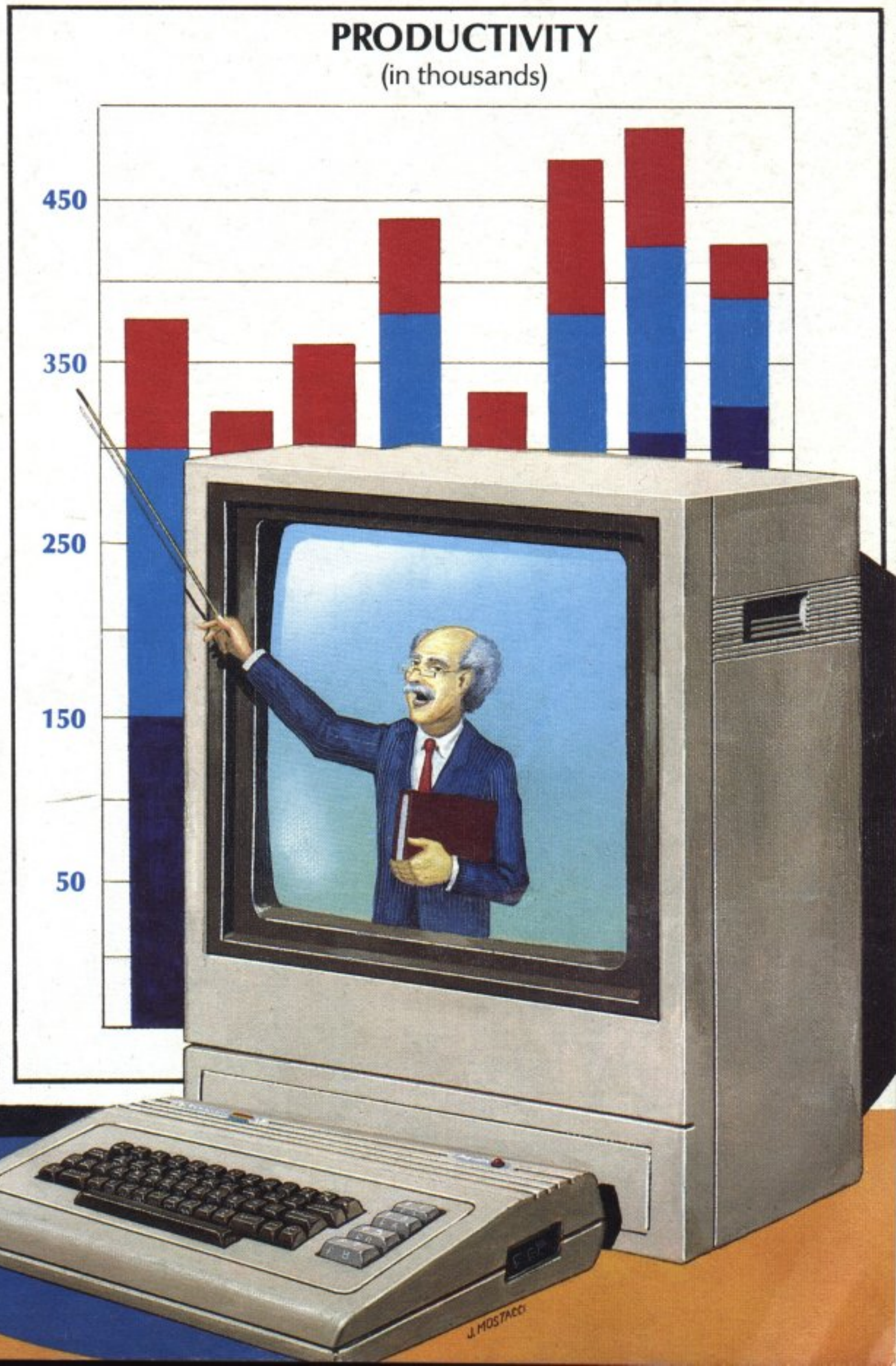
The Tech/News Journal For Commodore Computers Vol. 5

Business And Education

Issue 04
$2.95

- Machine Language Cascading Sort In Under 600 Bytes!
- Dynamic Expression Evaluation: Enter Math Functions As INPUT
- Structured Programming In Commodore BASIC
- Home Budget Program
- Compound Interest Calculator
- Phile Master: Your BASIC Database
- Office Automation For The Nineties
- Speller: A Drill Program Using Vectors
- Plus Lots More... And ALL On Commodore!

90% Advertising Free!



0 72246 00594 01

INTRODUCING

THE PRO-LINE TEAM

PAL 64
The fastest and easiest to use assembler for the Commodore 64.† Pal 64 enables the user to perform assembly language programming using the standard MOS mnemonics. **$69.95**

POWER 64
Is an absolutely indispensible aid to the programmer using Commodore 64 BASIC. Power 64 turbo-charges resident BASIC with dozens of new super useful commands like MERGE, UNDO, TEST and DISK as well as all the old standbys such as RENUM and SEARCH & REPLACE. Includes MorePower 64. **$69.95**

TOOL BOX 64
Is the ultimate programmer's utility package. Includes Pal 64 assembler and Power 64 BASIC soup-up kit all together in one fully integrated and economical package. **$129.95**

SPELLPRO 64
Is an easy to use spelling checker with a standard dictionary expandable to 25,000 words. SpellPro 64 quickly adapts itself to your personal vocabulary and business jargon allowing you to add and delete words to/from the dictionary, edit documents to correct unrecognized words and output lists of unrecognized words to printer or screen. SpellPro 64 was designed to work with the WordPro Series* and other wordprocessing programs using the WordPro file format. **$69.95**

NOW SHIPPING!!!

For Your Nearest Dealer Call (416) 273-6350

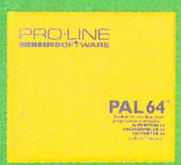
†Commodore 64 and Commodore are trademarks of Commodore Business Machines Inc.

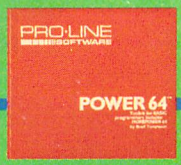
*Presently marketed by Professional Software Inc.

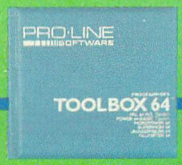
Specifications subject to change without notice...

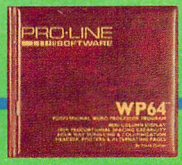
WP64
This brand new offering from the originators of the WordPro Series* brings professional wordprocessing to the Commodore 64 for the first time. Two years under development, WP64 features 100% proportional printing capability as well as 40/80 column display, automatic word wrap, two column printing, alternate paging for headers & footers, four way scrolling, extra text area and a brand new 'OOPS' buffer that magically brings back text deleted in error. All you ever dreamed of in a wordprocessor program, WP64 sets a new high standard for the software industry to meet. **$69.95**

MAILPRO 64
A new generation of data organizer and list manager, MailPro 64 is the easiest of all to learn and use. Handles up to 4,000 records on one disk, prints multiple labels across, does minor text editing ie: setting up invoices. Best of all, MailPro 64 resides entirely within memory so you don't have to constantly juggle disks like you must with other data base managers for the Commodore 64. **$69.95**









PRO·LINE SOFTWARE

(416) 273-6350
755 THE QUEENSWAY EAST, UNIT 8,
MISSISSAUGA, ONTARIO, CANADA, L4Y 4C5

Volume 5
Issue 04
Circulation 54,000

The Transactor

The Transactor
The Tech/News Journal For Commodore Computers

Managing Editor
Karl J. H. Hildon

Editor
Richard Evers

Technical Editor
Chris Zamara

Advertising Manager
Kelly M. George
416 826 1662

Art Director
John Mostacci

Subscriptions
Mandy Sedgwick

Contributing Writers
Don Bell
Michael Bertrand
Daniel Bingamon
Jim Butterfield
Gary Cobb
Elizabeth Deal
Domenic DeFrancesco
G. Denis
Brian Dobbs
Bob Drake
Mike Forani
Jeff Goebel
Dave Gzik
Phil Honsinger
Garry Kiziak
Scott Maclean
Glen Pearce
Louis F. Sander
George Shirinian
Darren J. Spruyt
Colin Thompson
Mike Todd
Vikash Verma
James Whitewood

Production
Attic Typesetting Ltd.

Printing
Printed in Canada by
MacLean Hunter Printing

Program Listings In The Transactor

All programs listed in The Transactor will appear as they would on your screen in Upper/Lower case mode. To clarify two potential character mix–ups, zeroes will appear as '0' and the letter "o" will of course be in lower case. Secondly, the lower case L ('l') has a flat top as opposed to the number 1 which has an angled top.

Many programs will contain reverse video characters that represent cursor movements, colours, or function keys. These will also be shown exactly as they would appear on your screen, but they're listed here for reference. Also remember: CTRL-q within quotes is identical to a Cursor Down, et al.

Occasionally programs will contain lines that show consecutive spaces. Often the number of spaces you insert will not be critical to correct operation of the program. When it is, the required number of spaces will be shown. For example:

print" flush right" – would be shown as – print" [space10]flush right"

Cursor Characters For PET / CBM / VIC / 64

Down	– q	Insert	– T
Up	– Q	Delete	– t
Right	– [	Clear Scrn	– S
Left	– [Lft]	Home	– s
RVS	– r	STOP	– c
RVS Off	– R		

Colour Characters For VIC / 64

Black	– P	Orange	– A
White	– e	Brown	– U
Red	– £	Lt. Red	– V
Cyan	– [Cyn]	Grey 1	– W
Purple	– [Pur]	Grey 2	– X
Green	– ↑	Lt. Green	– Y
Blue	– ←	Lt. Blue	– Z
Yellow	– [Yel]	Grey 3	– [Gr3]

Function Keys For VIC / 64

F1	– E	F5	– G
F2	– I	F6	– K
F3	– F	F7	– H
F4	– J	F8	– L

The Transactor is published bi-monthly by Transactor Publishing Inc., 500 Steeles Avenue, Milton, Ontario, L9T 3P7. Canadian Second Class mail registration number **6342.** USPS **725–050,** Second Class postage paid at Buffalo, NY, for U.S. subscribers. U.S. Postmasters: send address changes to The Transactor, 277 Linwood Avenue, Buffalo, NY, 14209, 716-884-0630.

Subscriptions:
Canada $15 Cdn. U.S.A. $15 US. All other $21 US.
Air Mail (Overseas only) $40 US. ($4.15 postage/issue)

Send all subscriptions to: The Transactor, Subscriptions Department, 500 Steeles Avenue, Milton, Ontario, Canada, L9T 3P7, 416 876 4741. From Toronto call 826 1662. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address (above) will be forwarded to Milton HQ.

Back Issues: $4.50 each. Order all back issues from Milton HQ.
SOLD OUT: The Best of The Transactor Volumes 1 & 2, and Volume 4, Issues 04 & 05
Still Available: Best of The Transactor Vol. 3, Vol. 4: 01, 02, 03, 06, Vol. 5: 01, 02, 03, 04

Quantity Orders:

CompuLit
PO Box 352
Port Coquitlam, BC
V5C 4K6
604 438 8854

U.S.A. Distributor:
Capital Distributing co.
Capital Distributing
Charlton Building
Derby, CT
06418
(203) 735 3381
(or your local wholesaler)

MICRON DISTRIBUTING
Micron Distributing
409 Queen Street West
Toronto, Ontario, M5V 2A5
(416) 593 9862
Dealer Inquiries ONLY:
1 800 268 9052
Subscription related inquiries are handled ONLY at Milton HQ

Master Media
261 Wyecroft Road
Oakville, Ontario
L6J 5B4
(416) 842 1555
(or your local wholesaler)

Editorial contributions are always welcome. Writers are encouraged to prepare material according to themes as shown in Editorial Schedule (see list near the end of this issue). Remuneration is $40 per printed page. Preferred media is 1541, 2031, 4040, 8050, or 8250 diskettes with WordPro, WordCraft, Superscript, or SEQ text files. Program listings over 20 lines should be provided on disk or tape. Manuscripts should be typewritten, double spaced, with special characters or formats clearly marked. Photos of authors or equipment, and illustrations will be included with articles depending on quality. Diskettes, tapes and/or photos will be returned on request.

From the (Technical) Editor's Desk

Chris Zamara is the latest addition to The Transactor crew. Although Chris joined us before the release of the last issue, this will be his first from start to finish. Consulting, programming, and special installations for several Metro schools and businesses were just a few of his pastimes before. So when Chris asked if he could write the editorial I thought, "he'll have a cleaner, fresher opinion of the subject at hand - plus a new outlook on the magazine business." And I was right! – M.Ed.

Changes

"There's nothing as constant as change". That's the way Karl normally closes his editorials, and it's an appropriate way to begin this one. Change comes to the magazine once again, in the form of another editor (me), and in the dropping of almost all advertisement. The increase in manpower means we can cover more ground in the form of technical information, editorial content, and relevant news. The dropping of ads suggests a more impartial viewpoint on new products, and implies greater integrity than that of magazines who fill 90% of their pages with advertising.

With this issue's theme in particular, it is fitting to talk of change. So much is changing so quickly with microcomputers in both business and education. So much, in fact, that it seems over–ambitious to tackle the topic in a single issue. Well, were going to try. We obviously can't comprehensively cover every aspect of Commodore in the classroom and the boardroom, but most articles and features are geared towards that theme.

While IBM seems to have the stronghold on the business micro market, there are many applications where a Commodore machine is more appropriate. An 8032 with an 8250 drive, for example, outperforms an IBM compatible machine hands–down in terms of disk access. With Commodore's new 8296, an even greater segment of the business market will be captured. If you own a small business, or even if you work for a large corporation, you can probably put your existing PET, 64, or whatever to work for you.

As usual, the manager column appears in this issue – the manager is an excellent tool for the kind of file management so often needed in business computing. There are also several articles containing business–oriented programs. The best thing about getting a program in a magazine as opposed to buying one is that you get a listing, and an explanation of how it works so that you can modify it to suit your individual needs – and maybe learn a few programming techniques in the process.

The education topic is one with many branches. Computers as teachers, educational information, and learning about computers themselves are a few. In various ways, we cover each of these branches. For example, the bits & pieces section is geared towards more educational pieces and less screen dazzlers. Structured programming techniques are addressed, to coincide with the way programming is taught in schools.

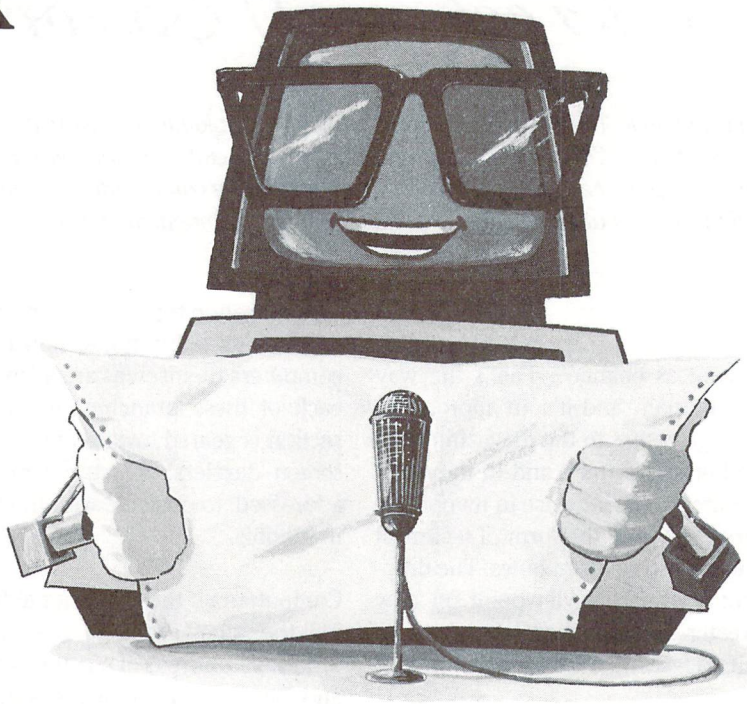
Computers on the educational front are probably undergoing the most dramatic changes right now. Schools are ordering hundreds of machines for computer labs, and more and more young people are being exposed to the beasts that many of us originally played with as a hobby. This is sure to have a dramatic impact on the future of the micro. In a way, the educational market goes hand–in–hand with the home market: students who become "hooked" on computing will have to get their fix by buying a system of their own. In many of these cases, that system will be a Commodore 64.

The coverage of these topics is done in the usual manner of this magazine. Articles reflect the personal opinions of the writer – be it one of the editors, or an outside contributor. There are sometimes differences of opinion – just read "Unveiling the Pirate, Part 1" and "Privacy vs. Protection – Who Loses?" in the last issue. While the views presented may seem totally polarized, there are often no simple answers regarding unfamiliar new situations brought about by the microcomputer revolution. It's up to you to decide where you stand on inflammatory issues – we just provide the fuel.

Finally, as the world changes, and The Transactor changes with it, we want to stay close to our readers. Send us letters. Tell us what you like. More editorial commentary? More highly technical information and techniques? More bits & pieces? By knowing what you want (and what you don't), we can change for the better. Because in the end, it's not the editors that decide the content of the magazine, and it's certainly not the advertisers. It's you.

Chris Zamara, Technical Editor

News BRK



Transactor News

Early Renewal Notices

When we first began publishing as an independent magazine, renewing subscriptions was left as the responsibility of the subscriber. Except many complained they were missing issues. But when we started mailing renewal notices we found that even with 2 months notice the complaints were the same. In an effort to change all this we may have over-compensated. Some subscribers have been getting renewal notices so early that it appears as though something may have gone wrong. We apologize for this but please be assured that you will not be short shipped. If you return your renewal forms early, an extra 6 magazines are merely tacked on to your allotment (we've been receiving some odd payments too, however extras are added appropriately). Our expiry update program will not cut you off until all magazines are sent. In fact, it will even send you one extra magazine before it actually deactivates you, and even then your record is not erased, just deactivated.

From now on, renewal notices will be sent just prior to your **second last** issue. This way you will get your notice and your second last issue at about the same time. At this point you have about 3 months to decide on renewing and still be guaranteed of not missing any. Your last issue will serve as a reminder. If you don't renew within a month after receiving this, we can't guarantee you won't miss an issue and it will cost you $4.50 to get it.

The reason? We get hundreds of new and renewal subscriptions every week. But the list must be printed up, sorted by postal/zip code, about 2 weeks before we actual print the magazines. So if your subscription order doesn't arrive 3 weeks before press time (worst case) chances are we'll have to print up the list before servicing your order. In essence, a magazine becomes a back issue as soon as the mail list is printed. Although we do have a short grace period, the post office does not like dealing with magazines that are submitted to them out of sorted order. This is known as "residue mail", and if there is too much, it must be posted at full rate.

"Full" and "rate" are two of the worst four-letter words in the subscription business. When we send out subscriber copies in bulk, the cost per piece is high, but acceptable. The cost for sending out singles is unnacceptable, on average about $1.40 per piece. Handling must also be considered. The time spent handling all subscribers at once stays economical, but handling singles (especially several singles) can be a time consuming task. Actually, according to calculations, $4.50 just barely covers it.

So please bear with us. Watch your mailing label. Your expiry issue number is printed on the first line of every label. If for some reason you don't get a renewal notice you can avoid missed issues by acting swiftly (remember, you have about a month after your last issue to make a smooth renewal).

When you do renew, please indicate RENEWAL by checking the box on the card. We do a search of everything that comes in, but there is always the possibility of entering you twice. If that happens you'll get two copies of each issue, instead of one copy for twice as long. And if that happens, please let us know immediately so it can be fixed. So far there have only been a couple, and those have been due to address changes.

By watching your label you can be sure exactly where you stand. When it is adjusted please check that the appropriate number of issues has been added, and give us at least 1 issue as a buffer to make the update in case timing is such that one issue goes out just before we process your mail.

Post Dated Cheques

One last note on subscriptions. . . please do not send us post dated cheques without plenty of enclosed warning. After the bank sends

them back once they won't accept them a second time. A note of any kind will help us intercept them so they can be stored until the bank is ready for them.

Business Reply Cards

Recently we've been getting some feedback on our business reply cards at the center of the magazine.

If you like making payments with a cheque, you can still take advantage of the free postage. Enclose your cheque in an envelope and tape it to the back of the business reply card. The Post Office says this is ok and we don't mind either. Just be sure the information on the card won't be damaged when the envelope is removed.

Some subscribers with charge cards have expressed concern over sending their card numbers through the mail in plain sight. Good point. But once again, don't hesitate to paste on a backing to cover it (or use an envelope) just so long as you don't put tape over vital information. If we rip the ink off, our only chance is to hold it up to a strong light and read it in the mirror.

And don't forget to include your address (we've actually received a few without) and especially your postal code.

Commodore News

The MCS 801 Colour Dot Matrix printer looks like an unconfirmed maybe for any new Commodore product catalogs. Inotherwords, don't wait for it.

Look for new 1541s with the drive door that has those "lever-handle" diskette ejectors. No word on write-compatibility with old 1541 diskettes, but we'll keep you posted.

New Commodore Peripherals

DPS 1101 Daisy Wheel Printer: A letter-quality printer ideal for crisp, professional correspondence. It is compatible with the Commodore PLUS/4 and features a bi-directional, logic-seeking print mechanism at 18 cps, 13' maximum width and letter quality reproduction.

MPS 802 Dot Matrix Printer: Bi-directional impact dot matrix printer with a 60 characters per second. It is completely compatible with all Commodore computers and prints numerics, symbols, and all PET graphics.

MPS 803 Dot Matrix Printer: A low-priced peripheral that adds versatility to its counterpart, the C-16 computer. It prints alphabetic, numeric, and all graphic characters with a variety of print styles and graphics capabilities.

1531 Cassette: A storage device for users of the C-16. It allows for quick load and access and uses standard audio cassette tapes—digital tapes are not necessary. (Editor's Note: Tape formats for the new plus4/16 cassette units have been changed making old tapes non-transportable to new machines, contrary to previous reports.)

CM 141 Colour Monitor: A high quality, high resolution peripheral redesigned to cosmetically coordinate with the Commodore PLUS/4. The CM 141 not only gives a clear, quality picture, but is also completely compatible with all of CBM's computer equipment.

Commodore Software

EASYCOMM 64

EASYCOMM 64 is Commodore's new disk-based terminal emulator for the C-64. Commodore has bought marketing rights from CompuServe for CompuServe's popular VIDTEX Terminal Program and will sell and distribute this program under the name EASYCOMM 64.

EASYCOMM 64 allows users to transfer messages and programs from electronic bulletin boards in CompuServe's large library to and from their own computer's memory and even directly to/from disk. It will also support 64 to 64 transfer via a large RAM buffer. EASYCOMM 64 has been described as one of the most powerful and easy-to use terminal programs ever developed for home computer Using CompuServe's 'B' protocol, it offers:

- 100% error detection and correction.
- A complete 30K RAM buffer which can capture data from a host system for immediate use or for disk storage for later use.
- Printer support, capable of capturing data at 120 char. per second.
- 10 programmable function keys to give ID or other frequently used commands.
- Color graphics and cursor positioning.

VIDEOTEX 64

Commodore's new VIDEOTEX 64 package for the C-64 combines two trends in the microcomputer industry – graphics and telecommunications. With VIDEOTEX 64, you can create business graphics and other pictures in high resolution color and combine them with text before transmitting them easily over regular phone lines, using either a VICMODEM or AUTOMODEM, to other VIDEOTEX 64 users.

Featuring single keystroke switching between interactive text and graphics functions, or between color and monochrome, you can create 'pages' of information, which can be saved, displayed and recalled from disk sent and received by modem, edited or printed.

VIDTEOTEX 64 features the latest in communications technology. Instead of the traditional ASCII protocol, it uses the new NAPLPS protocol, which has much greater power to create and transmit graphics. It is simple to use, with just a few menu screens, an users can get on-line help at an time without it interfering with what they are doing.

Real Estate brokers, interior designers, graphic artists and advertising executives are just a few of the numerous professionals who could benefit from the power of VIDEOTEX 64.

JUST IMAGINE an Animated Story

Commodore introduces 'JUST IMAGINE', the newest program in its educational software line. This innovative program is designed to help children combine visual and verbal skills to create an animated story on the C-64.

This program allows you to choose from nine exotic settings (a jungle, the moon, a barnyard are just a few). Country settings can be seen in summer or winter, others by day or night. You then pick from a selection of 50 static characters and objects to put into the setting (a gorilla, a cowboy, circus bear, taxi cab, a damsel in distress——as many as you like). After that you can choose three characters who can be made to move around the picture under control of a joy- stick as you develop a 'plot'. By using the built-in word processor, you write the story about the world that you have just created. Then with the touch of a key, you animate the scene and your story unfolds complete with sound effects. JUST IMAGINE encourages you to do just that --imagine a story in pictures, imagine it in words, and bring it all to life.

Other Commodore Software for the C-64 computer include, FISHMETIC, NUMBER BUILDER (elementary education), READING PROFESSOR, and WINDOW TO THE GALAXIES.

Marvel Comics' Superheros

Marvel Comics' Superhero favourites come alive. Commodore has entered the spine-tingling world of super hero adventure after signing with the Marvel Comics Group and Adventure International to produce and distribute six software programs featuring dynamic comic book favorites, including THE HULK and SUPERMAN. Marvel's new series, called QUESTPROBE is a unique comic for the C-64, showcasing a different Super hero in every issue. Unlike any other comic book series, each QUESTPROBE will have a corresponding computer software program, which continues the adventures of the Marvel Super Heroes.

The original concepts and creative direction behind Marvel's QUESTPROBE were developed by Scott Adams, founder of Adventure International and a pioneer of text adventure games on micros.

Marvel plans to distribute between 11 and 13 million of each comic book that will tie in with the Commodore computer games. The first game of the series, THE HULK, is scheduled for late August release. The series will run on the C-64 and the Commodore PLUS/4 computers.

Events

World Of Commodore II Show

The second annual World Of Commodore Show is rapidly approaching. Last year's show recorded the largest draw for any show in the history of Toronto's International Centre with over 38,000 total attendance. This year should be no different and stands a good chance of topping its own record.

Show dates are November 29 & 30, December 1 & 2 (Thurs thru Sunday). We'll be there too. Transactor Publishing will be at Booth No. 712. So come on out. . . we'll be happy to meet you!

Educational Software: Focus on the Future

The First Annual Regional Educational Software Infomart will be held in New York City at the Penta Hotel (the former New York Statler), November 2-4, 1984. This newsmaking event is long overdue in a region which has the largest concentration of the nations most affluent schools in terms of per pupil spending as well as the largest concentration of owners of personal computers.

The focus of the entire event will be on parents and teachers in the New York tri-state area. The attendees will be drawn from the more than 11,000 schools in the metropolitan area as well as the current home owners of personal computers which number 60,000 in New York City alone. Many of the latter population purchased their computers for business use and are now ready to address the many educational applications available but do not know how to properly evaluate software or even how to buy it. We will provide that service by creating an atmosphere totally conductive to learning. Friday, November 2, will be set aside strictly for educators, and Saturday and Sunday, November 3 and 4, will be oriented to parents.

The seminar programs on all three days will address issues in educational software development of current and future concern to the specific audiences of each of the days. Some of the topics to be covered are:

- The computer as a tool for problem solving or drill and practice.
- When are children ready for computer education.

The impact of computer-aided-education on self-learning. How will teaching be streamlined according to IQ and ability - will students learning at their own speed with desk top computer.

- How is and will software be evaluated?
- The home entertainment center as a learning center.
- The computer as a flexible tool for the entire family.
- Hands-on, how-to workshops for Music, Art, Math, Reading, Science . . .

The event will emphasize a hands-on demonstration of software in an environment conducive to learning. "Home Rooms" will be set up for parents to learn in a parent-friendly space.

Special events will focus on: Computer Graphics and Electronic Art curated by Art Expo, New York; The Role of Robotics in Education Future; Special Effects Production Utilizing Computers. For further information please contact:

Nina T. Kurtis
National Educational Software Informarketing Corp.
225 East 57 Street, 17H
New York, NY
10022 212-688-8904

Books

Commodore 64 PRG Price Reduction

If you've been waiting for the price of the Commodore 64 Programmers Reference Guide to come down, good news! 2295 Canadian pennies will now get you one! The 64 PRG is Copp Clarks' second largest selling book in Canada.

Tricks & Tips for The Commodore 64

Tricks & Tips for the Commodore 64 is a collection of easy-to-use programming techniques for the world's most widely owned com-

puter. This 250+ page book is the perfect companion volume for those of who have run up against those nasty programming problems. Tricks & Tips for the Commodore 64 makes programming simpler and more exacting. A partial list of the contents include:

- Advanced graphics – 3D graphics, defining and modifying the character set
- Easy Data input – cursor positioning, turning the cursor on and off, repeat function for all keys, making a mouse out of a joystick.
- Advanced BASIC – Copying BASIC to RAM; GOTO using calculated line numbers; defining a new INSTR and STRING$ function tricks.
- Other Languages – FORTH, Pascal and LOGO
- CP/M on the Commodore 64
- Data Processing – Sequential, relative and direct access files
- POKES and other Routines – Pokes and zero page, sorting strings, DUMPing variables

Don't let your Commodore 64 sit unused in a corner because it's too hard to program. If you want to get more from your programming sessions with your Commodore 64, then try a few "TRICKS & TIPS".

Available now, 280 pages, $19.95 US., ISBN# 0-916439-03-8, from your local dealer or directly from Abacus Software.

Abacus Software
PO Box 7211
Grand Rapids, Michigan
49510 616-241-5510

The Graphics Book For The Commodore 64

The Graphics Book For The Commodore 64 takes you from the fundamentals of graphics to advanced topics such as computer aided design. This book is for all of you who want to use the fantastic graphics capabilities of the '64. Author Axel Plenge delves into subjects that include:

- creating new character sets
- sprite design and movement
- high resolution and multicolor graphics
- programming for the light pen
- controlling the VIC Chip
- shifting the screen memory
- IRQ handling
- 3D graphics, projection and curves
- animation and moving pictures

The Graphics Book For The Commodore 64 is filled with many program listings that make learning by example both easy and straight forward. Here is a book that enables anyone to make his mark in the fascinating world of computer graphics. An optional diskette is available so that the reader does not have to key in the programs from the listing.

An optional diskette containing the program listings from the book available for $14.95 US. Available now, 250+ pages, $19.95 US., ISBN# 0-916439-05-4, from your local dealer or directly from Abacus.

The Commodore 64 For Scientist & Engineers

The Commodore 64 For Scientists & Engineers is an introduction to the world of computers for scientific applications. Author Ranier Severin has tailored a book specifically for the sciences and the '64.

He discusses the different variable types; computational accuracy; POKEs that are useful in solving scientific problems; various sort algorithms such as bubble, quick and shell sorts.

Examples have a mathematical orientation and include differential equations, linear and nonlinear regression, CHI-square distribution, Fourier analysis and synthesis, scalar and vector products, matrix calculations and much more. Programs cover the fields of chemistry, physics, biology, astronomy, electronics and much more.

All software listing illustrate the enormous range of capabilities which the Commodore 64 has in the sciences and engineering.

Available November 1984, 250+ pages, $19.95 US., ISBN# 0-916439-09-7, from your local dealer or directly from Abacus.

The Machine Language Book For The Commodore 64

The Machine Language Book For The Commodore 64 is aimed at the Commodore 64 owner who wants to progress beyond BASIC. If the reader wants to write programs that run faster, use less memory or perform functions that are not available in BASIC, then this book will help him understand machine language.

This is a 200+ page detailed guide to the complete instruction set of the 6510 processor of the Commodore 64. The book is filled with examples of machine language routines so that the reader can learn from working programs. These examples are geared specifically to architecture of the Commodore 64. You'll learn to add your own keywords to BASIC, access peripheral devices, program high resolution graphics and more.

Included in these pages are listings of three full length programs. One is a working assembler so the reader can create his own machine language programs. The second is a working disassembler so the reader can inspect other machine language programs. The third is a 6510 simulator so that the reader can "see" the operation of the processor.

An optional diskette is available so that the reader does not have to key in the programs from the listing.

Optional diskette containing the assembler, disassembler and 6510 simulator programs available for $14.95 US. Available now, 215 pages, $14.95 US., ISBN# 0-916439-02-X, from your local dealer or directly from Abacus.

Advanced Machine Language Book For The 64

Advanced Machine Language Book For The Commodore 64 is Lothar Englisch's companion to his best selling introductory book about machine language programming.

He discusses many in depth topics about the machine language programming on the Commodore 64. You'll learn how to handle interrupts from the CIA; program the video controller, timer and

real time clock; perform serial and parallel input and output at machine language speed; extend BASIC with new commands and functions. Englisch packs this book with dozens of tips and tricks for the machine language programmer.

If you work with machine language, then you'll want to own this valuable book. Available now, 200 pages, $14.95 US., ISBN# 0-916439-06-2, from your local dealer or directly from Abacus.

Software News

OS-9: TPUG Makes A Standard Operating System Available For The SuperPET

TPUG is currently planning to implement the popular 6809 operating system "OS-9" on the SuperPET. OS-9 greatly expands software availability and the hardware capabilities of this computer while at the same time preserving access to the Waterloo languages and programs. The methods of implementation, are for the most part resolved. A prototype will be available in September.

The cost of OS-9 to club members will be around $150 US., which will include the cost of a hardware modification that will not affect the normal operation of the SuperPET. Because every copy requires the purchase of a license from Microware Inc., a limited number of copies will be available through TPUG which is sponsoring the project on a cost recovery basis. To reserve your copy please mail $68.09 to TPUG. (1912A Avenue Rd., Suite 1, Toronto Ont., M5M 4A1, Canada)

Features of OS-9 include:

1. A true operating system with the features of UNIX and the simplicity and command style of Commodore BASIC;
2. A multi-tasking and multi-user environment;
3. The ability to redirect and 'fork' input and output to printers or to other devices;
4. A flexible command interpreter which allows users to define and create custom commands;
5. File management is structured to permit multi-level directories similar to what is now available in MS DOS;
6. Directory entries (files) automatically receive a time and date stamp;
7. File access privileges may be restricted by the owner of a file.

Extensive software is available for OS-9 all of which will run on SuperPET OS-9.

System Software Provided with OS-9: assembler, editor, command (shell) library monitor, symbolic debugger

Available Languages (compilers) include: BASIC, Pascal, CIS-Cobol, 'C', FORTH, 6809 Assemblers . . . and others.

Available Application Programs: Word processors and spelling checkers, inventory and accounting applications.

Public Domain: Terminal emulation, utilities etc.

TPUG will participate in the acquisition of public domain software and assist users in the conversion of commercial software so that it will operate on Commodore drives.

Portability and Expandability

1. SuperPET OS-9 programs will run on all OS-9/based microcomputers.
2. Programs developed under OS-9 for other computers (such as the Radio Shack Color Computer) will run on the SuperPET.
3. OS-9 will give users direct access to hardware drivers that could operate devices such as parallel printers, additional serial ports, hard drives etc.
4. There will be source code compatibility to versions of OS-9 that are planned for the Motorola 68000.

Those of us in TPUG who are involved with the installation of OS-9 are excited about the prospects of new applications with this operating system. We are certain that it will prolong the utility of the SuperPET but we do urgently need your support.

Gerry Gold (416) 667 3159 / 225 8760
Avy Moise (416) 667 3954 / 667 9898

Software Publisher, Bookstore Chain Plan Innovative Joint Promotion

Waldenbooks and Scarborough Systems have announced an innovative joint book-software promotion in the bookstore chain's 860 nationwide outlets. Under the promotion, consumers who purchase Scarborough's MASTERTYPE, the best-selling educational software program on the market today, and any book at Waldenbooks, will be eligible to receive a $10 rebate from Scarborough. The refund is obtainable by mailing in proofs of purchase of MASTERTYPE and of the book.

The promotion will run this summer, said Sanford K. Bain, Vice President, Marketing, who announced the cooperative agreement. "In developing the promotion, we wanted to encourage the consumer to be comfortable buying software in a bookstore environment," commented Bain. "We're very proud of our new association with Waldenbooks, a chain which as aggressively devoted itself to serving computer enthusiasts by building a broad line of computer books and software for the home."

MASTERTYPE, an educational program with an arcade game format, makes learning typing and keyboard skills both easy and fun. The program has sold over 200,000 copies, an industry record in the home educational category. Used in home, schools and businesses as the first step to computer literacy, MASTERTYPE will be available at Waldenbooks for the entire Apple family, Atari and Commodore-64 at $39.95 US.

Scarborough $4 Million Giveaway Will Let Parents Donate Quality Software To Schools

Responding to an urgent need for quality software in the nation's schools, Scarborough Systems, publisher of the best-selling educational software program of all time, MASTERTYPE, has announced it will sponsor a program designed to provide donated software valued at up to $4 million to public and private schools this fall.

Through an innovative merchandising technique, consumers who purchase one of Scarborough's seven educational software programs between September 15 and December 15 will become donors of another Scarborough program of their choice to any teacher and school they select.

"Although a majority of schools across the country have at least one microcomputer, all our available information tells us schools are woefully ill-supplied with quality software that teaches youngsters to use the computer for creative learning, not just drill and practice," said Scarborough President Francis P. Pandolfi.

"Our campaign, which we've called 'Be a Hero and Software a School,' will encourage parents and other education-concious adults to help schools by supplying them with innovative creative software that has been tested in the home market.

Purchasers of any Scarborough educational product will receive a "donation certificate" entitling them to give a Scarborough product of their choice to a teacher and school of their choosing. The certificates will appear in leading educational and consumer publications, be available from computer and software dealers, and also be attached on products and distributed through the educational system.

To make the donation, consumers will return the certificate with a product warranty card and a handling and mailing fee of $3.50 to Scarborough. The publisher, in turn, will send the designated software program to the school with a gift card indicating the donor's name.

The software giveaway program is being supported by Scarborough with an aggressive $1 million trade and consumer advertising and promotional campaign. A special direct amil and advertising program to schools is also in the plans.

Peter DuPont, Scarborough's Vice President of Sales, indicated that national chains such as Sears, K-Mart, Target Stores, along with leading software retail chains, have pledged their support of the innovative software giveaway.

Additionally, Scarborough has encouraged dealers to visit and "adopt" schools in their local community and maintain a file to assist consumers who want to donate software but do not have access to a teacher's name or the computer brand in the local school.

Scarborough Systems, a Tarrytown, NY-based publisher which shipped its first product in October, 1983, is recognized today as one of the fastest-growing software publishers in the competitive educational productivity segments of the home market.

In addition to MASTERTYPE, Scarborough programs include PHI BETA FILER, a list management program for children; RUN FOR THE MONEY, a business game with a space-age theme; SONGWRITER, PICTUREWRITER, PATTERNMAKER and LASER SHAPES. Retailing for $39.95 to $49.95 US., all are or soon will be compatible with the Apple II family of computers, Commodore-64, IBM PC and PCjr and Atari.

Scarborough Systems Offers Combined Format for ATARI, C-64

RUN FOR THE MONEY, Tom Snyder's popular business strategy game, and YOUR PERSONAL NET WORTH, a powerful new personal finance product to manage, track and organize family money matters, are the first software titles developed by Scarborough Systems with Atari and Commodore-64 formats in one box, without an increase in price. The new packaging was exhibited at the CES in June, and will be available to retailers shortly.

The desire to help merchants turn over inventory more frequently prompted Scarborough to develop the new combined format, said Peter Dupont, Vice President, Sales at the Tarrytown, NY - bases software publisher.

"We've had a tremendous response from software dealers everywhere," Dupon said. "The need to use limited shelf space more efficiently is a very real problem in retailing, plaguing both smaller dealerships and the mass merchandiser." In addition, he said, the combined format will help Scarborough to better manage its growing product line.

Scarborough also announced that all future titles the firm develops for both Atari and the Commodore-64 computers will be packaged in this manner.

Scarborough continues to be an innovative publisher of home software, with its packaging that features the use of four-color photographs, dual purpose plastic boxes and color-coded series and age bars.

Scarborough Systems, Inc.
Sanford Bain
25 N. Broadway
Tarrytown, New York
10591 914-332-4545

Solid Modeling Study From The S. Klein Newsletter On Computer Graphics Reports That System Installations To Double By Year-End 1985.

Solid modeling is a computer graphics technology used to describe physical objects completely, inside and outside. And because of that unique capability, sold modeling "is destined to become a ubiquitous tool," especially in CAD/CAM, engineering, architecture and animation.

So finds a 65-page study just issued by The S. Klein Newsletter On Computer Graphics of Sudbury, Massachusetts, published by Technology & Business Communications Inc.; also of Sudbury. Entitled "Solid Modeling In Computer Graphics: Technology; Applications; Supply Sources", the report documents the emergence of solid modeling technology as a powerful analytic and descriptive tool.

It points out that the number of commercially-available programs have increased from seven only two years ago to twenty five currently. Similarly, the report forecasts that solid modeling systems, totaling forty user sites in 1982, will jump to 350 installations by the end of this year and to 600 system sites by year-end 1985.

There's good reason for the projected growth. Unlike the other modeling techniques — wireframe and surface — only solid models contain complete data on the interior of objects. This facilitates mass property computations, interference checking, finite element analysis, numerical control tape preparation, and a host of other calculations and manipulations. It is why the S. Klein Newsletter staff that prepared the report anticipate that solid modeling will eventually become the prevalent way of doing computerized geometric modeling.

The study describes solid modeling technology in-depth, shows where it is applicable, and discusses its future. Also covered is the "surprising role" of the personal computer in solid modeling. The study also includes a directory of solid modeling supply sources and comparisons of their products. It contains many illustrations, a glossary, and a bibliography for further reading.

Report price: $129 prepaid; (outside US., $143). To order, or for further information, Write - Solid Modeling Study, The S. Klein Newsletter On Computer Graphics.

Technology & Business Communications, Inc.
730 Boston Post Road, PO Box 89
Sudbury, Massachusetts
01776 617-443-4671

CADPAK-64 - Superb Design Tool for the Commodore 64 and 1541 Disk Drive

CADPAK-64 is a superb tool for computer aided designs and drawings. You draw directly on the high resolution screen using a lightpen.

CADPAK-64 lets you create and edit graphic pictures, drawings, layouts and renderings - quickly, accurately and artistically. The output is suitable for reproduction as hardcopy printout or photographs.

CADPAK-64 is very easy to use. The main menu lets you choose from a complete list simply by selecting an option with the lightpen. Graphics are drawn on the screen at the exact location that you point to using the lightpen. Your interaction with the keyboard is minimal.

CADPAK-64 provides two high resolution graphics screens. You can draw any combination of LINEs, BOXes, CIRCLEs, ELLIPSEs; FILL using solid colors or patterns; freehand DRAW; COPY sections of the screen to other areas; ZOOM in and do detailed design within a small section of the screen. You can choose point placement down to the pixel level by using our AccuPoint cursor positioning.

CADPAK-64 also has a powerful OBJECT EDITOR that lets you define the shape of OBJECTs such as furniture, electronic circuitry or machinery. your definitions can be as intricate as the screen resolution permits. You can name these OBJECTs, build a library of them on disk, and recall/display them on the screen at varying SCALEings or ROTATIONs.

When your designs are complete, you can SAVE/RECALL your finished pictures to/from the disk. Finally you can reproduce the results to one of the popular dot matrix printers: Commodore 1525E/MPS-801; Epson MX, RX or FX series; Okidata; or C.Itoh Prowriter.

CADPAK-64 uses a Commodore 64 with 1541 disk drive and requires a high quality lightpen. Includes detailed user's manual and tutorial in 3-ring binder. Price is $49.95 US.

We recommend Madison Computer's McPen available thru us for $49.95 US. Other high quality lightpens are suitable. Available from your local dealer or directly from Abacus Software.

Abacus Software
PO Box 7211
Grand Rapids, Michigan
49510 616-241-5510

Ideas To Use On Your Commodore 64

Ideas To Use On Your Commodore 64 presents dozens of helpful and fun things to do with your '64. It's written for the novice and reads like a novel, but contains program listings that prove the '64 to be *the* home computer. Here's some of the themes that are covered:

- recipe card filer
- automobile expense minder
- electronic calculator
- store window advertising
- strategy games
- computer poetry
- construction cost estimator
- publicity letter generator
- party invitations

All programs in IDEAS TO USE ON YOUR COMMODORE 64 are ready to key in and use. If you don't know what to do with your '64, you will after reading this book.

Optional diskette containing the program listings from the book available for $14.95 US. Available in November 1984, 200 pages, $12.95 US., ISBN# 0-916439-07-0, from your local dealer or directly from Abacus Software.

XREF-64 - BASIC Cross Reference for the Commodore 64 and 1541 Disk Drive

XREF-64 is an indispensible programmer's aid for authors of BASIC programs. XREF-64 indexes the usage of all variables, line numbers, numeric constants and BASIC keywords. You know immediately which line numbers refer to a given variable name or which line numbers use certain BASIC commands. XREF-64 is almost a necessity for debugging your lengthy BASIC programs.

The cross reference is listed in sorted order onto either your screen or printer. XREF-64 reads your program from diskette and prints to any Commodore or Ascii printer. It's fast since sorting is performed at machine language speed.

XREF-64 even lets you cross reference non-Commodore keywords. So if you have ULTRABASIC-64 from ABACUS Software or VICTREE from Skyles Electric Works, for example, you can customize XREF-64 for these products and display all references to these extended BASIC commands.

XREF-64 requires a 1541 disk drive. Printer is optional. Available on diskette for $17.95 US. from your local dealer or directly from Abacus Software.

ASSEMBLER/MONITOR 64

Our ASSEMBLER/MONITOR 64 package is for the development of machine language programs on your Commodore 64. This low-cost package has high-priced features. The assembler capabilities:

- fast macro assembler
- conditional assembly capabilities
- full screen editing of source program
- object code assembles to memory, disk or tape
- complete symbol table listing
- source file chaining capabilities

The monitor capabilities has 15 functions including:

- hunt (for characters)
- disassemble code
- transfer blocks of data
- compare blocks of data
- access to other memory banks
- single step execution
- quick trace with breakpoints
- can coexist with the assembler

Both the ASSEMBLER and MONITOR are written in machine code for speed and efficiency. ASSEMBLER/MONITOR–64 is available on diskette and includes complete user's guide in 3–ring binder for $39.95 US., from your local dealer or directly from Abacus Software.

PASCAL–64 – Full Compiler for the Commodore 64 and 1541 Disk Drive

PASCAL–64 is a full Pascal compiler and language development package. in addition to almost all of the elements of the Jensen & Wirth language, you get special features such as high resolution graphics, sprites and file management.

PASCAL–64 is so advanced that you can even handle interrupts in the Pascal language. And for extra special needs, by interfacing to our new ASSEMBLER/MONITOR package you have direct access to your Pascal variables.

PASCAL–64 is fast, since it compiles to 6502 machine language. Your compiled programs can be SAVEd, LOADed and RUN like BASIC programs, but will run much faster.

Here's the run down on the language.

- standard programming structures: FOR TO/DOWNTO, IF THEN ELSE, REPEAT UNTIL, WHILE DO, CASE OF, GOTO, EXIT, WITH DO
- data types: REAL, INTEGER, CHAR, BOOLEAN, SET, RECORD, ARRAY, PACKED ARRAY, pointer, FILE
- functions: SIN, COS, TAN, ARCTAN, EXP, LN, SOR, SORT, ABS, TRUNC, NOT, PEEK, ORD, CHR, RND, SUCC, PRED, LENGTH, VAL
- input/output: READ, READLN, WRITE, WRITELN, GET, REST, CLOSE, SEEK
- extensions: GRAPHIC, SCREENCLEAR, PLOT, UNPLOT, SPRITE, POKE, SYS, INTERRUPT, FILLCHAR, NEW
- predefined names: TRUE, FALSE, NIL, ERROR

You can use PASCAL–64 for program development since it has full file handling capabilities for sequential and relative data management; multi–dimensional arrays, dynamic storage with the procedure NEW and pointer variables, and easy string handling procedures and functions.

PASCAL–64 is available on diskette with complete manual in 3–ring binder for $39.95 US., from your local dealer or directly from Abacus Software.

DATAMAT–64 For Commodore 64 and 1541 Disk Drive

DATAMAT–64 lets you design your data base in free form using the full screen editor. You can define up to 50 fields per record. DATAMAT–64 can store up to 2000 records per diskette. When you're done defining your database, you simply press a function key to save your data base template.

Reporting capabilities are just as flexible and complete. You can sort on multiple fields in any desired combination. Then you can select records for printing in your required format.

DATA–64's menu screens take you from function to function. You can go from data entry, to data correction, to sorting and reporting just by following the menus.

Our data base management software doesn't cost more and can perform better than the more expensive ones. Available on diskette with complete manual in 3–ring binder for $39.95 US., from your local dealer or directly from Abacus Software.

TAS–64 – Stock Market Technical Analysis System for the Commodore 64 and 1541 Disk Drive

TAS–64 is for the serious stock market investor who requires the intricate charting capabilities that only a computer can provide. Many sophisticated investors use technical indicators to determine when to buy and sell securities. TAS–64 can analyze and chart these indicators to help the investor make his decisions.

Using TAS–64 you can download your indicators from the Dow Jones New/Retrieval Service. Alternatively, you can manually enter, edit, review, save and recall these indicators. You can track high, low, close, volume, bid and ask by date. You can place 300 periods of information for up to 10 different issues on a data diskette. You can format as many data diskettes as your portfolio requires.

TAS–64 walks you through the chart building with easy understand menus. You can build a variety of chart types on the split screen: 7 moving averages, 3 oscillators, 1–99% trading bands, least squares, 5 volume indicators. You can also build comparison and relative charts for two different issues. Finally you can record your charts on the printer for more detailed analysis afterwards.

TAS–64 requires a standard Commodore 64 with 1541 disk drive. For printing your charts, TAS–64 works with the Commodore 1525E; Epson MX, FX and RX series; Gemini series; Okidata; and C.Itoh Prowriter printers.

The TAS–64 includes the master diskette, a sample data diskette and an easy to follow 150+ page manual in 3–ring binder. Price is $84.95 US., available from your local dealer or directly from Abacus Software.

Arrays, Inc./Continental Software Announces "GET RICH: STRATEGIES, VOL. I"

Arrays, Inc./Continental Software announces "Get Rich: Strate-

gies," the first volume in an exciting new series of personal financial planning programs. Designed to teach basic money management skills, "Get Rich: Strategies" also offers a range of financial solutions tailored to individual needs.

According to Arrays, Inc. Executive Vice President Hank Scheinberg, "'Get Rich: Strategies' is a potent set of financial 'tools' that can direct the end-user in accumulating greater wealth or in planning any number of money-related 'what if' situations in-between. The entire series will be as easy to run as our 'Tax Advantage' program," Scheinberg concludes, "so that both novice and professional computer users can benefit from the wealth of information contained in each volume."

Get Rich: Strategies incorporates three major financial planning tools, WORKSHEETS enable users to set goals, as well as determine net worth and discretionary income; CALCULATIONS are provided for solving a variety of problems involving money, time and interest; and GRAPHS analyze the performance of investments, interest rates and other related matters over a long period of time. Results may be printed out in a convenient form.

The following worksheets also are included in Get Rich: Strategies — Saving Goals, Assets, Liabilities, Income, Expenses and Calculations. Following the same format as Continental's successful "The Tax Advantage," the program "asks" for information in a very straight-forward, non-intimidating fashion, with the end-user merely typing in answers.

A "function menu" at the bottom of the screen always reminds users of the basic methods available to design a financial plan. The program is set up so that users can move easily and quickly through different forms, and also offers a helpful tutorial.

Three more volumes in the Get Rich! series of personal financial programs. The new modules are titled "Get Rich: Real Estate Planning," "Get Rich: Insurance Planning" and "Get Rich: Retirement and Estate Planning."

Volumes II through IV are designed to work in tandem with Get Rich: Strategies, the first program in the series. While Get Rich: Strategies gives users a profile of their entire financial picture, the supplemental volumes are more subject-specific.

Get Rich: Real Estate Planning covers all aspects of investing — from types of properties to methods of buying. It helps determine, for example, if it's better to buy or rent a property and what type of mortgage might be appropriate.

Get Rich: Insurance Planning concentrates on how much and what types of life insurance to buy, answering a multitude of 'what if' questions on this crucial topic in an non-pressurized way.

Get Rich: Retirement And Estate Planning helps the end-user plan for later years by acting as an educational tool about such investments as IRA and Keogh and then mapping out a potential worry-free retirement based on individual needs.

Volumes I to IV in the Get Rich! series for the Commodore 64 are priced at $49.95 US. Volume I and II will be available in September of this year, while Volumes III and IV will be available in October.

PHONE CALL – An Innovative, Affordable Telecommunications Program for The 64

Arrays, Inc./Continental Software introduces "PHONE CALL" an innovative, affordable telecommunications program for the Commodore 64 computer. Quick, reliable and easy to use, PHONE CALL converts the C-64 into a "smart" terminal capable of performing a variety of transactional operations, such as home banking, electronic mail, data retrieval, travel planning and many other uses.

Available for only $49.95 US., suggested retail, PHONE CALL offers a range of features usually found in programs priced significantly higher. Extensive help screens and practical, result-oriented documentation enable fast access to PHONE CALL's functions. It also will grow with the user as his or her telecommunications needs become more advanced. PHONE CALL makes it possible to "speak" to on-line databases, to digitized appliances, and to other computers — whether micros or mainframes. It is the only software in its price range that permits uploading and downloading of machine language programs, including the most sophisticated programs written for the Commodore 64.

Scheduled availability for PHONE CALL is late summer '84.

Linda Feldman
Arrays, Inc./Continental Software
11223 South Hindry Avenue
Los Angeles, California
90045 213-410-3977

ENTECH Extends Music Contest Deadline

EnTech Software has extended the deadline of the First Annual Computer Song Writing Contest to December 1, 1984, because according to EnTech Chairman Ray Soular, "a lot of people who have just recently become aware of the contest have requested more time to enter."

EnTech is offering a grand prize of $1,000 plus free studio recording time to the best song written on the Commodore 64 with its Studio 64 program. Entries submitted on disk will be judged by a panel of ten music industry professionals, which at this time includes Vince Flemming of Strangeland Music (ASCAP) and Dan Seitz of Aleph-Baze (BMI).

Soular said that to fill the numerous requests for contest information, EnTech is printing and sending additional contest entry blanks to thousands of retailers across the country. Retailers needing entry blanks can contact:

ENTECH Computer Song Writing Contest
PO Box 185
Sun Valley, CA
91353 818-768-6646.

SOFTSYNC Wins ARKIE Nomination For DANCING FEATS

SOFTSYNC announced that DANCING FEATS, its critically acclaimed music/entertainment program, has been nominated for an ARKIE award, Electronic Games Magazine's annual Arcade Award which salutes excellence in the electronic gaming field.

"We are extremely pleased to have DANCING FEATS receive such an honor," stated Kenneth P. Currier, Vice President of programming at the New York based computer software company.

"The program has been enthusiastically reviewed in almost every major computer magazine, and it is wonderful to cap off its debut with a commendation such as this," Currier added.

DANCING FEATS turns the computer into a musical instrument but requires no previous knowledge of music.

"You select a bass, beat, style, tempo, and ending which the computer transforms into a back-up band," explained Currier. "Then you use the joystick to compose an improvised melody which works within the greater musical structure."

DANCING FEATS's hook seems to be that it offers immediate gratification. "The program lets you play music immediately – and it splashes brightly colored bar chords across your TV screen to boot!" An additional feature of the program allows users to digitally record their compositions onto disk or tape and play them back later.

DANCING FEATS for the Commodore 64 has a suggested retail price of $29.95 US. for the disk version and $24.95 US. for cassette versions. Additional information is available from SOFTSYNC.

Linda Schupack
SOFTSYNC, Inc.
14 East 34th Street
New York, NY
10016 212-685-2080

Design Your Own Computer Games Without Computer Knowledge with 'ADVENTUREWRITER'

Codewriter Corporation, has introduced ADVENTUREWRITER, a games system based on their CodeWriter concept, that allows the user to design games by programming in plain English.

"ADVENTUREWRITER provides the software owner with some significant advantages over purchasers of packaged games that are "locked" in a single format", says Warren Shore, president of Codewriter Corporation which is based in Niles, Illinois.

"One distinct advantage of ADVENTUREWRITER and all of our CodeWriter-based products is versatility," says Shore. "Your are not buying a one-purpose game application. You are getting a games system that allows you to create a virtually limitless number of games."

ADVENTUREWRITER also provides another level of entertainment — the challenge and sense of accomplishment of designing your own unique games.

ADVENTUREWRITER was announced in January at the Consumer Electronics Show in Las Vegas and Shore reported "an extremely enthusiastic response — commitments for 20,000 units."

He added that many distributors and merchandisers said ADVENTUREWRITER would open up a whole new market for game enthusiasts who have ideas for games but no programming capabilities.

ADVENTUREWRITER games are easily designed. The software instructs the user how to proceed through a systematic process of building a format, requiring the operator only to type in appropriate responses in simple English.

Throughout the programming process, ADVENTUREWRITER provides "open windows" that allow the operator to build new concepts into the game. Treasure hunting, jungle escapes, mazes, hazards, villains and heroes can be designed by the user.

Once the game is set, ADVENTUREWRITER automatically converts the English responses into computer language and the game can be recorded on the user's own diskette.

Since each game created is the user's own design, the "author" can claim ownership and even copyright the game.

"Introduction of ADVENTUREWRITER's unique concept at a time when sales of computer adventure games are soaring positions Codewriter Corporation at the leading edge of a whole new market," says Shore.

Look for ADVENTUREWRITER at your local computer store, or for dealer information contact: Micro Marketing Canada, 169 Inglewood Drive, Toronto.

Budman Math Inc.,
Public Relations and Publicity Consultants,
505 Eglinton Avenue West,
Suite 303,
Toronto, Ontario. M5N 1B1

A CHRISTMAS ADVENTURE
A Very Special Program From BITCARDS INC.
Can be Custom-Programmed For Holiday Gift-Giving

BitCards Inc. proudly announces the release of a unique adventure program written especially for the upcoming holiday season. Set in and around Santa Claus' ice-castle at the North Pole, the player will discover at the outset of the adventure that Santa has mysteriously disappeared. With Christmas only hours away, his annual gift-delivery run is in grave jeopardy! The player's mission is clear: Explore the many rooms of the castle and its outbuildings, unravel the mystery of Santa's disappearance, find and free him, and thus 'save' Christmas.

Care has been taken in the scripting and design of the program to make it appealing to a broad range of micro-users. A Christmas Adventure is more than a challenging 'puzzle'; it is, above all, a holiday entertainment filled with fun and surprises: The unexpected appearance of an errant Pacman, for example, munching his way across the screen; and the computer's response, "Get out of here dummy, you're in the wrong game!"

An extensive 'intelligent' HELP utility ensures that players of all skill levels will be able to proceed through the adventure, so that they don't miss any of the fun.

A Christmas Adventure CAN ALSO BE CUSTOM-PROGRAMMED!

Anticipating that many people may wish to purchase additional copies of A Christmas Adventure as holiday gifts, BitCards is simultaneously releasing an enhanced version of the program, (ACA. C), designed specifically for this purpose. A utility provided with this version allows the buyer to customize the program, such that the recipient will discover several references to himself as he progresses through the adventure. When the player first activates Santa's computer, for example, he or she will be called upon by name to help solve the mystery of Santa's disappearance. And later, part of a note will be found in the reindeer's stable, praising the player's skills and character, and providing Santa with his or her home phone number. Through such personal references, the player will come to feel that he is an integral part of the adventure itself.

The customization routine also allows the sender to incorporate a personal holiday greeting into the culminating sequence of the adventure. Thus, the programmable version of A Christmas Adventure enables the buyer to produce a personalized software gift-and-greeting-card all-in-one. (BitCards will do the customization of the program to the customer's specifications if he lacks the appropriate computer to do it himself.)

$14.95 – Adventure alone (ACA.N)
$16.95 – Adventure + customization routines (ACA.C)
$17.95 – Customized version prepared by vendor
($US. add $2.25 shipping/handling – all versions)

Available on disk for Commodore-64 and for Apple II family and compatibles (48k RAM required). Apple version has over 20 pages of superb hi-res graphics including animation and zoom sequences. C-64 version is mainly text.

BitCards Inc.
30 W. Service Road
Champlain, NY
12919 514-274-1103

Hardware News

KAPRI Offers Advanced Graphic Accessory

SUN VALLEY – Kapri International has acquired a new supplier, PERSONAL PERIPHERALS, INC., that manufactures SUPER SKETCH, a sophisticated controller board for the COMMODORE 64. SUPER SKETCH retails for $59.95 US. The uses of SUPER SKETCH range from simple doodling to computer art, to a variety of applications in business. Unlike most video games, SUPER SKETCH utilizes the computer or video game unit as a creative tool that enhances anything that is drawn or traced. By simply moving the stylus control, as you would a pencil, SUPER SKETCH reproduces the movement on the screen. Compared with other video graphics products, SUPER SKETCH does more than joy sticks, paddle controllers and mouse controllers, and is less expensive than other pad products, such as KOALAPAD and CHALKBOARD. SUPER SKETCH can create a wide variety of business charts, or graphs, which can be saved on disk or conveniently transferred to another location via modem. PERSONAL PERIPHERALS is constantly producing new software. John Ovanessian, Product Acquisitions, stated, "We have been putting more emphasis on graphic accessories that can be used in education home or business. We have always been the leader in introducing new products to dealers."

For more information Dealers can call Kapri International at 1-800-22-KAPRI outside of California, or (818) 768-7888 within California, or by writing to 11671 Sheldon Street, suite K, Sun Valley, CA. 91352.

KAPRI International
Veritta Roberson
11671 Sheldon Street, Bldg. K
Sun Valley, California
91352 818-768-7888

Parallel Printer Adapter for the IEEE-488 BUS

Connecticut microComputer has announced a new Centronics printer adapter for the IEEE-488 (GPIB) bus. This device (the GPAD-C) allows any printer with a Centronics printer interface to be connected to any computer or controller with an IEEE-488 interface. The computers/controllers include those made by Hewlett-Packard, Tektronix, Commodore, Osborne and most others. Compatible printers range from low cost dot matrix types to high speed letter quality daisy wheel types.

The GPIB address is selected by a five position DIP switch. No special programming or software is required. The GPAD-C makes the printer look just like any other GPIB device.

The GPAD-C measures only 3 1/2 x 5 3/4 inches without cables. The GPAD-C includes two cables and a power supply. The cables allow six feet between the printer and the GPIB connection. The power supply means that power is not needed from the printer or computer.

The GPAD-C sells for $279.00 US. and is available from stock.

Connecticut microComputer Inc.
Shirley Fletcher
36 Del Mar Drive,
Brookfield, CT
06804 203-775-4595
Twx: 710-456-0052

Bits & Pieces

64 Quick Beep

The 64 is highly capable when it comes to sound generation, but it lacks a simple method of making a single beep, or ringing a "bell", as in the 40/8032 machines. The following POKEs will create a pleasant "ding", and can be used to get your attention after the computer has completed a certain task.

```
poke 54273,70: poke 54278,249: poke 54296,15:
poke 54276,17: poke 54276,16
```

Note: changing the argument in the first POKE varies the frequency of the ring.

Colour Bar

(Credit for this goes to someone out there with a stylish but somewhat unreadable signature. . . M.S. Renouf, perhaps?)

"Recently while developing a colour select routine for a program of mine (colour of background, border, sprites, etc.) I discovered that certain colours side by side were virtually impossible to see. So I took a look at the Reference Manual, and using some sophisticated analysis methods (trial and error) came up with the best possible general colour map using all 16 colours strung out in a line side by side. If you POKE the colours below in consecutive screen positions, you will get a most readable Colour Bar:

```
10 data 4,0,8,2,10,9,7,12,6,3,14,1,11,13,5,15
20 c=55295 : s=1023
30 for j=1 to 16 : read a
40 poke s+j,160 : poke c+j, a : next
```

Dazzler of the Month

You were waiting for it, weren't you? Just so we don't disappoint you, here's a screen dazzler for any BASIC 4.0 machine (4032/8032):

```
10 for i=47 to 57 : poke 59521,i : for d=1 to 100
   : next d,i : goto 10
```

Enter the above program and RUN it without clearing the screen. I call it "Attack of the Killer Program (in 3-D)".
P.S. It doesn't look like it's very healthy for the video circuitry, so don't keep it running too long.

Which Way Did He Go?

Here's an effect that owes more to the nature of human visual perception than it does to the graphics capabilities of your computer. On any 40 column machine, enter this program:

```
10 print"**********                    ";:goto 10
```

(Note that there are 20 asterisks and 21 spaces. The exact number of asterisks is not important, but there must be 41 characters altogether. You may use your favourite graphics symbol in place of the asterisks).

Now run it. You probably first see lines of asterisks running from the bottom of the screen to the top. Try fixing your eyes onto the centre of one of the bars of asterisks. See them moving slowly from left to right? You'll probably find the illusion flipping between vertically and horizontally moving bars.

The illusion is even more pronounced on 80-column machines. Use this program:

```
10 print"****************************************";
20 print"                                        ";
   :goto10
30 rem 40 asterisks, 41 spaces
```

The 80-column version creates slow-moving bars that are very difficult to see as moving vertically. A procession of diagonal bars is seen moving slowly from left to right.

Aquarius

While we're doing special effects, here's another dazzler for 80-column machines. It's based on the program by Giovani Polese in last issue's Bits & Pieces section, but works especially well on 80-column machines. In upper/lowercase text mode, enter:

```
10 print chr$(142)
20 print " EDCFRFCDE " ;:goto20
```

I won't describe the resulting effect, but it's better than you'd expect from such a small program. TRY IT!

Quick Note: The more sprites you have displayed on the screen of the 64, the slower the processor operates due to wait states from the VIC chip.

SHIFTing your WAIT

Here's a handy technique that comes to us from Rico Mariani of Downsview, Ont. To insert a pause in a program that can be enabled from the keyboard, use the command: WAIT 654,1 (C64) or WAIT 152,1 (40/8032), which will wait until the shift key is pressed. To enable a program halt at that point, engage the shift/lock key. This is a good way to synchronize a program with an external process: just disengage the shift lock key to continue program execution. This way you can be certain whether or not the program will halt at the WAIT statement, simply by knowing the position of the shift lock key.

Interrupt Key-Scanning

Sometimes it is desirable for some action to be performed any time a certain key is pressed. A routine may be set up to run during the interrupts, but the desired routine must be performed **once** when the key is depressed, not every interrupt as long as the key is held down. The following examples in assembler show an easy way to accomplish this.

C64 Example

The following assembler program, once initialized with SYS 49152, will change the border colour whenever the F1 key is pressed.

```
 10 *=     $c000     ;start at 49152 decimal
 20 keybd =197       ;key pressed
 30 ;set up irq vector
 40 sei
 50 lda    #<intrtn
 60 sta    $0314
 70 lda    #>intrtn
 80 sta    $0315
 90 cli
100 rts
110 ;
120 prevkey .byte 0
130 ;
140 intrtn  =*
150 lda    #4         ;keyboard code for f1 key
160 cmp    keybd      ;f1 key pressed?
170 bne    out        ;no, exit to system irq
180 cmp    prevkey    ;check previous key pressed
190 beq    out        ;exit if f1 pressed previously
200 ;
210 inc    $d020      ;increment border colour register
220 ;(any desired code could be inserted here)
255 ;
230 out    =*
240 lda    keybd
250 sta    prevkey
260 jmp    $ea31      ;system irq routine
```

40/8032 Example

The example program for the CBM will switch between graphics/lowercase modes when the up-arrow key is pressed. Use the 64 program above, making the following changes:

```
10  *=     $7000     ;start at 28672 decimal
20  keybd =151       ;key pressed
60  sta    $90       ;irq vector low
80  sta    $91       ;irq vector high
150 lda    #222      ;keyboard code for up-arrow
210 lda    $e84c     ;i/o register for graphics mode
212 eor    #2        ;flip graphics mode bit
214 sta    $e84c     ;store back into register
260 jmp    $e455     ;system irq entry point
```

Use SYS 28672 to enable this version.

File Ripper

Need to look through some disk file in a real hurry? Actually File Ripper is far too fast for the eye, but if you want to see what's at the end of a large file and have no time to waste, File Ripper will get you there quick! Once at the point you're interested in, you can use the regular slowscroll or pause keys (back arrow, :, RVS, CTRL, etc.). The 64 version would theoretically work on the VIC 20 but it hasn't been tested.

```
1000 rem file ripper 4.0
1010 for j= 634 to 774 : read x
1020 poke j,x : ch=ch+x : next
1030 if ch<> 15942  then print " checksum error "  : end
1040 sys 634
1050 data 160,   2, 169, 209,  32,  29, 187,  32
1060 data 226, 180, 169,   0, 133, 218, 169,   2
1070 data 133, 219, 169,   5, 133, 210, 169,   8
1080 data 133, 212, 169,   5, 133, 211, 162,   0
1090 data 189,   0,   2, 240,   3, 232, 208, 248
1100 data 134, 209,  32,  99, 245, 166, 210,  32
1110 data 198, 255,  32, 207, 255,  32, 210, 255
1120 data 165, 150, 240, 246, 165, 210,  32, 226
1130 data 242,  32, 204, 255, 160,   2, 169, 238
1140 data  32,  29, 187,  32, 228, 255, 240, 251
1150 data 201,  89, 240, 172,  76, 255, 179, 147
1160 data  70,  73,  76,  69,  78,  65,  77,  69
1170 data  32,  63,  32,  40,  32,  68,  35,  58
1180 data  70,  73,  76,  69,  78,  65,  77,  69
1190 data  32,  41,  32,   0,  17,  18,  65,  71
1200 data  65,  73,  78,  32,  63,  32,  40,  32
1210 data  89,  32,  32,  79,  82,  32,  32,  78
1220 data  32,  41,  32, 146,   0
```

```
1000 rem file ripper 64
1010 for j= 828 to 926 : read x
1020 poke j,x : ch=ch+x : next
1030 if ch<> 11254  then print " checksum error "  : end
1040 sys 828
1050 data 160,   3, 169, 131,  32,  30, 171,  32
1060 data  96, 165, 169,   0, 133, 187, 169,   2
1070 data 133, 188, 169,   5, 133, 184, 169,   8
1080 data 133, 186, 169,   5, 133, 185, 162,   0
1090 data 189,   0,   2, 240,   3, 232, 208, 248
1100 data 134, 183,  32,  74, 243, 166, 184,  32
1110 data 198, 255,  32, 207, 255,  32, 210, 255
1120 data 165, 144, 240, 246, 165, 184,  32, 145
1130 data 242,  32, 204, 255,  76, 116, 164,  70
1140 data  73,  76,  69,  78,  65,  77,  69,  32
1150 data  63,  32,  40,  32,  68,  35,  58,  70
1160 data  73,  76,  69,  78,  65,  77,  69,  32
1170 data  41,  32,   0
```

Quick Note: to disable character set switching with the shift/Commodore keys, PRINT CHR$(8);. To re-enable, PRINT CHR$(9);. An easier way to do it is to simply key CTRL-H or CTRL-I, respectively. Thanks to Jeff Goebel for this latter tip.

File Loader

When a number of program files must be loaded in succession, for example sprite or character definitions, machine code, or high resolution screens, this simple loading technique is a good way to do it:

```
10 A=A+1
20 ON A GOTO 30,40,50,60,70
30 LOAD " FIRST FILE  ",8,1
40 LOAD " SECOND FILE " ,8,1
50 LOAD " THIRD FILE  ",8,1
60 LOAD " FOURTH FILE " ,8,1
70 final statement - sys, load, goto, etc.
```

Since BASIC automatically performs a RUN (without clearing variables) after a LOAD from program mode, the files are loaded in succession. Any number of files may be similarly loaded, but make sure none of them are BASIC program files, or the loader program will get clobbered. As indicated, the last statement may be a SYS or other statement to start the program instead of a LOAD

ASCII/CBM Conversion

If you've ever tried to print to an ASCII printer, or receive from the RS-232 port on the 64, you're familiar with the problem: Upper and lower case are reversed. To solve the problem, use one of the following lines of BASIC to convert a single character, stored in A$.

ASCII to CBM

```
a=asc(a$+chr$(0)):a$=chr$(a+32*
(a>96 and a<123)-128*(a>64 and a<91))
```

CBM to ASCII

```
a=asc(a$+chr$(0)):a$=chr$(a+128*
(a>192 and a<220)-32*(a>64 and a<91))
```

Another difference between regular ASCII and CBM ASCII are the control characters. ASCII codes from 0 to 31 are reserved for special control characters, such as bell, line-feed, carriage return, backspace, etc. There is no direct correlation between ASCII and CBM control characters, but the conversion that must frequently be made is substituting the Commodore's "DELete" character (20) with ASCII's "backspace" (8). This may be done by adding the line,

`if a=8 then a$=chr$(20)` For ASCII to CBM conversion, or
`if a=20 then a$=chr$(8)` For CBM to ASCII conversion.
`a$=chr$(a+12*((a=20)-(a=8)) )` will convert either way.

Any or all control characters may be converted from ASCII by setting up a conversion string which holds the desired CBM characters, such as cursor controls, tabs, etc. The position of each character in the string should correspond to the ASCI code that it replaces. To make the conversion using a conversion string 32 characters long, the following line could be added to the conversion program above:

```
if a<32 then a$=mid$(c$,a+1,1)
```

This technique may also be used to convert Commodore control characters into ASCII equivalents. Usually, however, only the delete/backspace characters need to be switched.

Quick Note: the GET statement can accept more than one argument, as in:

GET A$,B$,C$,D$, or using GET#

Easy Disk Salvaging

All programmers live in constant fear of losing their irreplacable work due to death of a disk. This leads to paranoic backing up of important files, a very healthy activity. Occasionally, however, even the most paranoid among us hear the horrifying klik-klik-klik-klack-griiiind-klkklk which signifies – horror of horrors – a read error!

After you curse yourself for not having made a recent back-up, what can you do? Well, the first thing you should do before resorting to sector-reading, is to restore the disk jacket. A common cause of read errors is a disk jacket that's been squeezed tightly near the edges because of careless handling or storing. This creates too much friction between the disk and jacket, slowing the disk to the point where the drive can't read it. To fix this problem, carefully run the edge of the disk along the corner of a table to flatten it. Tapping the edge of the disk on a corner at many points also helps. This should spread out the jacket enough so that you can read the disk and make a copy of it onto a fresh one. That's the happy ending of this story, but there's also a moral: treat disks **gently** and don't hold them by the edges or squeeze them in any way. And that's not a fairy tale.

A Magic Number?

Examine the following program:

```
10 input " enter any number " ;n
20 print n
30 n$ = mid$(str$(n),2)
35 k = 0
40 for i = 1 to len(n$)
50 : k = k + val(mid$(n$,i,1))*3
60 next i
70 n = k
80 goto 20
```

As you can see, it just accepts any number, and then sums all of the digits in the number, first multiplying each digit by three. The result then becomes the new number, and the process repeats indefinitely, showing the new value, "N", every iteration. What's so special about it? Try it with any number you like, and see what happens after the first three or four iterations. If you can figure out the reason for this strange numerical omnipresence, your math students await you!

Safe VAL Function

To permit "idiot proof" entry of numerical values from within programs, it is best to input a string, and then convert it to a floating point value with the VAL function. This technique is still not 100% idiot proof, however. If, for example, the string "1e99" is entered, attempting to take its VAL would result in an overflow error – disaster! The VAL of the string can only be taken if the result would not exceed 1.7e+38. The following subroutine will take the VAL of the string V$ and put the result in V if doing so would not cause an overflow error. If an error would result, the flag "OVERR" is set, and V is set to zero.

```
50000 rem* safe val subroutine
50001 rem* input parameter  : V$
50002 rem* output parameters: V, OVERR
50010 overr = 0: ll = len(v$)
50020 for ii = 1 to ll: if mid$(v$,ii,1)<>" e " then next ii
50025 if ii>ll goto 50065 :rem* no " e " s, ok
50030 : mn = val(mid$(v$,1,ii-1)) :rem* mantissa *
50040 : for jj = ii + 1 to ll: if mid$(v$,jj,1)<>" e " then next jj
50050 : ex = val(mid$(v$,ii + 1,jj-ii)) :rem* exponent *
50060 : if ex + log(mn) > 38.53 then overr = 1: v = 0
        : return :rem* too high *
50065 rem— endif —
50070 v = val(v$) :rem* ok *
50080 return
```

Quick Note: An elegant way to create an infinite loop without using GOTO is:

FOR I = 0 TO 1 STEP 0 . . . NEXT I

Hardware Random Number Generation on the 64

From BASIC, it's easy to get random numbers (actually, pseudo-random numbers) using the RND function. If random numbers are desired in a machine language program, or if better randomness is desired, the SID chip may be used to supply them. The amplitude of the output waveform from voice three may be read from SID register 27, and if voice three is set up for high frequency noise generation, this value will be random. If that doesn't make sense to you, don't worry. Just set up the SID chip with:

POKE 54287,255: POKE 54290,129

Any time after that, a random number from zero to 255 may be read with:

PEEK(54299) from BASIC, or
LDA $D41B in assembler.

Round–up

Here's some fun with floating point round–off errors that works with either BASIC 2.0 or 4.0. Enter:

?5.99999999 (8 nines)

The result, as you'd expect, is just what you entered. Now try:

?5.999999999 (9 nines)

This time the result is 6, which is quite reasonable, since it is just rounded off. But now go one step further and enter:

?5.9999999999 (10 nines)

What? Not so reasonable this time (try it). Before you trash your computer for being so stupid, don't worry: the floating point routines are accurate to about 7 decimal places – that's one part in 10 million!

Quick Note: I%=I is a quick way of taking the integer part of a variable without using the INT function.

Prime Number Generation

I know, I know, generating prime numbers probably isn't high on your list of fun things to do with a computer. Notwithstanding, you'll probably get a kick out of the following method and accompanying program. You may learn something, too – don't forget, this *is* the education issue.

Math class flashback: a prime number is a number which can only be evenly divided by 1 and itself. Thus, 11 is a prime because it has no factors other than 1 and 11, but 9 is not, since it's factors are 1, 3, and 9.

If you were asked to write a subroutine to determine whether or not a number is a prime, a reasonable approach would be to divide by each whole number from 2 up to the argument, and if none are found to divide evenly. . .

a=d/q: if a<>int(a) then. . .

. . .then the number is a prime. You could go one step further for efficiency and only try numbers up to the square root of the argument, since factors above that would be redundant. Now, here's another problem: write a program which prints all prime numbers from 1 up to a given value. You might be tempted to pass integers from one up to the limit to the above subroutine, and print the number if it is a prime. That will of course work, but there's a better, not–so–obvious way.

The prime number generation technique used here comes to us courtesy of Eratosthenes (E–RA–TOS'–THENEEZ), of Athens, Greece. Around 200 B.C., Eratosthenes had this great brainstorm for generating primes. The technology of the time did not include computers, so a long line of small stones was probably used to do the trick. Actually, it's no trick – here's how it works (We'll use a computer instead of the stones. Less work).

1) First we set up an array containing all zeros. The array must have as many elements as the maximum prime we want to generate. This array could be represented by a string of bits since only 0 or 1 is needed in any element.
2) We initialize the process by printing the first prime, which is 1, and setting the first element in the array accordingly.
3) The array is scanned from the current prime until a zero is found. The position of the next zero in the array represents the next prime, and it may be printed out. It will be 2 in this case.
4) The array element pointed to by this prime is set to a 1, and, in this first case, every second element thereafter is also set. In the next iteration (prime=3), the third element and every third thereafter would be set.
5) Steps 3 and 4 are repeated until the next prime is greater than the maximum prime to be found.

The technique may look strange on initial examination, but if you think about it a bit, you'll see why it works. By setting a given element, you're ruling it's position out as a prime, and thus the multiples of every prime are being ruled out as primes. This effectively cancels out all numbers which have factors (other than 1 and the number itself).

Why go through mental gymnastics to generate primes? Well, this technique spits out primes so fast, you won't be able to read them as they fly by on the screen. The first few primes come out slowly, then get faster and faster as they approach the specified limit. The below program prints all the primes up to 100 (there are 26 of them) in about 4 seconds. An optimized BASIC program does it in less than three, much of that time taken just to print the numbers out.

Bits & Pieces presents the following program to generate primes. Try different numbers for the maximum prime, and see the effects. Each array element is an integer variable instead of a single bit. A bit–oriented routine would allow

higher primes to be generated since less memory would be required per prime, but would run slower due to increased processing.

```
100 rem* prime number generation
110 rem* using "sieve of Eratosthenes"
120 :
130 input "maximum prime";max
150 :
160 ti$ = "000000"
165 dim sieve%(max + 1)
170 number = 0: rem* prime count     *
180 prime = 1 : rem* first prime is 1 *
190 sieve(prime) = 1
200 :
210 for mloop = 0 to 1
220 : print prime
230 :
235 : rem* find next prime *
240 : for np = 0 to 1
250 :  prime = prime + 1
260 :  np = -(sieve%(prime) = 0)
270 : next np: rem* until zero found
280 :
285 : rem* set multiples of prime *
290 : for set = prime to max step prime
300 :  sieve%(set) = 1
310 : next set
320 :
330 : number = number + 1
335 : mloop = -(prime> = max)
340 next mloop
350 :
351 tme = ti: rem* stop timing *
352 print: print
360 print number;" primes generated."
370 print tme/60;" seconds taken."
```

The above program was written so that you can easily understand the process, and modify it if neccessary. If you're too lazy too type the whole thing in, here's a simplified and slightly shorter version. Note that what is gained in brevity is paid for in clarity and versatility.

```
1 rem* sieve of eratosthenes *
2 input "maximum prime";m : dim s%(m + 1) : p = 1
  : for k = 0 to 1 : print p
3 for i = 0 to 1 : p = p + 1 : i = 1-s%(p) : next : for s = p to
  m step p : s%(s) = 1 : next : k = -(p> = m) : next
```

The disadvantage of using the sieve to generate primes is that the amount of memory available limits the highest prime that can be produced. On the 64, the above routines can go as high as about 19000. (Yes, I tried it. The highest prime was 18979. No, I don't know how long it took). Using a single bit per element, you should theoretically be able to get sixteen times that. A simpler modification would be to change the routine so that it doesn't set the prime locations after it prints the primes. This would leave the array intact so that the list could be printed again without re-setting the elements. (That's the correct approach to take: my sieve routines are a bit non-standard. Also, traditional sieve algorithms start with an array of ones and zero out the factors, but since DIM zeros the array free of charge, doing it this way means we don't have to initialize every element in the array).

I hope you found the above piece (or was it a bit?) interesting, even if it wasn't of *prime* importance.

Quick Note: the use of integer instead of floating point variables results in slower, not faster, execution times

Useless Fact:

A program with a line number zero can be RUN by typing anything beginning with the letters R-U-N, such as: RUNNING AWAY, RUN FAST, etc. Also, if you type R-U-N on a line containing other text, you need not type a colon to delimit the RUN command. And if your fingers occasionally go spastic after typing R-U-N, you need not delete that "sufferin suffix" before hitting Return. If there is no line zero in the program, such antics are rewarded with an "?UNDEF'D STATEMENT ERROR".

Useful Fact:

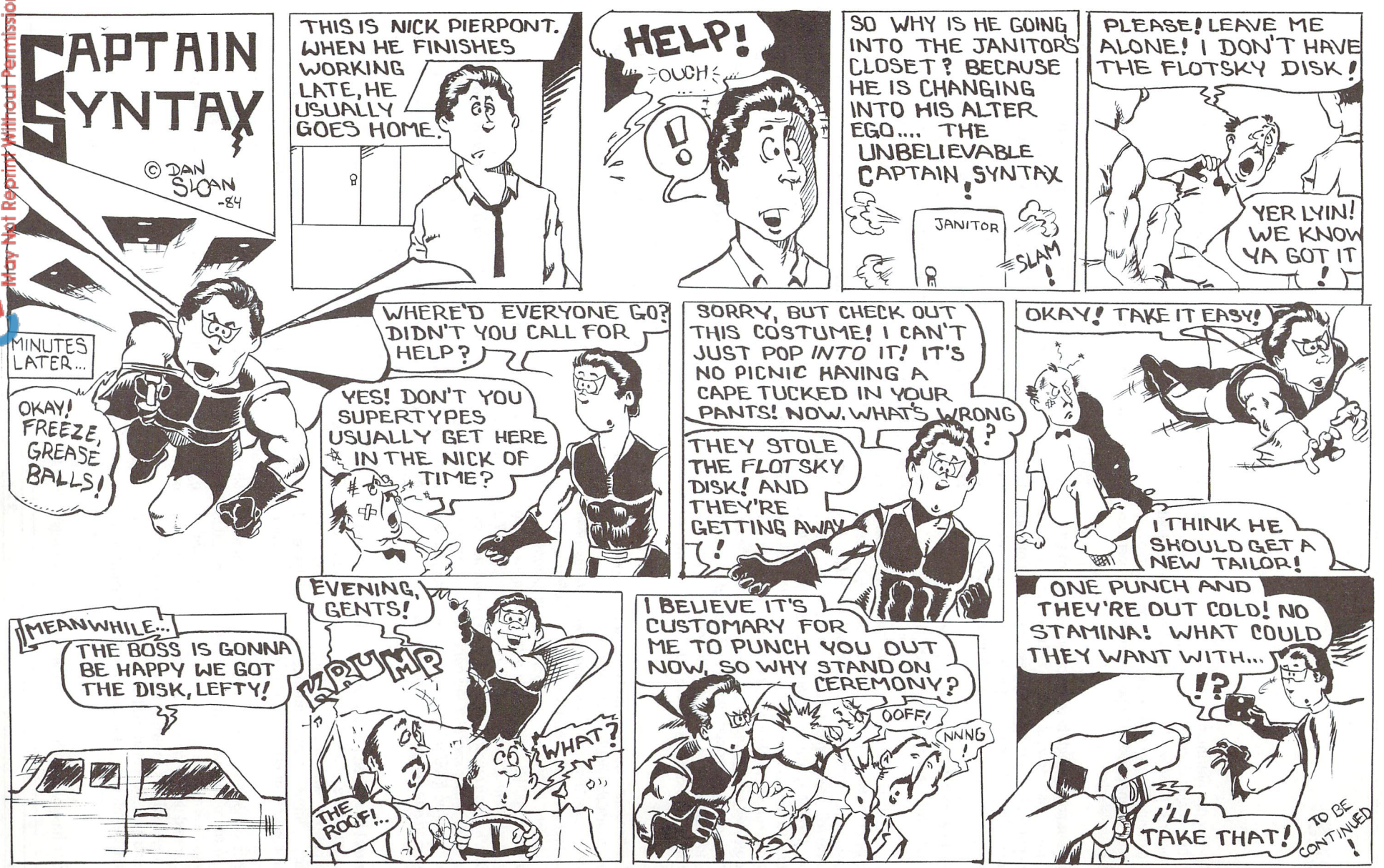
Yes, some obscure little bugs in BASIC can actually be "features". When documenting GOSUBs in a program, instead of using a REM, as in:

```
GOSUB 10000: REM* INPUT THE DATE
GOSUB 20000: REM* EXECUTE OTHER ROUTINE
GOSUB 30000: REM* ETCETERA, ETCETERA
```

You can fit more comments on the line by leaving out the REM, and following the destination line number with any character, for example:

```
GOSUB 10000 'INPUT THE DATE
GOSUB 20000 'EXECUTE OTHER ROUTINE
GOSUB 20000 'ETCETERA, ETCETERA
```

The apostrophe (') allows remarks beginning with numbers, and makes an attractive REM substitute. This tidy method of annotation works with GOTOs, too.



Letters

OP Oops: Mr. George Shirinian's review of word processing programs left out a major feature of 'Paperclip'; it won't run on recently manufactured C–64's. That is, many disks recently sold don't run on current 6510's, apparently as a result of over–enthusiastic use of unimplemented opcodes in making past versions of the program 'secure'.

The problem is common enough here in the boondocks; can it be unknown in Toronto? As a new subscriber to Transactor, I had the impression you published more facts and less hype than Commodore's house magazines or their commercial equivalents. Was I mistaken? Michael R. Wilson, Saskatoon, SK

ASO, RLA, LSE, RRA, AXS, LAX, DCM, INS,
ALR, ARR, XAA, OAL, SAX, SKB, SKW

These are operations that can be executed on most 65XX CPUs but are not officially part of the instruction set. As you said, they sure can foul up a program, especially when MOS acknowledges their existence, but also tells you to avoid them for fear of obsolescence in future chips. Well, I spoke to Batteries Included, and they informed me that there was never any problem with Paperclip in the coding department, just the key department. It seems that some older Commodore 64's were being snuck into Canada via the US, and they were not working properly with the Paperclip protection key. With the addition on one capacitor, the problem has completely cleared up. And as such, us folks in TO were never informed of the problem.

We have tried to become a no hype magazine, and firmly believe we have accomplished this goal. We are not owned or controlled by Commodore in any way, and our advertisers list has been condensed to a precious few. If ever you find a bubbling review, and you will find a few, its because the product is pretty terrific and deserves a few bubbles. We receive quite a few packages in the mail for review, and a few of these are OK, so we review them. If a product is terrible, we won't waste precious space extolling its non–existent virtues. Why waste space in the pages of our magazine with trash, if it could be filled with another enlightening article ? Our philosophy, and one that we hope you agree with.

WordPro Hints: I have been using Wordpro Plus 64 for almost a year now. In scanning the tables of Editing and Printing Functions which accompanied Mr. George Shirinian's article on wordprocessing programs for the Commodore 64 (Vol.5 Issue 1) I noticed that there were four errors in the tables vis a vis Wordpro.

The first error is understandable since the function is undocumented in the Wordpro manual. Neither the manual nor the Table indicate that one can access the bottom of text being worked with. I accidently discovered that by pressing the control key (so that the letter C on the line display is shown in reverse video) and then the left arrow key (tab key), the cursor will be moved to the bottom of the text. It would be interesting to find out why the manual does not cover this.

The second error has to do with the capability to delete word(s). This can be done by pressing the following sequence of keys:

<control><d><w><return>

The letter w is pressed as many times as there are words to delete.

The third 'error' may in fact not be an error. The Printing Functions table indicates that Bold printing is not an option. According to the manual, Bold printing (on letter–quality printers) is activated by pressing the control key and the 8 key. To deactivate, control–9. I own a Smith–Corona TP–II and have found that this does not work on the printer (which does boldface using Wordstar). I have found that the only way to 'bold print' is to force it by defining the code which my printer will recognize as a backspace, typing the code commands as many times as there are letters that I want boldfaced, and retyping the letters. One has to take care, since all key strokes will be counted as letters, so the margin for that line is affected. I have also found that although the manual indicates super and subscripts are supported, it does not work with the Smith–Corona.

Finally, the Printing Table indicates that enhancement mode is available. According to the manual, this is how Commodore printers recognize the underline commands. I own a Commodore 1525–E. The enhancement/underlining command does not work on it.

I just wanted to clear these points up. I use Wordpro constantly and find, overall, that it is a fine program. I am constantly amazed at the fact that Wordpro does its job as well as top of the line, dedicated wordprocessors.

Patricia Ann Wilkinson, Falls Church, Virginia

Thank you for writing us with some rather keen observations. The first error you describe is not really an error, but rather a neat way to use a documented function for an undocumented result. In fact, the reason "GOTO End Of Text" is not documented is because it is not implemented. When Control TAB is keyed, WordPro attempts to fill the first "Variable Data Block" with information retrieved from the Extra Text area. WordPro begins searching for the first Block, but if there are none the cursor ends up at the end of text and gives control back to the user. So if you don't use Variable Blocks, Control TAB will appear to do a ,"GOTO End". With Variable Blocks the results are somewhat different.

Club Plugs: We would like to add our name to your list of Commodore Users Groups. We are located in Germany, in the romantic city of Heildelberg. Our membership consists of U.S. Military and civilian employees of the military who are stationed here in this area and their dependents. We are just recently organized, however the response to our call for members has been gratifying.

Robert H. Jacquot
Sec/Treas.
Commodore Computer Users Group Heildelberg
PO Box, General Delivery
A.P.O. New York, N.Y. 09102

Maybe you would be kind enough to mention CLUB64, which could be best described as a Commodore 64 software user group, in a future edition of your publication.

As we have not yet fully decided the range of services that we should offer to our members, we are seeking constructive suggestions. It should be mentioned that we have a particular interest in hearing from users who cannot attend club meetings because they are living in remote parts, or because they are disabled or even because they do not have the time.

We have already established a library of high quality public domain programs, most of which have been checked and debugged. At present, ten disks are available and every one of them includes between ten and fifteen programs which may be copied and distributed to friends, members or users groups, schools, etc. We hope to add at least two disks per month and maybe, if we get enough suitable material, issue a regular newsletter on disk.

All 64 users with disk drives are invited to make use of our library. For anyone who only wishes to use the library there will be no membership fee but there will be a charge of 5.00 pounds per disk. This includes the cost of packaging and postage to any part of the British Isles (postage to others parts of the World will be extra), and the overheads involved in obtaining and copying programs.

As we are a non–profit making group, we do not, at present, have the manpower or resources to enable us to make programs available on tape. But if the demand is great enough for such a service, we will try to find a way.

We are interested in obtaining news, information, product details, programs or any information suitable or inclusion on our proposed disk newsletter. Would it be possible for software producers to supply short samples or trailers for inclusion on our disks?

For further information, please write:

Brendan Conroy
c/o Upper Drumcondra Road,
Dublin 9, Ireland.

Program Files To Sequential Files On Disk: I have a SuperPet and a copy of Petcom, which is a telecommunications package (an excellent one). My problem is that the SuperPet works in true ASCII, and although I can write a routine to convert sequential files to ASCII, or PETCII as needed, I don't know how to convert program files. This is obviously a problem in that many programs I send or receive to users of Pets of C64's will end up a scrambled mess. I hope that you can be of some help.

Robert Dray, Peterborough, Ontario

In answer to your problem, I am assuming that the programs in question are in Commodore BASIC completely. Anything other than this, like embedded machine code, would be a royal pain. To transform a BASIC program to a sequential file listing, you simply LIST the program to disk. Try the following :

```
open 8,8,8,"0:filename,s,w" : cmd8 : list
```

Then, after all the excitement has died down on the disk drive, close up the file:

```
print#8 : close8
```

You will have achieved a proper listing, as it would have appeared to the screen, in a sequential data file. If you have to transform machine language programs to sequential, it may be worth converting them into a BASIC loader with data statements, then listing them to disk.

PS. Thanks for the article on Telecommunications. Its terrific, and we will be running it in our Telecommunications issue a few issues from now.

So, You Want Us To Remain High Level: . . .The real reason why I'm writing, however, is in response to something I read in a magazine. The article was extolling the virtues of your magazine – how it provides much needed technical information that is very hard to find elsewhere – when suddenly it mentioned that the Transactor will be making a shift to the beginning computer user. Please, *please,* **please,** don't do it!!! I had hoped that I had finally found a magazine that caters to the advanced Commodore user. I'm so sick of the useless gobbledygook that I get in Compute, Run, etc.

In the event that the technically–oriented Transactor becomes an endangered species, please send me information on ordering back issues.

Scott Burns, Urbana, Illinois

Gobbledygook?! Aaaarggh, NEVER!! We are a high level magazine, and will remain so 'till our brains start to decay from too much activity. In all honesty though, we considered including more beginner level material to give us a larger target market, but reversed that decision in favour of a more stable and long term market.

Low level magazines are easy to come by, and as such, beginners can easily get more than enough info to help them along the way at the start. But time will advance, and beginners will progress into the intermediate and advanced stages of computer understanding. Exit stage left the regular mag, centre stage, Transactor. We neglected the fact that the portion of the market we don't service will be the same slice left unserviced by the beginner mags in only a short time.

We neglected another aspect too. In order to publish beginner articles, they must first be prepared. This was boring. So the higher the level, the happier we became. Now before you start flipping through and labeling some of our articles as perhaps not so advanced, remember this: the articles we choose for publication are meant not only to be typed in, but to provoke further thought. In each issue we attempt to include articles that provide you with ideas for making a program in another article better. For example, you might take Garry Kiziak's machine language sort and swap it into Phile Master by Bob Drake to make the sort portion of the program faster. So in the short run you get a program, but in the long run you get the tools for making more programs more useful more often. And we think that's more important.

Some people don't understand everything we print, just yet, but challenge is what makes the world revolve. Nothing feels better than to grasp a situation that before was without meaning. We run at a high level because we like it, and we are sure that we are not alone. Once again, we assure you, we will not decay into another regular collection of low level dribble.

The MANAGER Column

by Richard Evers

For this issue we are going to change our pace a little. As you know, the past few issues have discussed only C64 Manager applications, which our mail proves to be pretty popular. Well, the Manager has been around for some time now, and as much as we like the new Manager, the old one was more versatile in the programming department. For this reason, this issues column has been donated to the workings of the CBM Manager, just for those many people who feel left out of this column. Next issue you can expect to here from Don Bell again, and the C64 Manager express will continue on its merry way. Hope you enjoy this issue.

To start the ball rolling, how about a story relating my experiences with the Manager for reasons other than data base management. In the past, I have written a few fairly major business applications for the CBM 8032. With these, as with most business applications, relative files were the only way to go for data storage. In one particular application, the full capacity of two 8250 diskettes for both the various programs and the data, were used. On these poorly overworked diskettes were found relative pointer files, further relative sub-pointer files, and relative data files, with one reaching the extreme of 14,742 records in length. The trick is, the software was all written from the ground up, and the design work was being performed even while the program was being written. That particular application was helped along quite a bit by some careful planning with the Manager in the initial record design stage.

The normal creation process of relative records can be at times confusing. You decide what information is going to be in each record, then record it on a sheet of paper. Next, you figure out the length of the file and exactly how many records you can expect to use, then you go about creating the records. Later, if updates are needed, which they always are, that piece of paper has to be found back, and the update has to begin all over. With the Manager, these problem were easily bypassed. By entering the Create A File option I was able to custom create the relative records exactly as required. Comments and fancy formatting could also be included to help clarify the matter for later updates, if needed. Once completed, the file was created to the desired length, with the Manager placing an astra at the start of each record, and padding the balance of the record with spaces. A very nice touch.

One problem did get in the way immediately, but it was easily straightened out. Each record created was within the limit of one block less 2 bytes, as imposed by the CBM disk system. The trouble was, my total file length exceeded what the Manager would allow. Because of the 8050 drive unit and its shortcomings with its relative file capacity, it just can't handle over 183 K of relative data storage, my 8250 was held back. The 8250 drive can handle 1.04 meg of relative data, but the Manager didn't know it. It was left up to me to tell it.

By poking around a bit, an answer was found to this problem. In filename 'create', line #3070 was changed to allow for as many records as the disk could handle. Just by looking at the line, you can see how a cap was put on the system :

```
3070 rem if mx>182880 then mx=182880
```

Same with line #13070 in filename 'fileman'. With this code REM'd away, I was free to create files as large as required. The next problem was not as extreme, but still just as important in this application.

For future update purposes, the screen files were invaluable. They could be copied from the working disk onto a screen file disk, then scratched from the working disk. If ever an update was required to re-create a file, the Manager would only require the screen file for this purpose. As such, a diskette full of screen files was pretty useful. This all was fine until my final, largest file was in need of creation. Then trouble reared its ugly head. Pure, unadulterated greed took over my being and prompted me to want the room that the screen file consumed for relative data storage. Actually, it wasn't greed. The system was in need of every byte possible, and as such the screen file was in the way.

My monster file had a record length of 69 bytes, and it was left to me to get as many 69 byte records on disk as possible. For this reason, the data was given one entire diskette in which to reside. The balance, relative data records, pointers and the programs themselves, were all put on the other diskette in drive zero. By allocating one entire disk for this single file, there was 1.04 meg of room available. Enter, the problem.

A Manager screen file takes up 26 blocks on disk, which translates into 6604 bytes of data that could have been used for relative storage. My task, to get rid of the screen file before it was written to disk.

Now, considering that the problem was encountered while in the 'create' a file section, file name 'create' was once again loaded in. Without going into greater depth, the answers were found on two lines, lines #672 and #3060. Look below, and you will find the reason for the hold back :

```
672 \s,fs$,"6500","7dff"
3060 fb=x+y*256-1 : close1 : fb=fb-26-1-1
     : fb=fb-int(fb/120) : mx=fb*254
```

Line #672 performs the function of saving to disk the screen file, of which was comprised of data residing between $6500 and $7dff in memory. This line had to be rem'd out completely.

Line #3060 is responsible for the calculation of the number of free blocks available on disk for the storage of the relative file. A screen file is comprised of 26 blocks, therefore the calculation fb=fb-26-1-1 had to be changed to read : fb=fb-1-1

With these bits of code altered accordingly, my pursuit of 'the big one' could be accomplished, a total file length of 14,742 records!

Subroutine Eliminators 64

Jeff Goebel
Georgetown, Ont.

Once again, I pick up where I left off two issues ago and continue to bring you some more pokes and peeks that do the work of entire subroutines. This time, with a collection for the Commodore 64 owners. Stick with me, and you'll be scratching old disk routines and clearing up valuable disk and memory space.

First let me look back to a previous issue for a moment. In volume 4, issue 6, I talked about a powerful poke that took away the question mark and did not allow you to hit return out of a pet input statement. (POKE 16,0) Lots of people liked it and wanted to know how to do it on their Commodore 64. I looked around and came up with:

```
poke 19, 64
```

For those of you who don't have the original issue, I'll briefly describe it again. The above poke will somehow stop you from exiting an input prompt unless you enter something. It is indeed a handy trick. The format is as follows:

```
10 poke19,64 : input " Enter Name: ";a$ : print : poke19,0
```

Please note that there must be a "PRINT" following the input, and you must remember to reset the poke to 0 after you are finished inputting. Otherwise, you'll never get any line feeds. One other quick note: The poke has no effect unless you have a text prompt in your input statement.

Before I continue with more subroutine eliminators, let me first throw this quick subroutine your way. Although it is contradictory to do so in an article like this, I feel this is a short enough program to be included here. It simply prints the contents of your 64 screen to the printer using only three lines of BASIC. The key is in line 20, which is the routine to transfer Commodore screen poke codes to regular ASCII character string values. I'm sure you'll find other uses for that routine as well.

```
10 open4,4 : x = 1024 : for s = 1 to 25 : for t = 1 to 40
   : a = peek(x) : x = x + 1
20 a = a + 128*(a>127) : a = a-64*(a<32ora>95)
   -32*(a>63anda<96)
30 print#4,chr$(a); : next : next : close4
```

Now on to the serious subroutine eliminators. How many UN-NEW programs have you read lately? Be honest now, we all know the ones. Many of us have one or two we've typed in and they sit unused on a disk somewhere. Well, here is a short one which will restore a program to the point where it may be listed (attempting to edit a program restored by this method will result in a rather exciting crash)

```
poke 2050, 1 : sys 42291
```

It even works if you've typed in SYS64738 and reset the computer. There is a catch, however: I've used the command before with no problems, as long as your program isn't huge. Somewhere just over 8k, you'll start running into screwy problems. If it's a big program you've just accidentally NEW-ed, then maybe you can search through all those other disks.

Many other authors have tried to condense the Commodore 64's music chip into simple easy to use pokes. I've seen one program that was only 84 characters but it produced one beep. Now it's my turn! I admit that in the condensing I have lost a lot of musical control. I don't worry about waveforms or attack, decay etc, but sometimes all you need is a little tone. If a beep takes 84 characters, I can do a click in only 24 (even less if I abbreviate my POKE command). If I add a short loop, I can create a pleasant buzz. Enter line 10 below and run it for a click, then add lines 5 and 15 to hear a buzz.

```
5 for t = 1 to 50
10 poke54296,15 : poke54296,0
15 next
```

Although you can't change the buzzes pitch, you can adjust the volume (since all I'm doing is turning it on and off anyway). Just change the 15 to any smaller number.

```
poke 650, 255
```

This poke allows all your keys to automatically repeat when you hold them down for a second.

I also have a few pokes left over that could have gone into the my SPIFFY LISTINGS article which appeared in the last issue. I think they'll fit in here as well:

```
poke 774, 0
```

This vanishes your program listings. The statement numbers still appear, but the statements don't.

```
poke 774, 141
```

This poke simply disables your ability to list at all. By the way; poke 774,26 resets these to normal.

```
poke 775, 141
```

This will reset your computer to a RUN-STOP/RESTORE state whenever you try to list. It resets with 167. I'm sure these locations will yield some other interesting effects if you play around with various values in them.

```
poke 808, 237
```

This will disable not only the list, but also the capability to RUN-STOP/RESTORE your computer. It resets with poke 808,242

```
poke 818, 32
```

This will disable the save function of the Commodore 64. It must be reset with poke 818, 237 before any WRITE operation is successful.

I hope that this article has not offended any of the authors of subroutines I have eliminated here today. If I have, I'm truly sorry, and hope to offend more of you next issue with still more SUB-ROUTINE ELIMINATORS.

Office Automation For The Nineties

Major B.L. Olmstead
Victoria, BC

In the early 1950's there were approximately 1,000 computers in use in North America. By 1976 there were more than 220,000 large computers and three-quarters of a million microcomputers in use. By 1980 over ten million microcomputers were in use and their proliferation was growing at a rate of 25% per year. This is expected to continue throughout the decade.[1]

With this enormous proliferation of micros and the dramatic reduction in cost of main frame hardware and their increased capacity, the sheer momentum of technology will force a revolution in the office of the nineties. Virtually every office manager and secretary will have an intelligent terminal on their desks. This will induce a radical change in the information flow and traditional staffing of positions. Vocal and telephone communications, face-to-face meetings and many written messages currently processed will disappear. In their place will be shared data bases, common electronic mail/messaging systems, and common files. This will open up new avenues of communications among the individuals in an organization both internally and externally. In addition, the individual user will be able to supervise more closely the dissemination of their messages.

New pathways of communication will be established and interpersonal roles will change. Clerical workers will ne displaced and become an unknown entity as such highly routine work as filing, mail sorting, mail delivery and voluminous copying tasks are handled electronically by computer systems that perform these tasks by the mere push of a button or the command of a voice.

Out of the labour force will come a group of paraprofessionals who will assume semi-management positions in the office, using computer technology to monitor and control a variety of processing chores currently performed manually. Managers and executives will use teleconferencing and computer conferencing to replace time consuming and expensive travel. They will have decision support systems to handle budget preparation, sophisticated modeling tasks, maintenance of their calendar and to give them access to a variety of internal and external data bases.

The new era of electronic mail/video text services will provide graphs, charts and photographic images with text messages[2]. Documents will be annotated with voice comments and voice messages will refer to text[4]. Spread sheets from personal computers and information in public and private data bases will become a transparent part of the information infrastructure. The integrated message facility (IMF) will provide for a single interface for controlling all message facilities and the telephone system[1]. For example, a window on the user's work station will provide a complete status for the user on:

A. The telephone (incoming calls, camp-on, queing etc.)
B. Text/graphic and voice messages (electronic in basket)
C. Operator messages
D. Secretarial intercom

Another major accessory in the office of the nineties will be storage and retrieval facilities for information using optical disc technology[6]. It can store both analog and digital data of virtually every kind we now collect and use, on a single medium. Standard computer data, voice, music, motion picture and video footage as well as photographs and all manner of complex graphics (including color) may be stored on the optical disc. An added plus is that scanning of such material into a system takes a fraction of the time required to key it in.

In terms of performance, a 12 inch optical disc can contain a sheet of standard typewriter paper (54,000 square millimeters) on just .5 millimeter of disc surface and store up to 10-to-the-tenth-power bits – a number equal to the number of seconds in 317 years[6].

The work station to support these users will provide a transparent interface to all the communications facilities needed. Video text services will be an asset of growing importance to business along with voice-mail systems[2].

In the nineties there will be an indigenous system for computer assisted learning embedded in the work station software to teach the user to operate these complex systems. The value of this training method is that the user is actively involved, using the tools he will be using on the job. The user will also be able to proceed at his own pace, which will reduce the inequities between slow learners, gifted persons and the handicapped[7]. In addition to the embedded primary and adjunct CAI, user-friendly menu-driven displays with extensive "HELP" features will assist in the learning process.

For many users, this work station will meet the needs of individuals for private personal computation, data storage, graphics, mail, electronic funds transfers etc. Even now, some businesses and universities expect their personnel to have personal computers which can be linked to their main facilities.

In the nineties, flat-panel display technology will allow senior management to carry portable computers in their brief case and link to their offices to conduct business as required[8]. By the end of the century I would expect to see the automated office staffed by technicians and the "staff" operating from their residences. The societal changes which this could bring about are heady indeed. If you could be employed by a firm in Victoria and live in Hawaii or

Mexico or anywhere a data link could be established, the impact on our way of life would be greater than the industrial revolution produced. When linked up with interactive teletext on home computers and interactive videodisc entertainment systems, current living patterns may be completely disrupted[10]. Whether societal resilience is adequate to withstand such dramatic changes is a matter for conjecture. Computerization certainly points to a more reclusive posture for individuals which is contrary to human nature. Many people feel that nothing, not even the electronic revolution can substitute for being at the "center of the action" in the market place. "We communicate on a different level when we're within a few feet of each other," says J. Craver, recently retired futures research manager at Monsanto Co[12].

Most of what has been said to this point emphasizes the increased efficiency and productivity of the knowledge worker[9]. Given the future disruption of traditional systems of work and management which have evolved over the past decades, many problems can be foreseen.

The first, and probably most important are the "Quality of work life" and personal health aspects of using the work station. Quality of Work life (QWL) is considered the "democratization of work" and is designed to improve productivity[13]. Major changes in traditional management attitudes and "consultation" will have to take place to satisfy knowledge workers especially when operating from remove sites. In addition, many knowledge workers still perceive there are health problems associated with radiation from work stations. Although there is no scientific proof of radiation hazards and workers complaints can always be traced to ergonomic and environmental deficiencies, the perception must still be dealt with[14].

A real health problem may be induced psychologically should these sophisticated systems fail on a massive scale. Power "Blackouts or Brownouts" over large areas or deliberate sabotage could cause over reactions to the techology such as that resulting from the Three-Mile Island nuclear plant accident. That accident caused little real damage but was instrumental in virtually closing down further development of nuclear energy projects. Will societal resilience and national economics be able to accept massive disruptions of interrelated computer systems or their communications facilities without over-reaction against the technology? Only time will tell!

The question of National Security for information systems also looms large both from a physical security and a security of information point of view. Even now, international data flows are being restricted by some countries using foreign data protection laws[15]. This will cause international businesses to decentralize which may cut into their productivity, profitability and flexibility. It could also create concentrations of power for those countries that can afford this technology and have the expertise to employ it.

Maintenance of these massive interrelated systems will also create enormous problems unless systems are designed for maintenance[16]. Special technology and organizations will have to design built-in automated diagnostics and corrective mechanisms for these complex interrelated applications[17].

In conclusion, the optimists believe that the automated office will result in greater freedom and individuality and a more humane and personalized society. The productivity increases using computer-assisted manufacturing techniques and Robotics can only increase our standard of living, shorten the work week and increase leisure time to enjoy life.

The pessimists come to the conclusion that computer technology will dominate our lives as a society and as individuals and sweep us along in a tide over which we - the harassed and exposed victims of a depersonalized and dehumanized process that places greater value on efficiency than on the more noble qualities of life - shall have little control. They are also convinced that computer monitoring of individuals through the automation of the office and home and its concomitant dangers to privacy, is truly an Orwellian prophesy come true.

I believe that the automated office of the nineties will integrate itself into society because of its productivity capabilities and its sheer momentum. Whether you consider it will improve society or not, I feel that basically it will be the same story as in the past: Your children will be better off than you are, and their children will be better off than they are.

References:

(1) Finn, B. "Interfacing People with their Machines" AFIPS Conference Proceedings (Vol 52) p.355
(2) Davis, D.B. "US Business Targeted as Major Videotext Market" Mini-Micro Systems, Sep 1982 p.145-151
(3) Hasiuke, K., Konishi, K., Asami, T. and A. Kurematsu. "Text and Facimile Integrated Terminals" National Telecommunications Conference Record 80 (Vol.3) 1977 p.60:5.1 - 60:5.5
(4) Maxemchuk, N.F. and Wilder, H.A. "Experiments in Merging Text and Stored Speech" National Telecommunications Conference Record 80 (Vol.1) p.16:1.1 - 16:1.6
(5) Finnigan, Paul "Voice Mail" AFIPS Conference Proceedings (Vol.52) p.373
(6) Saxton, W.A. and Edwards, M. "The Optical Disc come of Age" Office Administration and Automation 1983 Vol 44 No. 5 p.74
(7) McKenzie, J., Elton, L., and Lewis, R. "Interactive Computer Graphics in Teaching" Halstead Press, New York 1978
(8) Weissmann, T. "Battery Powered 16-Bit Briefcase Micro" Ed. Canadian Data Systems Vol 15 No.6 p.30
(9) Johnson, B., and Taylor, J. "Innovation in Word Processing" Interum Report to the National Science Foundation Grant #ISI 811079, 1982
(10) Martino, J.P., "The Future of Telecommunications" Current, p.28 June 1979
(11) Smith, A., "All the News that fits in the Data Bank", Saturday Review p.18-19
(12) Sanders, H., "Computers in Society" McGraw-Hill Inc. 1981 p.592
(13) Daven Port, J., "Whatever Happened to QWL?", Office Administration and Automation May 1983 p.26-29
(14) McGurrin, H., "Video Display Terminals - A Guide to Managers and Supervisors" Occupational Health and Safety Bulletin 16-83 July 1983, p.1-7
(15) Turn, R., "Transborder Data Flows - Privacy Protection and Free Flow of Information", Arlington, V.A.: AFIPS Press 1979
(16) Lientz, B.P., and Swanson E.B. "Software Maintenance Management", Reading, Mass,: Addison-Wesley, 1980
(17) Richardson, G.L. and Butler, C.W., "Organizational Issues" AFIPS Conference Proceedings Vol 52 1983 p.155

Dynamic Expression Evaluation

Chris Zamara

I have been asked several times if there was a way to allow user-input of an expression, and have a program evaluate it. For example, an excellent aid for mathematical education on any level would be a program which would plot a function, given the function and the domain (range of x values) over which to plot it. Such a program would be extremely useful to illustrate given functions and allow experimentation, without tedious calculation and plotting on graph paper.

It sounds like a simple program to write, but the problem comes when the user must enter the function to be evaluated. If a string is entered, such as:

"SIN(X)+COS(2*X)"

. . .how can the program evaluate it? One method would be to halt the program, allow entry of an equation such as "Y=SIN(X)+COS(2*X)", and then CONTinue the program, which would use the variable 'Y'. The problem with this approach is that the possibly computer-naive user is suddenly thrust into the real world, out of the program. One wrong move on his part (like pressing "home" by mistake), and the program may die in a very ungraceful manner. Another technique, probably most used by programmers, is to just change a function definition before running the program. For example, entering:

```
1 DEF FNY(X)=SIN(X)+COS(2*X)
```

. . .would enter the function definition as line 1 in the program, allowing evaluation by subsequent calls to the function, as in "Y=FNY(X)". This technique may be useful to programmers, but once again, it is too error prone for a non-programmer, who can't be expected to modify and run programs in the normal course of usage.

Besides the above specific example, there are many occasions when it would be nice to enter an expression instead of a number, as in the following examples of prompts and responses.

PROMPT	RESPONSE
maximum range?	*pi
age in days?	1*365
length in kilometres?	*1.6
speed in km/h?	0*0.6
yearly revenue?	tly*12
monthly income?	rly/12
length of hypotenuse?	qr(a*a + b*b)

As you can see, sometimes entering an expression just saves you from having to whip out your calculator, and sometimes it lets you use internal program variables which have unknown values.

As is often the case, it seems machine language is the solution to our dilemma. What is needed is a routine which accepts a string expression – representing the function to be evaluated – and returns the result in the form of a floating point variable. The program to accomplish this, called "EXPEVAL", appears in BASIC loader form in Listing 1, and in assembler form in Listing 3. In its present form, it runs on the 64, but could be easily converted for other machines. The program is accessed through the USR function in BASIC, so the USR vector must be set up once before using it. The BASIC loader program does this automatically with:

"POKE 785,0: POKE 786,192"

After the vector is set, the program is called with the USR function from BASIC, using a dummy argument (the USR function can't use a string as a parameter). A comma, then a string expression, follows the USR command. The string expression represents the function to be evaluated, and is usually a simple string variable. A specific usage example follows.

```
A=5: F$="3*4+A"
PRINT USR(0),F$
```

After executing the above statements (assuming the USR vector has been previously set up), the value 17 will be

printed – the result of the expression held in F$. More complicated expressions may be evaluated, for example:

```
XP = USR(0),F$ + " SIN(X) + " + STR$(Y)
```

However, it's best to stick to using just a string variable name as the expression to avert the possibility of a "?FORMULA TOO COMPLEX ERROR", which seems to happen sometimes with more complicated expressions.

A brief explanation of how EXPEVAL works follows. You can just enter and RUN the program in Listing 1 to use it, so if you're the type that thinks of a subway ride when you hear the word "token", don't worry about the next paragraph too much.

The program makes extensive use of ROM routines to perform the following operations:

1) Check for a comma
2) Evaluate a string expression
3) Tokenize the resulting string
4) Evaluate tokenized expression, storing result in FPAcc#1

Using the ROM routines to accomplish these tasks is pretty straightforward, but the tricky part was tokenizing the line. The tokenizing routine seems very reluctant to work with any text that isn't in the input buffer, since all code is tokenized as it is entered. In order to keep it happy, I had to save the contents of the input buffer, move in the string to be tokenized, call the tokenizing routine, and finally, restore the input buffer (including the CHRGET pointers) to its original state. Not a very elegant way to do it, but it works. A good source of information about the ROM routines and how to use them is the article, "Getting BASIC To Communicate With Your Machine Code" By Darren Spruyt in the last issue. this article is also useful for cross-referencing ROM routines among PET, VIC, and 64 ROMs, and may be used to convert EXPEVAL to work on a machine other than the 64.

At any rate, the routine seems to work with no ill side effects, other than the previously mentioned FORMULA TOO COMPLEX errors, but it is quite basic and could be improved upon. A good addition to the program would be error traps in case a nasty, non-evaluatable function was passed to it. Without that, the mathematics application described at the beginning of the article would not be truly "idiot proof", since a bad function entry would kill the program. (An all purpose error trap method is on its way to a future Transactor). The code is fully relocatable, and uses a temporary storage area of 80 bytes ("BUFSTOR"), which lives in the cassette buffer. Just remember that the POKEs to the USR vector must change as the program is moved in memory.

Listing 2 shows a very simple application of the mathematics function-plotting program, with each plotted point displayed as an asterisk on the 25 X 40 grid of the screen. Note that the program uses EXPEVAL for every numerical input, so expressions are allowed as responses to all prompts. The y=0 line runs across the centre of the screen, and the x vales at either side of the screen are determined by the input to the "DOMAIN?" prompt. The first prompt asks for the scale. All function results are multiplied by the scaling factor to get screen units. As a quick demonstration of the program, try using a scale of 10, a domain of 0, 2*π, and the function SIN(x).

Using EXPEVAL in programs opens up all kinds of new possibilities, and hopefully the program itself helps to explain the mysterious ROMs a little bit better. The more you understand the inner workings of the machine, the easier it is to write useful little utilities, and the sharper your programming skills become. And that's a fair EVALUATION.

Listing 1:BASIC Loader For EXPEVAL

```
100 rem* data loader for " EXPEVAL " *
110 :
120 cs = 0    :rem* checksum
130 os = 49152:rem* object start
135 :
140 for rd = 0 to 1: rem.. data loop ..
150 : read b
155 : if b>0 then poke os,b: os = os + 1: cs = cs + b
160 : rd = -(b<0)  : rem.. until b<0 ..
170 next rd
180 :
190 if cs<>1330 then print " * checksum error * " : end
200 :
210 poke 785,0: poke 786,192
220 print " ** call EXPEVAL with: "
225 print " ** 'USR(0),string variable'
230 :
240 end
1000 data  32, 253, 174,  32, 158, 173
1010 data  32, 143, 173, 160,   0, 177
1020 data 100, 170, 200, 177, 100, 133
1030 data 251, 200, 177, 100, 133, 252
1040 data 160,   0, 185,   0,   2, 153
1p50 data  60,   3, 177, 251, 153,   0
1060 data   2, 200, 192,  80, 208, 240
1070 data 169,   0, 157,   0,   2, 165
1080 data 122,  72, 165, 123,  72, 169
1090 data   0, 133, 122, 169,   2, 133
1100 data 123,  32, 121, 165, 169,   0
1110 data 133, 122, 169,   2, 133, 123
1120 data  32, 158, 173, 104, 133, 123
1130 data 104, 133, 122, 162,  79, 189
1140 data  60,   3, 157,   0,   2, 202
1150 data  16, 247,  96,  -1
```

Listing 2: Function Plotting Program

```
10 rem************************
20 rem* function plot routine:      *
30 rem* this routine uses the       *
40 rem* usr function "EXPEVAL"      *
50 rem* to plot a user-supplied     *
60 rem* function.                   *
70 rem************************
80 :
100 scrn=1024 : colr=55296
105 poke 785,0 : poke 786,192
106 rem* set up usr vector *
110 for main=0 to 1 step 0
115 :   input "Sscale";sc$
120 :   skale=usr(0),sc$
130 :   input "domain (start x, end x)";d1$,d2$
140 :   d1=usr(0),d1$
150 :   d2=usr(0),d2$
160 :   input "function";f$
170 :   inc=(d2-d1)/40
180 :   8=d1
185 :
190 :   for xp=0to39
200 :      y=usr(0),f$
210 :      y=12-int(y*skale+.5)
220 :      sp=xp+40*y
230 :      if sp<0 or sp>999 then 245
240 :         poke scrn+sp,42 : poke colr+sp,1
245 :      rem.: endif ..
250 :      x=x+inc
260 :   next xp
265 :
270 fork=0to1 : geta$ : k=-(a$<>"") : next k
280 next main
```

Listing 3: The assembler source code for EXPEVAL

```
110 ; "EXPEVAL"
120 ;****************************
130 ;* evaluate expression:          *
140 ;* syntax is USR(0),stringname   *
160 ;****************************
162 ;assembled with PAL
165 *=$C000
170 ;
180 chrptr  =$7a ;pointer
190 strptr  =$fb  ;used to point to string
200 bufstor =828 ;temp. storage in tape buffer
210 ;
220 jsr  $aefd     ;check for comma
230 jsr  $ad9e     ;evaluate string expression
240 jsr  $ad8f     ;check for string
250 ;
260 nmr =*
270 ldy  #0        ;get string pointers:
280 lda  ($64),y   ;string length
290 tax            ;
300 iny            ;
310 lda  ($64),y   ;string start, low
320 sta  strptr    ;
330 iny            ;
340 lda  ($64),y   ;string start, high
350 sta  strptr+1
360 ;
370 ldy  #0
380 inbuff =*
390 lda  $0200,y   ;put string into input buffer
400 sta  bufstor,y ;
410 lda  (strptr),y ;and save current contents
420 sta  $0200,y   ;
430 iny            ;
440 cpy #80        ;
450 bne inbuff     ;
460 ;
470 lda  #0        ;mark end of expression
480 sta  $0200,x   ;
490 ;
500 lda  chrptr    ;save pointers
510 pha            ;
520 lda  chrptr+1  ;
530 pha            ;
540 lda  #0
550 sta  chrptr
560 lda  #2        ;point to
570 sta  chrptr+1  ;input buffer
580 jsr  $a579     ;crunch tokens
590 lda  #0
600 sta  chrptr
610 lda  #2
620 sta  chrptr+1
630 jsr  $ad9e     ;evaluate expression
640 pla
650 sta  chrptr+1
660 pla
670 sta  chrptr    ;restore pointers
680 ldx  #79       ;and input buffer
690 rebuff =*      ;to original state
700 lda  bufstor,x
710 sta  $0200,x
720 dex
730 bpl  rebuff
740 rts
750 .end
```

Compound Interest And You

by Richard Evers

While writing this program, an odd thought popped into my brain regarding a savings account of mine that has its interest compounded daily by the bank. In case you are not sure of what interest compounded daily means, please let me explain. The current interest rate that the bank pays during that particular day is divided by 365 then multiplied with the amount currently held in the account. This interest is then added to the present savings total, to be compounded further the following day. When ever I have my bank passbook updated, the value given is in a dollar and cent value, without fractions of a cent. The question is, where do the fractions go ? Does the bank round everything off to the nearest cent, then balance up this profit or loss at the end of the fiscal year ? There is a pretty good chance that this will balance out almost perfectly, but imagine for a second that the banks do not think the same way as we do about our money.

Contemplate the idea that the remaining fractions of cents were chopped off, instead of rounded off to the nearest cent. The difference in value for most customers of the bank would never be noticed, but added together they would amount to quite a bit of cash. If this money existed, it would probably be used for mortgages and loans, of which the interest charged is often double what is given for the same amount. Just think how quickly these fractions of cents compounded together then loaned out at ridiculous bank rates would increase in value. The total amount would be staggering. I wonder how this amount would be explained to the auditors if they ever stumbled across it ?

This far fetched hypothesis has been planted in your head not to prompt you to overthrow the banks, but to keep you on your toes about your own money. There is so many ways to use your money to produce or lose money for you, that it is often hard to comprehend. Just imagine if the government chopped off the remainder instead of rounding it off when it came time for your tax return. Massive dollars with no one noticing that it was missing. More food for thought.

The following program is something that I found to be fairly handy. It will calculate interest compounded either yearly, monthly, weekly or daily on a user specified amount at an interest rate and term also specified by the user. As the calculations are made, they are printed to the screen, just to keep you informed during the process. When the task has been completed, all the specifications that you entered earlier are displayed, along with the final amount calculated, the amount of interest earned and the percentage of increase that you realised on your investment. A clean and effective little routine. One note though : Before taking the results generated by this routine as the gospel truth, please remember that Commodore microcomputers are not the most accurate calculators in the world. They are pretty close, but not perfect.

If further variations are found for this quick collection of bits, or some answers to my rather obtuse questions are discovered, then write in and tell us all about it. Whatever the story, we hope this program is enjoyed by all who are *interested.*

```
100 rem : save " @0:compound calc " ,8
    : verify " 0:compound calc " ,8
105 rem ****************************************
110 rem * richard evers - transactor magazine - 1984 *
115 rem * calculates compound interest of values      *
120 rem * using specifications given by user.         *
125 rem ****************************************
130 :
135 print chr$(147) " compound interest calculation "
140 print
145 input " initial value " ;v
150 input " rate of interest " ;p
155 print " interest compounded : "
160 input " (y)ear, (m)onth, (w)eek, (d)ay " ;cr$
165 if cr$ = " y " then lp = 1    : pr = (p/lp)
170 if cr$ = " m " then lp = 12   : pr = (p/lp)/100
175 if cr$ = " w " then lp = 52   : pr = (p/lp)/100
180 if cr$ = " d " then lp = 365 : pr = (p/lp)/100
185 :
190 input " period of investment in years " ;yr : if yr = 0
    then 190
195 :
200 vlu = v
205 for year = 1 to yr
210 for compound = 1 to lp
215 vlu = vlu + (vlu*pr)
220 print " year " year " period " compound " value " vlu
225 next compound,year
230 :
235 print " original value                    : " v
240 print " interest rate                     : " p
245 print " compounded                        : " cr$
250 print " investment period in years : " yr
255 print " final value                       : " vlu
260 print " total interest                    : " vlu-v
265 print " percent return                    : " ((vlu-v)/v)*100
270 end
```

GETSTRING For The 64

Dave Gzik
Burlington, Ont.

Often enough you'll write yourself some sort of a program that requires entering information via the keyboard. Well for some of you a simple INPUT statement will do. If your like me, building strings with a GET statement is more appealing.

The reason I use a GET routine is because you maintain full control over what characters to accept, how many of them, and what type they are. Lets say you have written a mailing list program that uses a relative file for its storage. The records have to be the same length whether each of the fields are different. In the same manner the field inputs are restricted to length also. If you were using an INPUT statement, the length cannot be restricted within the input but can be altered after the input. This is somewhat tedious and wastes programming space. Therefore to cut down some steps we can use the GET statement. This will allow us to accept only what we want making the task a little easier.

The only drawback with the GET routine is speed. In BASIC the get routine is slow and most people can out type the computer. This is not the way to have it. The idea is right but we do need the speed. To achieve speed, we must use machine language.

This is the reason for GETSTRING.

Getstring only allows two types of input: numeric which allows the keys 0 – 9 and the decimal point and alphanumerics which allow the previous plus the character codes in the range of 46 to 90. The selection of string type can be made from BASIC with a POKE. The length of the string is also determined with a POKE from BASIC.

Getstring uses the cassette buffer for it's string storage thus allowing for a maximum string length of 191 characters. The routine will only accept the delete key if there is at least 1 character in the buffer. This routine will not accept any more than the preset length of characters and can only be terminated by a carriage return.

Getstring sits at location $C000 in the 64 and with its storage in the cassette buffer does not take up any BASIC programming area.

You can put GETSTRING anywhere you like, but I put it at $C000.

To use this routine requires a little setup prior to calling it. First the length must be set by POKEing a value between 1 and 191 into location 253. The type of string must be set by POKEing a value 0 for numerics or a 1 for alphanumerics into location 254. The position on the screen where the input is to take place is at the discretion of the user. Once all this is completed a SYS 12*4096 can be executed and the routine will take over.

After the string has been entered we need to get it into a BASIC variable. This is accomplished by finding the number of characters entered via a PEEK to the 'Y' INDEX Register [PEEK(782)]. With this number we set up a FOR NEXT loop and PEEK the characters out of the cassette buffer and build a BASIC variable.

For repeat inputs the above setups must be done for each one.

Below is a sample BASIC program with the loader that uses GETSTRING. Also for you machine language buffs, included is a source listing of the routine.

This routine is generic and can be used on any Commodore Computer with minor cosmetic changes.

If you wish to relocate this routine somewhere else in memory, just change the poke address and SYS address to whatever address you may choose. This routine will not accept any other characters except for the predifined range.

```
1 rem read data into $c000
5 for x=0 to 142 : read a : poke 12*4096+x, a : next
9 rem start of basic program
10 poke 253, 15 : poke 254, 1
20 print " Sqqqqq ";
30 sys 12*4096
35 for x=0 to peek(782)-1
36 a$=a$+chr$(peek(828+x))
37 next
40 print " qq ";: poke 254, 0 : poke 253, 5
50 sys 12*4096
60 for x=0 to peek(782)-1
65 b$=b$+chr$(peek(828+x))
70 next
90 print a$; "  "; b$
900 end
999 rem m/l loader data
1000 data 169,  60, 133, 251, 169,   3, 133, 252, 160,   0
1010 data 169, 166, 145, 251, 200, 192, 192, 208, 249, 160
1020 data   0, 177, 251,  32, 210, 255, 169, 157,  32, 210
1030 data 255, 152,  72, 169,   0,  32, 228, 255, 240, 251
1040 data 170, 104, 168, 138, 201,  13, 208,   6, 192,   0
1050 data 240, 235, 208,  76, 201,  20, 208,  22, 192,   0
1060 data 240, 225, 136,  32, 210, 255, 169, 166, 145, 251
1070 data  32, 210, 255, 169, 157,  32, 210, 255, 208, 207
1080 data 196, 253, 240, 203,  72, 165, 254, 208,   6, 169
1090 data  58, 133, 250, 208,   4, 169,  91, 133, 250, 104
1100 data  24, 201,  48, 144,   2, 176,  10, 201,  46, 240
1110 data  10, 201,  32, 208, 172, 240,   4, 197, 250, 176
1120 data 166, 145, 251, 200,  32, 210, 255,  76,  21, 192
1130 data 145, 251, 169,  32,  32, 210, 255, 169,  13,  32
1140 data 210, 255,  96
```

```
100 ;getstring    (may 84)
110 ;
120 ;dave gzik, burlington ont.
130 ;
140 ;get routine for use with a basic program.
150 ;this routine uses the cassette buffer for
160 ;transfer of string data to a basic variable.
170 ;maximum length of string, type of string must be
180 ;flagged in basic prior to entry on this routine.
190 ;the maxlength must be in the range of 1-191 and string type
200 ;can be either alpha or numeric by either 1 or 0 respectively.
210 ;exit out of this routine can only be acheived by a carriage routine.
220 ;
230 ;
240 ;variables and constants
250 caslo    =   $fb
260 cashi    =   $fc
270 maxlen   =   $fd
280 strtyp   =   $fe
290 outchr   =   $ffd2
300 getchr   =   $ffe4
310 temp     =   $f9
320 temp2    =   $fa
330 ;
340      *   =   $c000
350 ;
360 ;setup routine
370 ;
380          lda  #$3c       ;setup string area
390          sta  caslo      ;in cassette
400          lda  #$03       ;buffer
410          sta  cashi      ;for data transfer.
420 ;
430          ldy  #$00       ;set pointer to zero.
440          lda  #$a6       ;solid cursor for storing.
450 ;
460 fill
470          sta  (caslo),y  ;fill
480          iny             ;cassette
490          cpy  #$c0       ;buffer with
500          bne  fill       ;solid cursor.
510 ;
520 ;start of get routine.
530 ;
540          ldy  #$00       ;set pointer to zero.
550 ;
560 start
570          lda  (caslo),y  ;print cursor
580          jsr  outchr     ;and reset
590          lda  #$9d       ;print
600          jsr  outchr     ;position.
610 ;
620 get
630          tya             ;save y
640          pha             ;onto stack.
650          lda  #$00       ;get a
660 wait
665          jsr  getchr     ;non null
670          beq  wait       ;character.
680 ;
690          tax             ;store char tempory.
700          pla             ;retrieve y
710          tay             ;off the stack.
720          txa             ;retrieve character.
730 ;
740          cmp #$0d        ;check for <cr>
750          bne  next       ;yes or no?
760 ;
770          cpy  #$00       ;is it valid?
780          beq  get        ;ie. must have received
790          bne  exit       ;at least one character.
800 ;
810 next
820          cmp #$14        ;check for
830          bne  next2      ;delete key.
840 ;
850          cpy  #$00       ;is it
860          beq  get        ;valid? (y/n).
870 ;
880          dey             ;decrement string pointer.
890          jsr  outchr     ;print the delete.
900 ;
910          lda  #$a6       ;store and
920          sta  (caslo),y  ;print
930          jsr  outchr     ;a new cursor.
940 ;
950          lda  #$9d       ;reset print
960          jsr  outchr     ;position on screen.
970 ;
980          bne  get        ;go for another char.
990 ;
1000 next2
1010         cpy  maxlen     ;is it at its
1020         beq  get        ;maximum length?
1030 ;
1040         pha             ;save char.
1050         lda  strtyp     ;determine what type
1060         bne  alpha      ;must be
1070         lda  #$3a       ;a 0 or 1.
1080         sta  temp2      ;setup for validity range.
1090         bne  store
1100 alpha
1110         lda  #$5b       ;setup will be
1120         sta  temp2      ;for alhpanumerics.
1130 store
1135         pla             ;get char off stack.
1140 ;
1150 ;store character.
1160         clc             ;set carry off.
1170         cmp #$30        ;is it
1180         bcc  chkdec     ;less than zero?
1190         bcs  skip       ;no then skip.
1200 ;
1210 chkdec
1220         cmp #$2e        ;is it a
1230         beq  putchr     ;decimal point?
1240 ;
1250 space
1260         cmp #$20        ;is it a
1270         bne  get        ;space
1280         beq  putchr     ;yes or no?
1290 ;
1300 skip
1310         cmp  temp2      ;is it greater
1320         bcs  get        ;than range!
1330 ;
1340 putchr
1350         sta  (caslo),y  ;store the character
1360         iny             ;and point to next location.
1370         jsr  outchr     ;print character.
1380         jmp  start      ;go for another character.
1390 ;
1400 exit
1410         sta  (caslo),y  ;store <cr>.
1420         lda  #$20       ;print a
1430         jsr  outchr     ;space.
1440         lda  #$0d       ;print a
1450         jsr  outchr     ;<cr>.
1460         rts             ;and back to basic.
1470 .end
```

Sorting On The Commodore 64 And The PET/CBM

Garry G. Kiziak
Burlington, Ont.

Are you a teacher who is creating your own marks program? Are you creating your own mailing list program. Are you developing any type of data base system? If the answer to any of these questions is yes, then sooner or later you are going to need a sort routine.

Most people who work on these types of programs will already have such a routine – usually a Shell sort, a Heap sort, a Quicksort, or perhaps some variation of one of these. The sort routine will in all likelyhood be written in BASIC – and there are many good reasons for doing so.

1. It is usually a simple matter to code the routine, especially when there are so many listings of these routines in the various magazines. You don't even have to understand how it works, simply code it in.

2. It is frequently easy to alter the routine to suit your own needs.
   - you may require a routine to sort numbers, perhaps strings, perhaps both
   - you may require them to be sorted in ascending order or descending order or perhaps both
   - you may require singly subscripted or multiple subscripted arrays

Here is such a BASIC listing. It will sort the string array N$ into ascending order (i.e. alphabetically). The number of elements to be sorted is assumed to be stored in the variable NUM. The program is written as a subroutine so it can be called from anywhere within your BASIC program by a GOSUB 100. The algorithm used is a variation of the SHELL sort, a very efficient and popular sort.

```
100 Z=1 : IF NUM=1 THEN RETURN
110 Z=3*Z+1 : IF Z<NUM THEN 110
120 Z=(Z-1)/3 : IF Z<1 THEN RETURN
130 FOR I=Z+1 TO NUM : J=I-Z
140 IF N$(J)>N$(J+Z) THEN T$=N$(J) : N$(J)=N$(J+Z)
    : N$(J+Z)=T$ : J=J-Z : IF J>0 THEN 140
150 NEXT : GOTO 120
```

Figure 1.

Some of the advantages of a BASIC routine have been listed above. Perhaps the greatest disadvantage is speed. As long as you are sorting less than one hundred elements of an array, the speed is tolerable. But when you have to sort several hundred elements, the speed can become a definite problem – especially when sorting string arrays.

A very general rule of thumb for many sort algorithms (but not all) is:

'If it takes a certain amount of time to sort a specified number of elements in an array, then it will take about four times as long to sort twice that number of elements.'

So for example, if 100 elements can be sorted in 30 seconds, it will take about 2 minutes to sort 200 elements, about 8 minutes to sort 400 elements, and so on. Sixteen hundred elements would take a little over 2 hours. Actually, if you are sorting string arrays, there is a further complication. All of the swapping of array elements (in line 140 in the routine above) can force a process called 'garbage collection'. This by itself can take several minutes each time it is required. I have heard of sorts that have taken several hours to complete on a micro like the Commodore 64.

When this begins to happen, the 'happy programmer' becomes grumpy and looks for an alternative to speed up the process – perhaps machine language. For comparison purposes, the routine in this article will sort a hundred elements in the blink of an eye and it will sort 1600 elements in about 10 to 13 seconds.

As soon as you resort to machine language, you begin to lose some flexibility (unless of course, you create it yourself, exactly the way you want it). For example, it becomes difficult to adjust the routine to meet your needs. Instead, you find yourself adjusting your program to meet the requirements of the routine. This may not be too bad if the routine is the least bit versatile. If it is stringent and inflexible however, it may not be worth the effort.

How versatile should it be? Well, I have seen situations that required:

1. An integer array to be sorted in either ascending or descending order.
2. A real array to be sorted in either ascending or descending order.
3. A string array to be sorted in either ascending or descending order.
4. A doubly (or even multiply) subscripted array to be sorted in ascending or descending order. (The array could be real, integer, or string.)
5. One array to be sorted, and whenever elements of that array are swapped, corresponding elements of another array are also swapped.
6. One array to be sorted, but the original order was not to be disturbed. (This usually requires the introduction of a second array, called an index, which keeps track of the sorted order.)

. . .and in fact many others. Clearly no one routine can include all these features (Again this reenforces why a BASIC sort is desirable. The sort routine given in Figure 1. can easily be altered to meet any of these situations.)

To see what features I included in my sort routine and understand why I chose those features, you have to examine the method that I frequently use for storing data.

To illustrate, let's consider a Mailing List. Essentially what you want to do is store certain pieces of information about each person on that list; things such as: Surname, First Name, Street Address, City, Province or State, Postal or Zip Code, Telephone Number, and perhaps the number and names of children, birthdays, wedding anniversaries, etc.

In data base management terms, all this information, taken together for an individual, is called a record. Each individual piece of information (e.g. Surname, City, etc.) is called a field within that record, and the collection of all the records is called a file. There are many ways you could store these records in a program that you create. Likely you will use real arrays, integer arrays, string arrays, or some combination of all of them.

My personal choice, frequently (though certainly not always), is to use a singly subscripted string array and to have each element of that array correspond to a single record of the file. This means that all the fields within a record will have to be grouped together somehow and made into a single string of characters. To do this, I first have to decide on the maximum number of characters within each field (called the length of the field). Let's use the sample values in Figure 2. (Note: below is a sample of the data that might be entered into each field of a record.)

Field	Length of Field	Sample Data
Surname	15	Kiziak
First Name	12	Garry
Street Address	25	2381 Duncaster Drive
City	15	Burlington
Province or State	15	Ontario
Postal or Zip Code	7	L7P 3V9
Telephone Number	12	416-335-4837

Figure 2.

Clearly, the number of characters in any given field will not always equal the length of the field. In that case, I make them equal by padding the field with blank characters (either on the left or on the right, depending on what I want to do with that field). All these fields are then joined together into a single string and put into a single element of the array. The important point about padding is that it forces a given field to always start at the same character in all records. Thus the first three records in a file, stored in the array N$, might look like this:

Note that Field 2 always starts at character 16, Field 3 starts at character 28, and so on.

Now that I have all my data organized into records, I will want to arrange them into some logical order; that is, I will want to sort them. For example, it would be logical to sort them in alphabetical order by Surname (i.e. sort on Field #1). However, it might also be logical to sort them by City (if for example, I want to extract those individuals who live in Burlington). It might, in some circumstances, be advantageous to sort them by Telephone Number; in particular, to be able to pick out those individuals with the same area code. I am sure you can think of other reasons for being able to sort on any other field.

For my programs, I would like my sort routine to be able to sort an array by looking at some specified set of characters – at the first 15 characters in the example above if I wanted it sorted by Surname, at characters 53 through 67 if I wanted it sorted by City, at characters 90 through 101 if I wanted it sorted by Telephone Number, etc.

Surname	First Name	Address	City	Province	Zip Code	Tel #
1 15	16 27	28 52	53 67	68 82	83 89	90 101

It would also be nice to be able to sort it into either ascending order or descending order depending on the circumstances.

My sort routine will do all of this and more. Before we look at the other features, let's look at how it is used in a BASIC program. Let's assume that the routines have already been loaded into memory (see the sample programs to see how this is done). First, assign the starting address of the routine to a variable. I like to use SRT, so I will include the assignment:

```
SRT = 12*4096 + 256
```

near the beginning of my program. Let's also assume that we have 150 records in our file stored in the array N$ (i.e. N$(1),N$(2),. . .,N$(150)). To sort the records by Surname, I would use the following command in my program:

```
SYS SRT,N$,1,150,1,15,A
```

The first entry after the starting address of the routine, in this case N$, is the name of the array to be sorted. You can use any name you wish, but it must be a singly subscripted string array. If the array does not exist, you will get the error message ?ARRAY ERROR IN . . .

Record	Field #1	Field #2	Field #3
N$(1)	KiziaK	Garry	2381 Duncaster Drive
N$(2)	Montgomery	Katherine	241 First Avenue
N$(3)	Williams	Bill	111 Ridgeway Road

The next two entries (i.e. 1 and 150) are the first and last elements of the array to be sorted (i.e. we want to sort from N$(1) through to N$(150)). The next two entries (i.e. 1 and 15) specify which characters to look at when sorting – in this case characters 1 through 15, or the first field. The last piece of information required is the letter A to indicate an ascending sort.

If we wanted to sort the records by City, then we would use the command:

```
SYS SRT,N$,1,150,53,67,A
```

and if we wanted to sort by Telephone Number, we would use

```
SYS SRT,N$,1,150,90,101,A
```

Here are some other possibilities.

```
1) SYS SRT,N$,1,150,16,27,D
2) SYS SRT,N$,1,150,63,82,D
3) SYS SRT,N$,60,95,1,15,A
```

1) will sort the records by first name, but in descending order.
2) will sort by Province or State in descending order.
3) is a little different. It will not sort the entire array. It will sort just records 60 to 95 (i.e. N$(60) to N$(95)) by Surname in ascending order. This might be required for example, if earlier in the program you somehow determined that all the records in CALIFORNIA were situated between N$(60) and N$(95) inclusive and you now just want these records to be sorted by last name.

This last example illustrates the fact that the entire array does not have to be sorted. You can sort a part of an array simply by specifying the first and the last elements in the array that you want to sort. You can even start at the zeroth element, and you can use variables for any of these values.

The last example also brings out another point. Suppose you had the following two commands right after the other in a program.

```
SYS SRT,N$,1,150,1,15,A
SYS SRT,N$,1,150,63,82,A
```

The first command will sort the records into alphabetical order by Surname, and the second command will then sort them into alphabetical order by Province or State. In all likelihood there will be several people who live in the same state (California for instance). The second command will of course group these records one after the other. Since the first command ordered them alphabetically by name, it is natural to expect that all the names in this group from California will be in alphabetical order. Unfortunately this is not necessarily true. (A sort routine that allows you to sort on one field, and then on another and still maintain any order that was established by the first sort is called a stable sort routine.) Unfortunately, most of the popular sort routines like the Shell Sort, the Heap Sort, and the Quicksort are not stable.

For this reason, I have included another feature in my routine that will allow you to sort on several fields simultaneously. The command:

```
SYS SRT,N$,1,150,63,82,A,1,15,A
```

will sort the records first of all in ascending order by Province or State. Secondly, whenever matches occur in that field, it will sort those records into ascending order by Surname. You can sort on as many fields as you like and you can even change from ascending order in one field to descending order in another. For example

```
SYS SRT,N$,1,150,63,82,A,1,15,A,90,101,D
```

will do the same as the above, but when matches occur in both fields (e.g. all the Smiths in California) it will then sort those records in descending order by Telephone Number.

The speed of this sort routine is achieved partly (perhaps mostly) because it is written in machine language, but also because it avoids that dreaded 'garbage collection'. During the sort, the strings are not actually swapped like in a BASIC sort. Instead the pointers showing where these strings are located are swapped and this does nothing to force a garbage collection.

The Programs

Listing 1

This is the assembly listing of the sort routine that works on the Commodore 64. It is a translation of the routine given in Figure 1. Of course the extra features (like ascending or descending sorts, the ability to sort on many fields, etc.) have been incorporated as well. For those of you starting to learn assembly language, there is much to learn from this listing including sixteen bit arithmetic and passing parameters to a machine language program (as well as the idea of optional parameters). I hope you find this educational as well as useful.

Listing 2

This is a program that contains the sort routine in DATA statements. If you have a disk, you can run this program and it will create a PRG file called SORT 64 that you can load in and use from your own programs (see listings 3 and 4 for examples).

If you are using cassette, these routines will have to remain in DATA statements. Delete lines 10000 and 10005 and replace them with the following:

```
10000 FOR I = 49408 TO 49973 : READ X : POKE I,X : NEXT
      : RETURN
```

These lines (including the data statements) must now be included in any program that you wish to use with this sort routine.

Listing 3.

Listing 3 is a program that demonstrates the features of the sort routine. It should give you a number of ideas as to how you can incorporate these routines in your own programs.

Cassette users will have to replace line 100 with

```
100 IF PEEK(49500)<>76 THEN GOSUB 10000
```

and you will also have to include the routine from Listing 2 to the end of this program.

When you run this program you will be presented with a list of names (last name and first name) together with some mark assigned to each person in that list. Press the space bar and that list will be sorted by last name. Look at it carefully and notice that when the last names are the same, the first names are not necessarily in order. Press the space bar again and the list is re-sorted, this time according to last name and by first name as well whenever a match occurs. Press the space bar again and the list is sorted into descending order according to marks. Look carefully again and compare the names of those people that have the same marks. Continue on with the demo and you will see that the sort routine is in fact quite versatile.

Listing 4.

How fast is it really? The program in Listing 3 sorts the arrays virtually instantaneously. However, there are only seventeen elements in the array to be sorted, and even a BASIC program would take only a few seconds to sort them. How fast is the machine language sort when there are a large number of array elements? Well this program will allow you to test it. As it is presently set up, it will generate 100 records of 10 random characters each. It will display each record as it is created. When asked to press the space bar, do so. The program will then sort the randomly created records and print them out in sorted order for you to view.

For 100 records it takes only a fraction of a second to complete the sort. You can test any number of records simply by changing the value of NUM in line 130. Change it to 2000 to see how long it takes to sort 2000 records. When you change NUM to a large value, you will see the effect that a 'garbage collection' has. During the creation of the records, the records will be produced quite steadily at first. Then suddenly, the process appears to stop for a while. This is a 'garbage collection'. With NUM=2000, it will stop like this several times, for several minutes each time. During a BASIC sort, garbage collection like this will occur even more frequently.

By the way, do not use the 64's built-in time clock to time the sort – use a stop watch instead. The sort routine disables the interrupts and hence turns off the clock while the sort is in progress.

Listing 5.

This is the same as Listing 2 adapted for BASIC 4.0 PET/CBMs. A number of changes had to be made, most of them quite simple (e.g. redefine the variables used in zero page). The only major change was to adjust for the way strings are stored in BASIC 4.0 (a backward pointer is stored with any string defined in high memory – this enables garbage collections in BASIC 4.0 to be practically instantaneous). With these changes, the routine works perfectly well on the PET. In fact, it can be combined with some routines published a couple of years ago (i.e. The BMB String Thing and Glen Pearce's Keyed Random Access) to form a very powerful database tool.

The demonstration programs in Listing 3 and 4 will work on the PET with one minor modification. Namely, replace lines 100 and 110 with the following:

```
100 IF PEEK(31600)<>230 THEN LOAD "SORT PET",8
110 POKE 53,123 : CLR : SRT=7*4096+11*256
```

These lines load the PET version of the sort routine into memory and then lower the top of memory to protect it from destruction.

I hope you find this sort routine both informative and useful. It was certainly a lot of fun creating it. If you have understood everything I have said, here are a few questions to think about. Many answers are possible.

1. Suppose you have an array SN of positive integers (e.g. serial numbers for a parts list). How could you use this routine to get a printout of the serial numbers in numerical order?
2. How would you use this routine if you wanted to sort an array N$, but you also wanted to preserve its original order?
3. Suppose you are a teacher and have an array NA$ of student names and another array MA of corresponding marks (e.g. average marks for a test). How would you use this routine to rank the students according to mark?

Listing 1 PAL Source Code for Commodore 64s

```
 100 open4,4
1000 sys 700
1010 .opt p4
1020 linnum  = $14
1030 index1  = $22
1040 arytab  = $2f
1050 strend  = $31
1060 varnam  = $45
1070 lowtr   = $5f
1080 chrget  = $73
1090 zp      = $a3      ;beginning of zero page work area
1100 ptrna0  = $a3      ;pntr to descriptor for na$(0)
1110 l       = $a5      ;starting subscript for sort
1120 h       = $a7      ;ending subscript for sort
1130 num     = $a9      ;# of elements to be sorted
1140 m       = $ab      ;starting column for sort
1150 n       = $ac      ;ending column for sort
1160 a1      = $ad      ;flag for ascend/descend sort
1170 last    = $ae      ;last column for comparison
1180 z       = $af
1190 i       = $b1
1200 j       = $b3
1210 k       = $b5
1220 ptrnaj  = $b7      ;pntr to descriptor for na$(j)
1230 ptrnak  = $b9      ;pntr to descriptor for na$(k)
1240 najlen  = $bb      ;length of na$(j)
1250 naj     = $bc      ;pointer to na$(j)
1260 naklen  = $be      ;length of na$(k)
1270 nak     = $bf      ;pointer to na$(k)
1280 fld     = $fd      ;current field number
1290 numfld  = $fe      ;# of fields to be sorted - 1
1300 errprt  = $a447
1310 synerr  = $af08
1320 chkstr  = $ad8f
1330 frmevl  = $ad9e
1340 chkcom  = $aefd
1350 combyt  = $b7f1;   check for comma and get a byte
1360 getadr  = $b7f7
1370 tempz   = $c000    ;temporary area for zero page
1380 tptna0  = $c000    ;temporary location for ptrna0
1390 tl      = $c002    ;temporary location for l
1400 th      = $c004    ;temporary location for h
1410 tempm   = $c020
1420 tempn   = $c040
1430 tempa   = $c060
1440 *       = $c100
1450 ;
1460 ;get parameters
1470 ;
1480        jsr  chkcom
1490        jsr  getary
1500        lda  lowtr
1510        clc
1520        adc  #$07
1530        sta  tptna0
1540        lda  lowtr+1
1550        adc  #$00
1560        sta  tptna0+1
1570        jsr  chkcom
1580        jsr  frmevl
1590        jsr  getadr
1600        lda  linnum
1610        sta  tl         ;get starting subscript
1620        lda  linnum+1
1630        sta  tl+1
1640        jsr  chkcom
1650        jsr  frmevl
1660        jsr  getadr
1670        lda  linnum
1680        sta  th         ;get ending subscript
```

```
1690          lda  linnum+1
1700          sta  th+1
1710          lda  #$00
1720          sta  numfld
1730 getmor   jsr  combyt
1740          txa
1750          ldy  numfld
1760          sta  tempm,y    ;get starting column
1770          jsr  combyt     ;of field
1780          txa
1790          ldy  numfld
1800          sta  tempn,y    ;get ending column
1810          jsr  chkcom     ;of field
1820          cmp #"a"        ;"ascending sort
1830          beq  ascend     ;on this field?
1840          cmp #"d"        ;"descending sort
1850          beq  dscend     ;on this field?
1860          jmp  synerr
1870 ascend   lda  #$01
1880 .byte $2c
1890 dscend   lda  #$00
1900          ldy  numfld
1910          sta  tempa,y
1920 ;
1930 ;any more parameters
1940 ;
1950          jsr  chrget
1960          cmp #",",       ;if comma present then
1970          bne  srt        ;get more parameters
1980          inc  numfld
1990          bne  getmor
2000 srt      sei
2010          ldx  #$1e
2020          jsr  swap1
2030          sec
2040          lda  h
2050          sbc  l
2060          sta  num        ;calculate number of
2070          lda  h+1        ;elements to be sorted
2080          sbc  l+1
2090          sta  num+1
2100          inc  num
2110          bne  xeqs1
2120          inc  num+1
2130 ;
2140 ;find starting value for z
2150 ;
2160 xeqs1    ldx  #$01
2170 starta   lda  zval,x
2180          cmp num
2190          lda  zval1,x
2200          sbc  num+1
2210          bcs  begin
2220 ;
2230 ;try next value
2240 ;
2250          inx
2260          bne  starta
2270 ;
2280 ;are we finished yet
2290 ;
2300          begindex
2310          bne  loop
2320          ldx  #$1e
2330          jsr  swap1
2340          cli
2350          rts
2360 ;
2370 ;z=(z-1)/3
2380 ;
2390 loop     lda  zval,x
2400          sta  z
2410          lda  zval1,x
2420          sta  z+1
2430 ;
2440 ;i=z+l
2450 ;
2460 clc
2470          lda  z
2480          adc  l
2490          sta  i
2500          lda  z+1
2510          adc  l+1
2520          sta  i+1
2530 ;
2540 ;j=i-z
2550 ;
2560 loop2    sec
2570          lda  i
2580          sbc  z
2590          sta  j
2600          lda  i+1
2610          sbc  z+1
2620          sta  j+1
2630 ;
2640 ;k=j+z
2650 ;
2660 getk     clc
2670          lda  j
2680          adc  z
2690          sta  k
2700          lda  j+1
2710          adc  z+1
2720          sta  k+1
2730 ;
2740 ;compare and swap if necessary
2750 ;
2760          jsr  compar
2770          bne  nexti
2780          jsr  swap
2790 ;
2800 ;j=j-z
2810 ;
2820          sec
2830          lda  j
2840          sbc  z
2850          sta  j
2860          lda  j+1
2870          sbc  z+1
2880          sta  j+1
2890          bcc  nexti
2900 ;
2910 ;if j>=l then compare again
2920 ;
2930          lda  j
2940          cmp l
2950          lda  j+1
2960          sbc  l+1
2970          bcs  getk
2980 ;
2990 ;i=i+1
3000 ;
3010 nexti    inc  i
3020          bne  nexti2
3030          inc  i+1
3040 ;
3050 ;if i>h then begin again
3060 ;with a new value for z
3070 ;
3080 nexti2   lda  h
3090          cmp i
3100          lda  h+1
3110          sbc  i+1
3120          bcs  loop2
3130          bcc  begin
3140 ;
3150 ;compare na$(j) with na$(k)
3160 ;
3170 ;on exit  if a=1 then no swap is required
3180 ;         if a=0 then a swap is required
3190 ;
3200 compartxa
3210          pha
3220          ldx  #$00
3230 comp2    lda  j,x
3240          sta  zval
3250          lda  j+1,x
3260          sta  zval1
3270          asl  zval
3280          rol  zval1
3290          clc
3300          lda  j,x
3310          adc  zval
3320          sta  zval
3330          lda  j+1,x
3340          adc  zval1
3350          sta  zval1
3360          clc
3370          lda  zval
3380          adc  ptrna0     ;calculate location of
3390          sta  ptrnaj,x   ;descriptor for na$(j)
3400          lda  zval1      ;and na$(k) and store them
3410          adc  ptrna0+1
3420          sta  ptrnaj+1,x
3430          inx
3440          inx
3450          cpx  #$04
3460          bne  comp2
3470          ldy  #$02
3480 point    lda  (ptrnaj),y
3490          sta  najlen,y   ;store actual descriptors
3500          lda  (ptrnak),y
3510          sta  naklen,y
3520          dey
3530          bpl  point
3540          pla
3550          tax
3560 ;
3570 ;now do the comparison
3580 ;
3590          lda  #$ff
3600          sta  fld
3610 nxtfld   inc  fld
3620          ldy  fld
3630          cpy  numfld
3640          beq  cont
3650          bcs  which
3660 cont     lda  tempm,y
3670          sta  m
3680          lda  tempn,y
3690          sta  n
3700          lda  tempa,y
3710          sta  a1
3720          lda  najlen
3730          cmp m
3740          bcc  which
3750          sta  last
3760          lda  naklen
3770          cmp m
3780          bcc  greatr
3790          cmp last
3800          bcs  tryn
3810          sta  last
3820 tryn     lda  n
3830          cmp last
3840          bcs  docomp
3850          sta  last
3860 docompldy  m
3870          dey
3880 hereb    lda  (naj),y
3890          cmp (nak),y
3900          beq  incry
3910          bcs  greatr
3920 less     lda  #$00
3930          beq  there2
3940 incry    iny
3950          cpy  last
3960          bne  hereb
3970          beq  nxtfld
3980 which    lda  najlen
3990          cmp naklen
4000          bcc  less
4010 greatr   lda  #$01
4020 there2   eor  a1
4030          rts
4040 ;
4050 ;swap descriptors
4060 ;
4070 swap     ldy  #$02
4080 swapz    lda  najlen,y
4090          sta  (ptrnak),y
4100          lda  naklen,y
4110          sta  (ptrnaj),y
4120          dey
4130          bpl  swapz
4140          rts
4150 zval .byte 0,1,4,13,40,121,108,69,208,113,84,0
4160 zval1 .byte 0,0,0,0,0,0,1,4,12,38,115,128
4170 ;
4180 ;find array header
4190 ;
4200 getary   jsr  frmevl     ;get array name in varnam
4210          jsr  chkstr
4220          lda  arytab
4230          sta  lowtr
4240          lda  arytab+1
4250          sta  lowtr+1
4260 nxtnamldy  #$00          ;"array name found?
4270          lda  (lowtr),y
4280          cmp varnam
4290          bne  nxtptr     ;no
4300          iny             ;maybe, check next character
4310          lda  (lowtr),y
4320          cmp varnam+1
4330          bne  nxtptr     ;no
4340          ldy  #$04
4350          lda  (lowtr),y
4360          cmp #$01        ;is it a 1 dimensional array
4370          bne  nxtptr     ;no! check for more
4380          rts             ;yes
4390 nxtptr   ldy  #$02       ;calculate pointer to next
4400          clc             ;array name
4410          lda  (lowtr),y
4420          adc  lowtr
4430          pha
4440          iny
4450          lda  (lowtr),y
4460          adc  lowtr+1
4470          sta  lowtr+1
4480          pla
4490          sta  lowtr
4500          cmp strend      ;"any more array variables?
4510          lda  lowtr+1
4520          sbc  strend+1
4530          bcc  nxtnam     ;yes
4540          lda  #$a5       ;print "array error?"
4550          sta  index1
4560          lda  #$a2
4570          sta  index1+1
4580          jmp  errprt
4590 ;
4600 ;swap part of zero page
4610 ;
4620 swapzpldx  #$08
4630 swap1    lda  zp,x
4640          pha
4650          lda  tempz,x
4660          sta  zp,x
4670          pla
4680          sta  tempz,x
4690          dex
4700          bpl  swap1
4710          rts
```

Listing 2 for Commodore 64s

```
10000 open1,8,1," sort 64 ":print#1,chr$(0);:print#1,chr$(193);
10005 for i=49408 to 49973:read x:print#1,chr$(x);:next:close1:end
10010 data  32, 253, 174,  32, 217, 194, 165,  95
10011 data  24, 105,   7, 141,   0, 192, 165,  96
10012 data 105,   0, 141,   1, 192,  32, 253, 174
10013 data  32, 158, 173,  32, 247, 183, 165,  20
10014 data 141,   2, 192, 165,  21, 141,   3, 192
10015 data  32, 253, 174,  32, 158, 173,  32, 247
10016 data 183, 165,  20, 141,   4, 192, 165,  21
10017 data 141,   5, 192, 169,   0, 133, 254,  32
10018 data 241, 183, 138, 164, 254, 153,  32, 192
10019 data  32, 241, 183, 138, 164, 254, 153,  64
10020 data 192,  32, 253, 174, 201,  65, 240,   7
10021 data 201,  68, 240,   6,  76,   8, 175, 169
10022 data   1,  44, 169,   0, 164, 254, 153,  96
10023 data 192,  32, 115,   0, 201,  44, 208,   4
10024 data 230, 254, 208, 203, 120, 162,  30,  32
10025 data  38, 195,  56, 165, 167, 229, 165, 133
10026 data 169, 165, 168, 229, 166, 133, 170, 230
10027 data 169, 208,   2, 230, 170, 162,   1, 189
10028 data 193, 194, 197, 169, 189, 205, 194, 229
10029 data 170, 176,   3, 232, 208, 241, 202, 208
10030 data   7, 162,  30,  32,  38, 195,  88,  96
10031 data 189, 193, 194, 133, 175, 189, 205, 194
10032 data 133, 176,  24, 165, 175, 101, 165, 133
10033 data 177, 165, 176, 101, 166, 133, 178,  56
10034 data 165, 177, 229, 175, 133, 179, 165, 178
10035 data 229, 176, 133, 180,  24, 165, 179, 101
10036 data 175, 133, 181, 165, 180, 101, 176, 133
10037 data 182,  32,  12, 194, 208,  28,  32, 177
10038 data 194,  56, 165, 179, 229, 175, 133, 179
10039 data 165, 180, 229, 176, 133, 180, 144,  10
10040 data 165, 179, 197, 165, 165, 180, 229, 166
10041 data 176, 210, 230, 177, 208,   2, 230, 178
10042 data 165, 167, 197, 177, 165, 168, 229, 178
10043 data 176, 181, 144, 146, 138,  72, 162,   0
10044 data 181, 179, 141, 193, 194, 181, 180, 141
10045 data 205, 194,  14, 193, 194,  46, 205, 194
10046 data  24, 181, 179, 109, 193, 194, 141, 193
10047 data 194, 181, 180, 109, 205, 194, 141, 205
10048 data 194,  24, 173, 193, 194, 101, 163, 149
10049 data 183, 173, 205, 194, 101, 164, 149, 184
10050 data 232, 232, 224,   4, 208, 202, 160,   2
10051 data 177, 183, 153, 187,   0, 177, 185, 153
10052 data 190,   0, 136,  16, 243, 104, 170, 169
10053 data 255, 133, 253, 230, 253, 164, 253, 196
10054 data 254, 240,   2, 176,  65, 185,  32, 192
10055 data 133, 171, 185,  64, 192, 133, 172, 185
10056 data  96, 192, 133, 173, 165, 187, 197, 171
10057 data 144,  44, 133, 174, 165, 190, 197, 171
10058 data 144,  42, 197, 174, 176,   2, 133, 174
10059 data 165, 172, 197, 174, 176,   2, 133, 174
10060 data 164, 171, 136, 177, 188, 209, 191, 240
10061 data   6, 176,  17, 169,   0, 240,  15, 200
10062 data 196, 174, 208, 239, 240, 181, 165, 187
10063 data 197, 190, 144, 239, 169,   1,  69, 173
10064 data  96, 160,   2, 185, 187,   0, 145, 185
10065 data 185, 190,   0, 145, 183, 136,  16, 243
10066 data  96,  51,   1,   4,  13,  40, 121, 108
10067 data  69, 208, 113,  84,   0,   0,   0,   0
10068 data   0,   0,   0,   1,   4,  12,  38, 115
10069 data 128,  32, 158, 173,  32, 143, 173, 165
10070 data  47, 133,  95, 165,  48, 133,  96, 160
10071 data   0, 177,  95, 197,  69, 208,  16, 200
10072 data 177,  95, 197,  70, 208,   9, 160,   4
10073 data 177,  95, 201,   1, 208,   1,  96, 160
10074 data   2,  24, 177,  95, 101,  95,  72, 200
10075 data 177,  95, 101,  96, 133,  96, 104, 133
10076 data  95, 197,  49, 165,  96, 229,  50, 144
10077 data 206, 169, 165, 133,  34, 169, 162, 133
10078 data  35,  76,  71, 164, 162,   8, 181, 163
10079 data  72, 189,   0, 192, 149, 163, 104, 157
10080 data   0, 192, 202,  16, 241,  96
```

Listing 3 for Commodore 64s

```
100 if peek(49500)<>76 then load " sort 64 ",8,1
110 srt=12*4096+256
120 bl$=" ":n=17
130 dim a$(n)
140 for i=1 to n
150 read a$,b$,c$
160 a$(i)= left$(a$+bl$,15)+left$(b$+bl$,14)
      +right$(bl$+c$,3)
170 next i
180 print " Sq "tab(15)" r unsorted q "
190 gosub 380
200 gosub 400
210 sys srt,a$,1,n,1,15,a
220 print " Sq "tab(10)" r sorted by last name q "
230 gosub 380
240 gosub 400
250 sys srt,a$,1,n,1,15,a,16,29,a
260 print " Sqr sorted by last name and first name q "
270 gosub 380
280 gosub 400
290 sys srt,a$,1,n,30,32,d
300 print " Sq "tab(12)" r sorted by marks q "
310 gosub 380
320 gosub 400
330 sys srt,a$,1,n,30,32,d,1,15,a,16,29,a
340 print " Sq "tab(7)" r sorted by marks and name q "
350 gosub 380
360 gosub 400
370 print " S ":end
380 print "   r last name     first name   marks q "
390 for i=1 to n:print tab(4);a$(i):next:return
400 print " sqqqqqqqqqqqqqqqqqqqqqqqq ";tab(11);
      " r press space bar "
410 get a$:if a$<>" " then 410
420 return
1000 data smith, bob, 75
1001 data jones, bill, 66
1002 data miller, barney, 85
1003 data smith, barb, 88
1004 data smithers, jon, 56
1005 data miller, barbara, 85
1006 data smith, gerry, 75
1007 data jones, jim, 88
1008 data smiley, robert, 50
1009 data atkinson, william, 99
1010 data baker, don, 64
1011 data carson, johnny, 44
1012 data baker, carol, 100
1013 data atkins, chet, 75
1014 data white, ray, 51
1015 data walker, toby, 51
1016 data walker, willy, 91
```

Listing 3 for PET/CBMs

Change lines 100 and 110 of the previous listing to:

```
100 if peek(31600)<>230 then load " sort pet ",8
110 poke 53,123:clr:srt=7*4096+11*256
```

Listing 4 for Commodore 64s

```
100 if peek(49500)<>76 then load "sort 64",8,1
110 srt=12*4096+256
120 dim a$(1000),b$(1000)
130 num=100
140 print "Sqqqqqqqqq generating ";num;
    " random words"
150 for i=1 to num
160 b$=""
170 for j=1 to 10
180 b$=b$+chr$(rnd(1)*26+65)
190 next j
200 a$(i)=b$:b$(i)=b$
210 print "sqq  ";tab(10);i;tab(17)b$
220 next i
230 print "Sq   press space to begin sort"
240 get a$:if a$<>" " then 240
250 print "q sorting!"
260 sys srt,b$,1,num,1,10,a
270 print "q done q"
280 for i=1 to num
290 print tab(5);a$(i);tab(25);b$(i)
300 next
```

Listing 4 for PET/CBMs

Change lines 100 and 110 of the previous listing to:

```
100 if peek(31600)<>230 then load "sort pet",8
110 poke 53,123:clr:srt=7*4096+11*256
```

Listing 5 for PET/CBMs

```
10000 open1, 8, 1, "sort pet":print#1,chr$(0);
      :print#1,chr$(123);
10005 for i=31488 to 32085:read x:print#1,chr$(x);:next
      :close1:end
10010 data  32, 245, 190,  32, 254, 124, 165,  92
10011 data  24, 105,   7, 141,   0, 126, 165,  93
10012 data 105,   0, 141,   1, 126,  32, 245, 190
10013 data  32, 152, 189,  32,  45, 201, 165,  17
10014 data 141,   2, 126, 165,  18, 141,   3, 126
10015 data  32, 245, 190,  32, 152, 189,  32,  45
10016 data 201, 165,  17, 141,   4, 126, 165,  18
10017 data 141,   5, 126, 169,   0, 133, 254,  32
10018 data  39, 201, 138, 164, 254, 153,  32, 126
10019 data  32,  39, 201, 138, 164, 254, 153,  64
10020 data 126,  32, 245, 190, 201,  65, 240,   7
10021 data 201,  68, 240,   6,  76,   0, 191, 169
10022 data   1,  44, 169,   0, 164, 254, 153,  96
10023 data 126,  32, 112,   0, 201,  44, 208,   4
10024 data 230, 254, 208, 203,  32,  67, 125,  56
10025 data 165, 212, 229, 210, 133, 214, 165, 213
10026 data 229, 211, 133, 215, 230, 214, 208,   2
10027 data 230, 215, 162,   1, 189, 230, 124, 197
10028 data 214, 189, 242, 124, 229, 215, 176,   3
10029 data 232, 208, 241, 202, 208,   5,  32,  67
10030 data 125,  88,  96, 189, 230, 124, 133, 220
10031 data 189, 242, 124, 133, 221,  24, 165, 220
10032 data 101, 210, 133, 222, 165, 221, 101, 211
10033 data 133, 223,  56, 165, 222, 229, 220, 133
10034 data 224, 165, 223, 229, 221, 133, 225,  24
10035 data 165, 224, 101, 220, 133, 226, 165, 225
10036 data 101, 221, 133, 227,  32,   7, 124, 208
10037 data  28,  32, 172, 124,  56, 165, 224, 229
10038 data 220, 133, 224, 165, 225, 229, 221, 133
10039 data 225, 144,  10, 165, 224, 197, 210, 165
10040 data 225, 229, 211, 176, 210, 230, 222, 208
10041 data   2, 230, 223, 165, 212, 197, 222, 165
10042 data 213, 229, 223, 176, 181, 144, 148, 138
10043 data  72, 162,   0, 181, 224, 141, 230, 124
10044 data 181, 225, 141, 242, 124,  14, 230, 124
10045 data  46, 242, 124,  24, 181, 224, 109, 230
10046 data 124, 141, 230, 124, 181, 225, 109, 242
10047 data 124, 141, 242, 124,  24, 173, 230, 124
10048 data 101, 208, 149, 228, 173, 242, 124, 101
10049 data 209, 149, 229, 232, 232, 224,   4, 208
10050 data 202, 160,   2, 177, 228, 153, 232,   0
10051 data 177, 230, 153, 235,   0, 136,  16, 243
10052 data 104, 170, 169, 255, 133, 253, 230, 253
10053 data 164, 253, 196, 254, 240,   2, 176,  65
10054 data 185,  32, 126, 133, 216, 185,  64, 126
10055 data 133, 217, 185,  96, 126, 133, 218, 165
10056 data 232, 197, 216, 144,  44, 133, 219, 165
10057 data 235, 197, 216, 144,  42, 197, 219, 176
10058 data   2, 133, 219, 165, 217, 197, 219, 176
10059 data   2, 133, 219, 164, 216, 136, 177, 233
10060 data 209, 236, 240,   6, 176,  17, 169,   0
10061 data 240,  15, 200, 196, 219, 208, 239, 240
10062 data 181, 165, 232, 197, 235, 144, 239, 169
10063 data   1,  69, 218,  96, 160,   2, 185, 232
10064 data   0, 145, 230, 185, 235,   0, 145, 228
10065 data 136,  16, 243, 165, 233, 197,  42, 165
10066 data 234, 229,  43, 144,  11, 164, 232, 165
10067 data 230, 145, 233, 200, 165, 231, 145, 233
10068 data 165, 236, 197,  42, 165, 237, 229,  43
10069 data 144,  11, 164, 235, 165, 228, 145, 236
10070 data 200, 165, 229, 145, 236,  96,   0,   1
10071 data   4,  13,  40, 121, 108,  69, 208, 113
10072 data  84,   0,   0,   0,   0,   0,   0,   0
10073 data   1,   4,  12,  38, 115, 128,  32, 152
10074 data 189,  32, 137, 189, 165,  44, 133,  92
10075 data 165,  45, 133,  93, 160,   0, 177,  92
10076 data 197,  66, 208,  16, 200, 177,  92, 197
10077 data  67, 208,   9, 160,   4, 177,  92, 201
10078 data   1, 208,   1,  96, 160,   2,  24, 177
10079 data  92, 101,  92,  72, 200, 177,  92, 101
10080 data  93, 133,  93, 104, 133,  92, 197,  46
10081 data 165,  93, 229,  47, 144, 206, 162, 128
10082 data  76, 207, 179, 120, 162,  30, 189,   0
10083 data 126,  72, 181, 208, 157,   0, 126, 104
10084 data 149, 208, 202,  16, 241,  96
```

Phile Master

Robert Drake
Brantford, Ont.

I try to keep up with what's available in software for the C64. Recently I saw an ad for a simple, easy to use filing system. And then I saw the price. A little hunting found at least 3 other advertisements at 3 other prices. The prices bothered me because I wrote a simple file program with a computer science class a couple of years ago. That program, called Phono Phile, demonstrated simple files, arrays, menus, printing, search and sort routines. All this for free.

Phono Phile has been resurrected, improved, made more powerful and is free for the typing. Phile Master creates simple files to your specifications easily. The program is about 7.5K long. There are only two limitations. Your file must fit into the rest of memory. You have about 30K for your data. Secondly, no field can exceed 80 characters since all data is handled by INPUT, INPUT#, and PRINT#.

What can you do with it? You can use it for recipes, to take an inventory, to create labels for library books. Keep track of your stamp collection or photos or the membership of your favorite club. Making a bibliography of magazine articles is easy but for the typing.

Phile Master is written with only one statement per program line where possible. The code is fairly 'clean' and easy to read. Everything is in BASIC for easy modification. Learn from the program. Enterprising programmers may want to lift a few subroutines for use elsewhere. The screen colour rotation (lines 100–124) is one you might want for other programs. Custom printer routines are easy to write using the given routine as an example. Most array handling is done using FOR–NEXT loops. You can cut the program size considerably by concatenating lines.

Let's get the terminology down. Assume we're putting together a phone list/address book. For each person (record) we will want the name (a field), the street address (another field), the city (field), province (field), area code (field) and phone number (field). So each record has 6 fields. If we set this up for 150 people (records) then we have a file. If a program can use several files without your notice then you have a data base.

Phile Master lets you enter the number of records you want, the number of fields per record, and whatever you want to describe the fields.

Let's work through an example and see what happens and what you can do. Run the program and the first thing to do is to adjust the screen, border and text colours for your system using the function keys F1, F3 and F5. Exit via F7.

You will be asked for a file name. Remember a file is the collection of records. We don't have a file yet so just press the return key. You are going to create a file to your definition. Type in a name for the file. Your disk drive will come on as the program checks for that name. If you already have a directory entry with that name an error will be caught by the program. Next enter the MAXIMUM number of records (people) on whom you will want to keep a record (say 150). Then enter the number of fields (7). A little preplanning is helpful here. Lay your application out on paper first. Changing your mind after this step is going to be VERY difficult. For each field type in a description. (Try GIVEN NAME, SURNAME, ADDRESS, CITY, PROVINCE, AREA CODE, and PHONE.) The disk will whirr and you will probably have to wait a few seconds while arrays are dimensioned for the file you have requested. This section of program actually writes 2 files. One is prefixed with "PM–" and it holds your file description. The other is a dummy file holding only a terminal record (one that marks the end of a load or save).

You are now facing one of 2 menus which control the program. Both run off the function keys. Let's take these one at a time in some kind of a logical (?) order. Press F1:ADD. You will see the field description and an INPUT question mark. The quotes will let you enter colons and commas as part of your entry. Enter a name. Enter some information under the other fields. Just press return if you don't want anything entered. When the record is complete the program comes back and asks "IS THAT INFORMATION CORRECT?" Just for the heck of it press N for no. See where the cursor goes? If a field is correct press return; if wrong move the cursor and correct your errors. When everything is right then answer the question Y. Enter a half a dozen names and addresses in this way so you have a sample with which to work. By the way, I originally had upper and lower case in the program as the only option. I changed my mind. Use the shift/Commodore key to display lower case if you want capitals and small letters.

Pressing F2 lets you save your file on disk. Pressing 'A' now lets you get out of this option. Most of the choices have an exit. If the error could be fatal to your file or intentions the option is given immediately. Otherwise it is given eventually. The save function executes by erasing (scratching) the old file and saving the new one. There are both good and bad aspects to this. A file can be cleared (emptied) by doing a save without loading or entering new information. Disk errors will be reported to you. If all is fine the disk will whirr and in a few seconds the main menu will reappear. Press a key and save your file.

F5 lets you load your file. Normally loading your file would be one of the first things you would do. This load is kind of neat. Each load adds the disk file to whatever you have in memory. This avoids the problem of a load destroying records you have painfully entered from the keyboard. Press F5 to see how it works. Notice you again have the abort option. Load the file you just saved. See how a visual load is accomplished. I like to see things happen when the computer goes away by itself to do something and nothing seems to be happening.

Press F3 to get to the search routine. All your field descriptions are given and lettered (in our example) from A to G. Pressing * gets you back to the menu. Pick one of the letters and you will be asked what you want to find. This is a "wild card" search. Entering a single letter like "S" will find ALL entries in that field starting with

S. A more restrictive entry (" SMIT ") will find all entries beginning with SMIT (such as SMITH, SMITHE, SMITHERS, and so on). When the last entry is found the program returns to the menu. I found searching to the last item in a 95 record list required less than 2 seconds.

F4 does the printing. Since this is probably the one section you will want to customize it is the last routine in the program. That makes it easy to remove and replace. Press F4 to jump into a simple report writer. The title is optional. If you don't want one, press return. Otherwise just type it in. The fields are listed and lettered as with the search routine. Enter how many you want to print. Then enter which field (by letter) and the number of columns to allow. The fields can be printed in any order you want. Watch the number of columns you're printing. Your printer is assumed to have 80 columns.

Do you want headers? What's a header you ask? They are the field descriptions printed at the top of the page. If your column widths are too narrow, you will find your field descriptions are cut back to that width.

Which record to start at and end at are the next questions. The records are in the order of the last sort you did on them. (More about sorting them later.)

The next to last question: Do you really want the printer or do you want a trial run to the screen? I find a screen dump can save a lot of paper. Finally you can abort. Pressing return while the program is printing your report will create a pause until another key is pressed. The report columns are left justified, the same as you get with a typewriter. You might want to add a few lines to output upper and lower case to your printer.

Well, only F6 and F7 are left. Let's take the easy one first. Press F6. You are now out of the program. I always find it annoying to accidently leave a program and not have a way back in without losing all my work. That's why you have to shift to exit and why the cursor is sitting on the command to reenter the program safely with ALL your information intact. Pressing return now will get you back to the main menu with no damage at all. If you should exit the program via RUN/STOP then you can usually get back in with CONT. If you get a ?CAN'T CONTINUE ERROR then try GOTO 500. Don't OPEN 1,8,15 unless you have closed the disk file down.

Last but not least is F7. Pressing F7 gives you the other menu. This menu is also operated with the function keys. Press F1. Quick press a key, any key! As with the printer routine there is a pause built in to let you stop and examine the listing. Pressing F2 also lets you examine the file but one record at a time. The records start at zero and go to one less than the number stated. Forty six records go from 0 to 45. Blame this on me. I like to count from zero. You don't have to shift to use the > and < keys to go back and forth. 'R' will let you access any record immediately.

F5 sorts on any field. Press this and you get a printer type field list. Choose your field and it is automatically sorted from least to greatest or alphabetically. Depending on the size of your file, give it four or five seconds to get going. You will see the sorted fields listed on your screen. A word of warning. I cheat a little here and really don't sort the records but just their pointers. To make the sort permanent, save the file to disk. The sort isn't the fastest in the world but it isn't bad.

F7 lets you change a record. You are asked to confirm your choice or abort. Change the record and notice it uses the same technique (overlaying the cursor) as Adding Information. This is to minimize typing on your part.

F6 does exactly what it says it does. F6 deletes records. You must confirm your choice of record. Both F6 and F7 mess up your sort since both change the file. F8 returns you to the menu.

If you want to use Phile Master on the PET some changes are needed. First, delete lines 102–199. After that the main changes are those to replace the function keys and to handle null inputs which can be fatal on the PET. Here's a list of the needed changes.

```
206 print " Sr phile master R " : print " enter your file name.
    press r return R if "
208 input " you do not have a file. *[3left] " ;fi$
210 if fi$ = " * " then goto 300
304 input " q enter a name for the file *[3left] " ;fi$
306 if fi$ = " * " then goto 304
504 print " Sr phile R 1:add 2:save 3:search 4:print "
506 print " r master R 5:load 6:juggle 7:uc/lc 8:end "
510 k = val(key$)
512 if k<1 or k>8 then goto 508
514 if k = 8 then goto 520
516 on k gosub 600,800,700,2100,900,1000,528
528 rem ****upper case/lower case
530 c = peek(59468)
532 if c = 14 then poke 59468,12
534 if c = 12 then poke 59468,14
536 return
720 poke 623,34 : poke 158,1
1004 print " Sr juggle R 1:list all 2:list one 3:sort "
1006 print " r records R 4:change 5:delete 6:menu "
1010 m = val(key$)
1012 if m = 6 then goto 1422
1014 if m<>2 then goto 1100
1040 if key$ = " > " then k = k + 1
1042 if key$ = " < " then k = k-1
1102 if m<>1 then goto 1200
1202 if m<>5 then goto 1300
1302 if m <>4 then goto 1400
1402 if m <>3 then goto 1000
1810 poke 623,34 : poke 158,1
2106 poke 623,34 : poke 158,1
```

Phile Master can also be tape based. This will make the save and load routines slower but they will work. Make the following changes.

Delete lines 204, 216, 218, 220, 222,224, 226, 228, 230, 312, 314, 316, 318, 320, 348, 350, 352, 354, 356, 358, 826, 828, 830, 832, 916, 918, 920, 922.

Change and/or add the following lines.

```
214 open 2,1,0, " pm- " + fi$
310 open2,1,1, " pm- " + fi$
802 print " sqqr record information on tape
804 print " q place your data tape in the recorder.
805 print " rewind the tape fully and press 'stop' " .
814 open2,1,1, " pm- " + fi$
815 print " saving file description "
```

```
816 print#2, n$ cr$ nf$ cr$;
818 for i = 1 to nf
820 print#2, fd$(i) cr$
822 next i
823 close 2
824 open 2,1,1,fi$
825 print "saving data";
841 print ".";
904 print "qplace your data tape in the recorder."
914 open2,1,0,fi$
```

Well, that's it. Phile Master gives you a simple yet powerful tool for your computer. If you come up with some modifications or file templates that may be useful to others, let me know so we can share them. Have fun!

(If you don't want to type the program in, send a money order for $10 to R.D.M.C., 8 Centennial Dr., Brantford, Ont., N3R 5X6, and I'll send you a disk or tape with Phile Master for both the C64 and the PET.)

```
100 rem "phile master.64 - r.drake (c)1983"
102 rem ****adjust screen colour-r.drake 1983
104 v(1) = 0 : v(2) = 0 : v(3) = 1
106 poke 53280,v(1) : poke 53281,v(2) : poke646,v(3)
108 print chr$(142) "Sr phile master R
    adjust screen colours"
110 print"f1:border f3:screen f5:type f7:exit
112 get key$
114 k = asc(key$ + chr$(0))-132
116 if k<1 or k>4 then 112
118 if k = 4 then 124
120 v(k) = v(k) + 1 : if v(k) = 16 then v(k) = 0
122 goto 106
124 clr
200 rem ****set up files
202 cr$ = chr$(13)
204 open 1,8,15
206 print"S enter your file name. press r return R "
208 input"if you do not have a file.";fi$
210 if fi$ = "." then goto 300
212 if len(fi$)>13 then fi$ = left$(fi$,13)
214 open 2,8,10,"0:pm-" + fi$ + ",s,r"
216 input#1,a$,b$
218 if a$ = "00" then goto 232
220 print "qq disk error:R " b$
222 close2
224 fi$ = ""
226 for i = 1 to 1000
228 next i
230 goto 206
232 input#2,n$,nf$
234 nf = val(nf$)
236 dim fd$(nf)
238 for i = 1 to val(nf$)
240 input#2,fd$(i)
242 next i
244 close 2
246 goto 400
300 rem ****no file - so create one
302 print"Sr create a file"
304 input"q enter a name for the file";fi$
306 if fi$ = "" then goto 304
308 if len(fi$)>13 then fi$ = left$(fi$,13)
310 open2,8,14,"0:pm-" + fi$ + ",s,w"
312 input#1,a$,b$
314 if a$ = "00" then 322
316 print "qq disk error:R " b$
318 close2
320 goto 304
322 print"q enter the maximum number of"
324 input"records";n$
326 print#2,n$ cr$;
328 input"q how many fields per record";nf$
330 nf = val(nf$)
332 dim fd$(nf)
334 print#2,nf$ cr$;
336 for i = 1 to nf
338 print "description for field" i;
340 input fd$(i)
342 print#2,fd$(i) cr$;
344 next i
346 close 2
348 rem ****create a null file
350 open2,8,10,"0:" + fi$ + ",s,w"
352 for i = 1 to nf
354 print#2,"eod" cr$;
356 next i
358 close 2
400 rem ****initialization
402 n = val(n$)
404 dim rec$(n,nf),r%(n),s%(2*n),w(nf),f(nf),fw(nf)
406 for i = 0 to n
408 r%(i) = i
410 next i
412 l = len(fd$(1))
414 fw(1) = l
416 for j = 2 to nf
418 fw(j) = len(fd$(j))
420 if l< fw(j) then l = fw(j)
422 next j
424 bl$ = "                                "
426 n = -1 :rem ****current # of entries
500 rem **** menu
502 print"S"
504 print"sr phile R f1:add f2:save f3:search f4:print"
506 print"r master R f5:load f6:end f7:list/sort/change"
508 gosub 1500
510 k = asc(key$)
512 if k<133 or k>139 then goto 508
514 if k = 139 then goto 520
516 on k-132 gosub 600,700,900,1000,800,2100
518 goto 502
520 close 1
522 print"S end:press r return R to re-enter"
524 print "qq open1,8,15 : goto 500 s"
526 end
600 rem **** add information
602 n = n + 1
604 print"sqqr add information"
606 gosub 1800
608 if key$ = "y" then goto 614
610 print"sqqq"
612 goto 606
614 print cr$; "do you have further records to add?"
616 gosub 1500
618 if key$ = "y" then print"S" : goto 600
620 return
700 rem **** search
702 print"sqqr search"
704 print"q you may look by:";
706 gosub 1900
708 gosub 1500
710 if key$ = "*" then 738
712 k = asc(key$)-64
714 if k<1 or k> nf then 708
```

```
716 print " S "
718 prompt$ = fd$(k) + " to find "
720 poke 631,34 : poke 198,1
722 gosub 1600
724 for i = 0 to n
726 if a$<>left$(rec$(r%(i),k),len(a$))then 736
728 print " S record # " i
730 gosub 1700
732 print " q press a key to continue. "
734 gosub 1500
736 next i
738 return
800 rem **** record information
802 print " sqqr record information on disk
804 print " q place your data disk in disk drive 0
806 print " press a key when you are ready to
808 print " to continue. press 'a' to abort. "
810 gosub 1500
812 if key$ = " a " then goto 852
814 print#1, " s0: " + fi$
816 input#1,a$,b$
818 if a$ = " 00 " or a$ = " 01 " then goto 824
820 print " r disk error: R " b$
822 stop
824 open 2,8,10, " 0: " + fi$ + " ,s,w "
826 input#1,a$,b$
828 if a$ = " 00 " then goto 834
830 print " r disk error: R " b$
832 stop
834 for j = 0 to n
836 for k = 1 to nf
838 print#2,chr$(34) rec$(r%(j),k) cr$;
840 next k
842 next j
844 for j = 1 to nf
846 print#2, " eod " cr$;
848 next j
850 close 2
852 return
900 rem **** read information
902 print " sqqr read information
904 print " q place your data disk in drive 0. "
906 print " press a key to continue. press 'a' to
908 print " abort. "
910 gosub 1500
912 if key$ = " a " then goto 944
914 open2,8,10, " 0: " + fi$ + " ,s,r "
916 input#1,a$,b$
918 if a$ = " 00 " then goto 924
920 print " r disk error: R " b$
922 stop
924 print " loading " ;
926 n = n + 1
928 for j = 1 to nf
930 input#2,rec$(n,j)
932 next j
934 print " . " ;
936 if rec$(n,1) = " eod " then goto 940
938 goto 926
940 n = n-1
942 close 2
944 return
1000 rem **** manipulate/sort
1002 if n = -1 then return
1004 print " S juggle R f1:list all f3:list one f5:sort "
1006 print " r records R f7:change f6:delete f8:menu "
1008 gosub 1500
1010 m = asc(key$)
1012 if m = 140 then goto 1422
1014 if m<>134 then goto 1100
1016 print " Sqqr list the data file "
1018 print " q there are " ;n + 1; " records. "
1020 input " start at record " ;k
1022 if k> = 0 and k< = n then goto 1028
1024 print " qr illegal record number. please re-enter "
1026 goto 1020
1028 i = r%(k)
1030 print " Sr > R forwards r < R backwards
     r nR estart r m R enu
1032 print
1034 print " record number: " k
1036 gosub 1700
1038 gosub 1500
1040 if key$ = " . " then k = k + 1
1042 if key$ = " , " then k = k-1
1044 if key$ = " r " then goto 1016
1046 if key$ = " m " then goto 1000
1048 goto 1022
1100 rem **** list all items
1102 if m<>133 then goto 1200
1104 for j = 0 to n
1106 print " r " j " R " ;
1108 get a$
1110 if a$ = " " then goto 1116
1112 get a$
1114 if a$ = " " then goto 1112
1116 for k = 1 to nf
1118 print rec$(r%(j),k) ; "  " ;
1120 next k
1122 print
1124 next
1126 gosub 1500
1128 goto 1000
1200 rem **** delete an item
1202 if m<>139 then goto 1300
1204 prompt$ = " item number to delete "
1206 gosub 1600
1208 if val(a$)<0 or val(a$)>n then goto 1324
1210 print " r delete R : "
1212 i = val(a$)
1214 gosub 1700
1216 print " confirm: press r y R or r n R "
1218 gosub 1500
1220 if key$ = " n " then goto 1000
1222 if key$<> " y " then goto 1216
1224 for j = 1 to nf
1226 rec$(r%(i),j) = rec$(r%(n),j)
1228 next j
1230 n = n-1
1232 goto 1000
1300 rem **** change an item
1302 if m <>136 then goto 1400
1304 prompt$ = " change item number "
1306 gosub 1600
1308 if val(a$)<0 or val(a$)>n then goto 1400
1310 i = val(a$)
1312 print " Sqqr item #R " ;i
1314 gosub 1700
1316 print " q is this the correct item?
1318 print " press y for yes; n for no; a to abort.
1320 gosub 1500
1322 if key$ = " a " then goto 1000
1324 if key$ = " y " then goto 1330
1326 if key$ = " n " then goto 1304
1328 goto 1320
1330 for k = 1 to 80 : print chr$(20); : next k
1332 print " sqq hanging record "
1334 flag = 1
```

```
1336 b=n
1338 n=i
1340 gosub 1800
1342 flag=0
1344 n=b
1346 goto 1000
1400 rem **** sort
1402 if m <>135 then goto 1000
1404 print" sqqr sort items"
1406 print " qq     sort on:";
1408 gosub 1900
1410 gosub 1500
1412 if key$="*" then 1422
1414 k=asc(key$)-64
1416 if k<1 or k> nf then 1410
1418 print"wait. . ."
1420 gosub 2000
1422 return
1500 rem **** get a key
1502 get key$
1504 if key$="" then goto 1502
1506 return
1600 rem **** input string
1602 print promt$;
1604 a$=""
1606 input a$
1608 if a$<>'' then goto 1614
1610 print" Q ";:rem cursor up
1612 goto 1600
1614 return
1700 rem *** print fields *** use commas &
     semicolons to get desired effects
1702 for j=1 to nf
1704 print rec$(r%(i),j)
1706 next j
1708 return
1800 rem **** input various fields
1802 for k= 1 to nf
1804 a$=""
1806 print left$(fd$(k)+bl$,l+2);
1808 if flag =1 then print tab(l+3) rec$(r%(n),k) cr$;tab(l);
     " Q ";
1810 poke 631,34 : poke 198,1
1812 input a$
1814 if a$="" then a$="*"
1816 rec$(r%(n),k)=a$
1818 next k
1820 print" qq is that information correct?";
1822 gosub 1500
1824 if key$="y" then 1834
1826 if key$<>"n" then 1822
1828 for i=1 to 30 : print chr$(20); : next
1830 print" sqq "
1832 goto 1802
1834 return
1900 rem ****list fields
1902 for i=1 to nf
1904 print tab(16)" r "chr$(i+64)" R " fd$(i)
1906 next i
1908 if flag=1 then goto 1914
1910 print" q you may also return to the menu (r*R)."
1912 print"press a letter."
1914 return
2000 rem ****tournament sort
2002 m=0 : n=n+1 : x=0 : b=n-1 : for j=0 to b
     : s%(j)=j : nextj
2004 for j=0 to n*2-3 step2
2006 b=b+1 : i1=s%(j) : i2=s%(j+1)
2008 gosub2030
2010 s%(b)=i : nextj
2012 x=x-1 : c=s%(b) : if c<0 goto2034
2014 printrec$(c,k), : r%(m)=c : m=m+1
2016 s%(c)=x
2018 c%=c/2 : j=c%*2 : c=n+c% : if c>b goto2012
2020 i1=s%(j) : i2=s%(j+1)
2022 if i1<0 then i=i2 : goto2028
2024 if i2<0 then i=i1 : goto2028
2026 gosub2030
2028 s%(c)=i : goto2018
2030 i=i1 : if rec$(i2,k)<rec$(i1,k) then i=i2
2032 return
2034 n=n-1
2036 return
2100 rem **** printer output
2102 print" sqqr printer output
2104 t$=""
2106 poke 631,34 : poke 198,1
2108 input" q enter a title";t$
2110 print "the fields are:";
2112 flag=1
2114 gosub 1900
2116 flag=0
2118 input" q print how many fields";f
2120 for i=1 to f
2122 input"field letter";a$
2124 f(i)=asc(a$)-64
2126 if f(i)>nf or f(i)<1 then goto 2122
2128 print" Q " tab(20);
2130 input"# columns";w(i)
2132 next i
2134 input " print headers (y/n)";h$
2136 input"start at record # 0[3left]";b
2138 print"stop at record #  ";n;
2140 input"[5left]";e
2142 if e>n then e=n
2144 input"s)creen or p)rinter";dv$
2146 dv=4
2148 if dv$="s" then dv=3
2150 print" q prepare the printer.
2152 print"press a key to continue."
2154 print"press 'a' to abort.
2156 gosub 1500
2158 if key$="a" then goto 2202
2160 open 3,dv
2162 print#3,left$(bl$,(80-len(t$))/2) t$
2164 print#3
2166 if h$<>"y" then goto 2178
2168 for i=1 to f
2170 print#3,left$(fd$(f(i))+bl$,w(i));"  ";
2172 next i
2174 print#3
2176 for i=1 to 80 : print#3,"=";: next i
2178 print#3
2180 for i=b to e
2182 for j=1 to f
2184 print#3,left$(rec$(r%(i),f(j))+bl$,w(j));"  ";
2186 next j
2188 print#3
2190 get a$
2192 if a$="" then 2196
2194 gosub 1500
2196 next i
2198 print#3
2200 close3
2202 return
```

Home Budget

Brian Dobbs
Timmins, Ont.

Have you counted the number of personal finance programs on the market? Can't decide which one to buy? Can't decide if you even need one? Home Budget will let you put both those decisions on the back burner. Then, once you enter it, you might find it's all you need. And, unlike the others, you can add your own custom modifications - a truly economical alternative.

Home Budget was written for the Commodore 64 with disk and printer. The program allows the user to keep records of monthly spending within a household, and view spending trends for any number of monthly bills.

On running the program, the first thing that comes up is the menu. You can:

1. Write and View data for monthly bills
2. View barchart trends of bills for a 1 year period
3. Receive Printout of all 12 bills for a 1 year period
4. Load and Save data of bills as a file on disk
5. Create initial file.

The first thing to do is create an initial file. Upon choosing this selection you are asked to fill in 12 bills that are paid on a monthly basis (mortgage, hydro, telephone, etc.). Then enter a filename and the program will create a file with the 12 bills and 144 zeros (12 bills for 12 months) You now have a file to work with and can update it every month as the bills come in.

At this point you no longer need to create a new file for the data to be entered. Simply select "Write Monthly Bills" each time you wish to add data for a new bill.

Once you have some data filled in, you can "View Monthly Bills". All the data for a particular month as well as the monthly total spent will be displayed.

A barchart trend of each bill can also be viewed. The chart will cover a 1 year period. It enables the money conscious person to view spending trends for each bill. The only requirement here is that the user supply a maximum scale for the chart since this can vary from bill to bill.

An added feature of this program is to get hard copy printout consisting of the 12 bills for the 12 months. This is handy if a permanent record of yearly spending is required, like at income tax time.

The user has access to limitless data for trend analysis simply by creating new initial files.

In this state of uncertain economic times I have become very conscious of how I spend my hard earned money. Using Home Budget I can now monitor all my spending to try and cut down on spending wastefully. Home Budget is a must for any money conscious household.

```
10 rem**********************
20 rem*                     *
30 rem*     home budget     *
40 rem*          by         *
50 rem*      brian dobbs    *
60 rem*                     *
70 rem*    timmins,ontario  *
80 rem*                     *
90 rem**********************
100 poke53280,12 : poke53281,0 : dima$(13,13)
    : goto180
110 gosub1230
120 open14,8,14, "0: " + nm$ + ",s," : x = 1
130 y = 2
140 input#14, a$(x,y) : y = y + 1 : if y>13 then160
150 goto140
160 x = x + 1 : if x>13 then close14 : goto180
170 goto130
180 print " Sqq " tab(14) " budget menu qq "
190 printtab(10) " 1- r w Rrite monthly bills " : print
200 printtab(10) " 2- r v Riew monthly bills " : print
210 printtab(10) " 3- r b Rarchart trend of bills " : print
220 printtab(10) " 4- r p Rrinter trend of bills " : print
230 printtab(10) " 5- r s Rave data to disk " : print
240 printtab(10) " 6- r l Rload data from disk " : print
250 printtab(10) " 7- r c Rreate initial file " : print " qq "
260 printtab(12) " select choice ? " : y = 0
270 get an$ : if an$ = " " then 270
280 for x = 1 to 7 : if an$ = mid$(" wvbpslc ",x,1) then y = x
290 next : on y goto410,300,500,730,960,110,990 : goto180
```

```
300 print " Sqqqq " : input " enter month to view " ;an$
    : gosub1020
310 print " S " : printtab(15)an$ : print " qq " : y = 2
320 printa$(1,y) : y = y + 1 : if y>13 then 340
330 goto320
340 print " QQQQQQQQQQQQ " : y = 2
350 printtab(15)a$(x,y) : y = y + 1 : if y>13 then 370
360 goto350
370 gosub1300
380 print " qq monthly total " tab(14)q + aa + bb
390 get a$ : if a$ = " " then 390
400 goto180
410 print " Sqqqqq " : input " what month to write bills " ;an$
420 gosub1020
430 y = 2 : print " S " : printtab(15)an$ : print " qq "
440 printa$(1,y) : y = y + 1 : if y>13 then 460
450 goto440
460 print " QQQQQQQQQQQQ " : y = 2
470 printtab(15) : inputa$(x,y)
480 y = y + 1 : if y>13 then 180
490 goto470
500 input " Sqqqq enter name of bill " ;an$
510 for p = 2 to 13
520 if an$ = a$(1,p) then y = p
530 next
540 print " qq what is maximum scale for " a$(1,y)
    : input " qq " ;b
550 print " S " : printtab(20-(len(a$(1,y))/2))a$(1,y)
560 print " qq " b : z = 0
570 z = z + (b/20) : if z = b then 590
580 printint(b-z) : goto570
590 r = 1992 : x = 2 : t = 56264 : u = 2
600 for e = r to r-(40*((val(a$(x,y))/(b/20)))) step-40
    : pokee,224 : next
610 for f = t to t-(40*((val(a$(x,y))/(b/20))))step-40
    : pokef,u : next
620 r = r + 2 : x = x + 1 : t = t + 2 : u = u + 1
630 if u = 3 then u = 1
640 if x = 14 then 660
650 goto600
660 r = 1992 : t = 56264 : poker,138 : poket,1 : poker + 2,134
    : poket + 2,1 : poker + 4,141
670 poket + 4,1 : poker + 6,129 : poket + 6,1 : poker + 8,141
    : poket + 8,1 : poker + 10,138
680 poket + 10,1 : poker + 12,138 : poket + 12,1
    : poker + 14,129 : poket + 14,1 : poker + 16,147
690 poket + 16,1 : poker + 18,143 : poket + 18,1
    : poker + 20,142 : poket + 20,1 : poker + 22,132
700 poket + 22,1
710 geta$ : if a$ = " " then 710
720 goto180
730 open4,4 : print#4,tab(38) " budget 1984 " : print#4
    : l = 2 : m = 7
740 print#4, " bill[15spc] " ;
750 print#4, " january[3spc]febuary[3spc]march[5spc]
    april[5spc]may[7spc]june "
760 gosub850
770 l = 2 : m = 7
780 gosub890
790 l = 8 : m = 13 : print#4, " bill[15spc] " ;
800 print#4, " july[6spc]august[4spc]september[1spc]
    october[3spc]november[2spc]december "
810 gosub850
820 l = 8 : m = 13
830 gosub890
840 close4 : restore : goto180
850 for y = 1 to 13 : z = 20-len(a$(1,y))
860 print#4,a$(1,y)tab(z);
870 for i = l to m : print#4,(a$(i,y))tab(10-len(a$(i,y))); : nexti
    : print#4,chr$(10)
880 nexty : print#4 : return
890 print#4, " monthly total " tab(6)
900 for x = l to m
910 gosub1300
920 c = q + aa + bb : c$ = str$(c)
930 print#4,q + aa + bbtab(9-len(c$)); : nextx
940 print#4,chr$(10)
950 return
960 gosub1230
970 open14,8,14, " @0: " + nm$ + " ,s,w "
980 x = 1 : goto1170
990 print " Sq a total of 12 bills will be entered. qqq "
    : x = 1 : y = 2
1000 input " name of bill " ;a$(x,y) : y = y + 1 : if y>13 then 1070
1010 goto1000
1020 for w = 1 to 12
1030 readd$
1040 if an$ = d$ then x = w + 1
1050 next
1060 restore : return
1070 print " S " : x = 2
1080 y = 2
1090 a$(x,y) = " 0 "
1100 y = y + 1 : if y>13 then 1120
1110 goto1090
1120 x = x + 1 : if x>13 then 1140
1130 goto1080
1140 gosub1230
1150 open14,8,14, " 0: " + nm$ + " ,s,w "
1160 x = 1 : print " qqr creating initial file R "
1170 y = 2
1180 print#14,a$(x,y) : y = y + 1 : if y>13 then 1200
1190 goto1180
1200 x = x + 1 : if x>13 then 1220
1210 goto1170
1220 close14 : goto180
1230 input " Sqqr filename R " ;nm$
1240 print " qq press ' r f1 R ' to continue "
1250 geta$ : if a$ = " " then 1250
1260 if a$ = chr$(133) then return
1270 goto1260
1280 data january,february,march,april,may,june,july
1290 data august,september,october,november,december
1300 q = (val(a$(x,2))) + (val(a$(x,3))) + (val(a$(x,4)))
     + (val(a$(x,5)))
1310 aa = (val(a$(x,6))) + (val(a$(x,7))) + (val(a$(x,8)))
     + (val(a$(x,9)))
1320 bb = (val(a$(x,10))) + (val(a$(x,11))) + (val(a$(x,12)))
     + (val(a$(x,13)))
1330 return
```

Your BASIC Monitor Part 3: The Assembler

Bob Drake
Brantford, Ont.

Well, here it is, the last installment in our small saga. The first thing we have to do is swat a couple of bugs. Must be summer the way those things sneak into the code. The first one is a counting error in the disassembler. The fix is in line 9440 and goes like this:

```
9440 m=m+1-1*(m<5)-2*(m>4 and m<10)
```

The second bug lies with the calculation of the offset or jump in the relative instructions. The low byte was not converted to decimal so that BASIC could handle it. Here's the fly swatter:

```
9575 by=peek(m+1)
9580 if by>128 then by=by-256
```

I have also enhanced the first installment of the program by deciphering the status register. The SR holds seven bits that track the current conditions in the processor. The bits or flags are:

S – sign (1 means negative)
V – overflow
B – break (1 if a break has been executed
D – decimal mode (1 if on)
I – interrupt disable status (1 means interrupts are disabled)
Z – zero status (1 for zero, 0 for non-zero)
C – carry status

The new code displays the SR in hex and on the next line breaks the hex value into binary and lists the current flag values. Here's the code – there isn't much involved for the extra information gained.

```
2080 print
2081 s$=" "
2082 for z=1 to 8
2083 b%=by/2↑(8-z)
2084 by=by-b%*2↑(8-z)
2085 s$=s$+str$(b%)
2086 next
2087 print"flags r s v  b d i z c"
2088 print"     "s$ :rem 5 spaces
2090 next pr
```

The work is done in lines 2083 – 2085. The first line, with the aid of Z in the for–next, divides the SR value by powers of 2 and takes the integer value. Line 2084 finds the remainder. The bit value (b%) is added to the status string in 2085 and that's that.

Let's get to the assembler. The easy parts are the modifications to the original program so that it knows the assembler is attached.

```
220 r$="xmrpslg*cda"
350 on r gosub 10,1000,2000,3000,4000,4140,5000,
    6000,7500,9000,10000
```

The assembler itself can be a very tricky thing to write depending on the complexity desired. The assembler presented here does not allow labels or macros. Like the rest of the program it is meant as a learning tool but one that works at the same time. Pressing A enters the assembler with a request"FROM?". Enter your hexadecimal start address. On the C64, $C000 is a safe place to put your ML code.

Lines 10060 – 10240 make up a very simple editor. A cursor is printed and the program waits for an entry. Line 10120 handles the delete key, 10130 filters out most possible illegal characters and 10140 makes it difficult to enter double spaces. When you finish the entry by pressing return, 10220 and 10230 strip spaces off both ends of the code entered. Why do all this work? The hardest part to writing the assembler is figuring out which addressing mode is in use. Extra spaces make this parsing harder.

If all has gone well, the mnemonic is the first three letters on the left end ofthe string (MN$). Line 10250 gets this code and 10260 – 10380 go looking through MN$(1) to MN$(4) in the disassembler data. Since MN$(2) has all those codes using implied addressing and MN$(3) has all the relative addressing, etc., the string where the mnemonic is found sets up the possible the addressing mode in use. I handled the easiest first.

Lines 10420 – 10470 handle the implied addressing codes. Note line 10420. The address or operand for the mnemonic is extracted from the list of op codes by its position. This is printed in hex, converted to decimal, and poked into place. A similar procedure is used on all the other op codes. Lines 10480 – 10620 handle the relative addressing mode. And jumps are deciphered in lines 10630 – 10870. Line 10660 represents one of my favorite tricks which is expressed as "Why do something the hard way if there is an easy way?". I know JSR is $20 and decimal 32. Instead of looking up the values and converting them (which is relatively slow) I just poke it and print it and charge ahead.

Look at lines:

```
10670 a=8 : b=6
10680 gosub 11650
```

The subroutine at 11650 is a workhorse that takes the assembler's input address, for example $FFD2, breaks it into a hi byte (FF), a low byte (D2), converts to decimal and pokes the values into place the way the 6502 expects them, backwards! That is, it pokes in the decimal equivalents of D2 and FF. The A and B values are critical. They locate the expected positions of those bytes within the code typed in. If the format expected is not followed, the assembler will give an error. That is one of the nice things about a BASIC assembler. If it 'crashes', all you do is type RUN to get it going again.

Lines 10880 – 11560 handle all the other addressing modes for the mnemonics found in MN$(1).

As I have implied, the format for entering your code is important. The format I have followed is standard 6502. Here is a summary to get you going:

```
Immediate mode      LDA #$23
Zero page           LDA $23
Indexed zero page   LDA $23,X
                    STX $23,Y
Absolute            LDA $FFD2
Indexed absolute    LDA $FFD2,X
                    LDA $FFD2,Y
Indexed indirect    LDA ($FFD2,X)
Indirect indexed    LDA ($FFD2),Y
Accumulator         ASL
Implied             CLC
Indirect            JMP ($FFD2)
```

Much of the power of commercial assemblers lies with the ability to use labels as reference points within the program. This simple assembler does not have that feature. Jumps and branches must refer to the address at the end of the jump. Offsets are calculated for you.

```
Relative            BCC $FFD2
```

Few changes are needed for a VIC. Add line:

```
10245 print
```

The lines printing error messages, such as 10370, print a carriage return first.

```
10370 print cr$ "*ERROR* UNKNOWN MNEMONIC"
```

You will also have to go through the code (10430 –) to remove the spaces used to line up the hex codes on the C64. But that's all the changes that are needed. Good luck with your adventures into machine language.

```
10000 rem basmon assembler
10010 print "rassemble"
10020 print "quit by entering 'end'"
10030 input "from";m$
10040 gosub 7120
10050 o=m  :rem origin
10060 print m$" R=";:rem shift r
10070 mn$=""
10080 a$=""  : get a$ :rem get the code
10090 if a$="" then goto 10080
10100 if a$=cr$ then 10190
10110 a=asc(a$)
10120 if mn$<>'' and a=20 then mn$=left$
      (mn$,len(mn$)-1) : printa$; : b=a : goto10080
10130 if a<32 or a>91 then goto 10080
10140 if a=32 and b=32 then goto 10080
        :rem disallow double spaces
10150 mn$=mn$+a$
10160 print a$"R=";
10170 b=a
10180 goto 10080
10190 print"  ";
10200 rem - got the code - now parse it
10210 rem strip spaces from either end
10220 if left$(mn$,1)="  " then mn$=mid$(mn$,2)
      : goto 10220
10230 if right$(mn$,1)="  " then
      mn$=left$(mn$,len(mn$)-1) : goto 10220
10240 if mn$="end" then print : return
10250 o$=left$(mn$,3) :rem mnemonic op code
10260 fl=0
10270 for i=1 to 4
10280 l=len(mn$(i))
10290 for j=1 to l step 3
10300 if o$<>mid$(mn$(i),j,3) then goto 10350
10310 fl=1  :rem found
10320 po=(j+2)/3 :rem position
10330 s=i  :rem which mnemonic string
10340 j=l+1  :rem exit loops nicely
10350 nextj,i
10360 if fl<>0 then 10390
10370 print"*error*unknown mnemonic"
10380 goto10060
10390 if s=1 then goto 10880
10400 if s=3 then goto 10480
10410 if s=4 then goto 10630
10420 if s=2 then m$=mid$(op$(22),po*2-1,2)
        :rem implied
10430 print "        " m$  :rem 10 spaces
10440 gosub 7120
10450 poke o,m
10460 o=o+1
10470 goto 11530
10480 rem relative addressing
10490 m$=mid$(op$(23),po*2-1,2)
10500 print"   "; :rem 4 spaces
10510 gosub 11570
10520 poke o,m
10530 m$=mid$(mn$,5) :rem calculate offset
10540 gosub 7120
10550 by = m-o-2
```

```
10560 if by>127 or by<-128 then print "*error* too far"
      : goto 11530
10570 if by<0 then by=256+by
10580 gosub 7000
10590 print by$
10600 poke o+1,by
10610 o=o+2
10620 goto 11530
10630 rem jumps
10640 if o$<>"jsr" then goto 10700
10650 poke o,32
10660 print"   20 "; :rem 4 spaces, '20', space
10670 a=8 : b=6
10680 gosub 11650
10690 goto 11530
10700 rem jmp
10710 fl=0
10720 if right$(mn$,1)=")" then fl=1 :rem indirect
10730 poke o,76+32*fl
10740 by$=mid$(mn$,6+fl,4)
10750 m$=mid$(by$,3)
10760 o$="   4c " :rem 4 spaces, '4c', space
10770 if fl=1 then o$="  6c " :rem 2spcs, '6c', 1spc
10780 print o$;
10790 gosub 11570
10800 poke o+1,m
10810 m$=left$(by$,2)
10820 gosub 11570
10830 poke o+2,m
10840 o=o+3
10850 fl=0
10860 print
10870 goto 11530
10880 rem all the rest - parse to find addressing mode
10890 if mid$(mn$,5,1)<>"#" then goto 11000
      :rem immediate
10900 mo=1
10910 print"    "; :rem 5 spaces
10920 gosub 11600
10930 if m$="**" then goto 11530
10940 m$=mid$(mn$,7,2)
10950 gosub 11570
10960 print
10970 poke o+1,m
10980 o=o+2
10990 goto 11530
11000 if len(mn$)<>3 then goto 11080
11010 mo=10 :rem accumulator
11020 print"       "; :rem 10 spaces
11030 gosub 11600
11040 if m$="**" then goto 11530
11050 print
11060 o=o+1
11070 goto 11530
11080 if len(mid$(mn$,5))<>3 then 11190
11090 mo=2 :rem zero page
11100 print"     "; :rem 6 spaces
11110 gosub 11600
11120 if m$="**" then goto 11530
11130 m$=mid$(mn$,6)
11140 gosub 11570
11150 poke o+1,m
11160 print
11170 o=o+2
11180 goto 11530
11190 if len(mid$(mn$,5))>5 or right$(mn$,1)="x" or
      right$(mn$,1)="y" then 11270
11200 mo=5 :rem absolute
11210 print"    "; :rem 4 spaces
11220 gosub 11600
11230 if m$="**" then goto 11530
11240 a=8 : b=6
11250 gosub 11650
11260 goto 11530
11270 rem xy modes
11280 if len(mn$)<>9 then 11390
11290 mo=3 :rem zero x,y
11300 if right$(mn$,1)="y" then mo=4
11310 print"    "; :rem 4 spaces
11320 gosub 11600
11330 m$=mid$(mn$,6,2)
11340 gosub 11570
11350 poke o+1,m
11360 o=o+2
11370 print
11380 goto 11530
11390 if len(mn$)<>11 then 11470
11400 print"  "; :rem 2 spaces
11410 mo=6 :rem absolute x,y
11420 if right$(mn$,1)="y" then mo=7
11430 gosub 11600
11440 a=8 : b=6
11450 gosub 11650
11460 goto 11530
11470 rem indirect xy are the only ones left
11480 mo=8
11490 if right$(mn$,1)="y" then mo=9
11500 gosub 11600
11510 a=9 : b=7
11520 gosub 11650
11530 n=o
11540 gosub 7030
11550 m$=by$
11560 goto 10060
11570 print m$"  "; :rem print byte,convert
11580 gosub 7120
11590 return
11600 m$=mid$(op$(po),mo*2-1,2)
      :rem locate op code,convert,print,poke
11610 if m$="**" then print"*error* illegal mode" : return
11620 gosub 11570
11630 poke o,m
11640 return
11650 m$=mid$(mn$,a,2) :rem a locates lo byte
      :rem poke lo,hi bytes
11660 gosub 11570
11670 poke o+1,m
11680 m$=mid$(mn$,b,2) :rem b locates hi byte
11690 gosub 11570
11700 poke o+2,m
11710 print
11720 o=o+3
11730 return
```

Structured Programming in Commodore BASIC

Chris Zamara

Telling GO TO Where TO GO TO

Warning: Careless Use of GOTOs Can Be Hazardous to Your Program's Health

When computer programming is taught in school today, the techniques of "structured programming" are usually stressed. If programmers follow those techniques, they create programs that are easier to understand, de-bug, modify, and get working the first place. This article explains how to use structured programming techniques, even though Commodore BASIC is not a structured language.

First, a little review of structured programming – the new and exciting stuff comes later. The basic theory of structured programming simply states that in any program, only three types of "structures" are needed: sequence, controlled looping (WHILE. . .ENDWHILE), and branching (IF. . .THEN. . .ELSE. . .ENDIF). The first of these, sequence, implies a sequential execution of statements with no transfer of control to any other statements. That, of course, is no problem in Commodore BASIC (or any language). The WHILE structure gives the ability to execute a group of instructions WHILE a given condition is true. Another form of WHILE is the UNTIL structure, which is similar, but the group of instructions are always executed at least once and repeated UNTIL the test condition is true (the UNTIL statement appears at the end of the loop, WHILE at the beginning). The final structure, IF. . .THEN. . .ELSE. . .ENDIF, allows execution of one of two groups of statements, depending on the result of the test condition. Examples of WHILE and IF. . .THEN structures will be given later, when we look at how to use them in Commodore BASIC.

As mentioned before, using nothing but these three structures when writing programs makes for clean, elegant, and easily readable code. Languages such as COMAL, Pascal, and Waterloo BASIC have the abovementioned structures, as well as others, built in. But since Commodore BASIC makes no provisions for programming this way, many people just forget all about the structures, and hack their way through a program, throwing in GOTOs willy-nilly, until it finally works. While it is true that GOTOs are needed sometimes to simulate the structures, that's all they should be used for – no jumping around haphazardly. Why be so careful with our gentle old friend GOTO? Read on.

The Problem With GOTO

Here's a bit of BASIC contrasted with a structured language. To wait until a key is pressed on the keyboard, a common method of coding would be:

```
100 GET A$: IF A$ = "" GOTO 100
```

The same code written in COMAL, using a WHILE loop, might read:

```
GET A$
  WHILE A$ = ""
  GET A$
ENDWHILE
```

Or, using UNTIL:

```
REPEAT
  GET A$
UNTIL A$<>""
```

What is wrong with the BASIC version? Well, first of all, the code is location–specific because of the implied GOTO at the end of the line. If this code were to be used in another part of the program, it would have to be changed.

Secondly, GOTOs (or implied GOTOs in THEN clauses) can be very frustrating when some line rearranging is being done. In a long program, there is no way of knowing if any given line is referenced by a GOTO somewhere, except by looking through the entire program. Thus, changing a statement's line number, perhaps to make room for some new lines being inserted, is just asking for trouble. Deleting a line can also result in premature program death, when some long–forgotten GOTO tries doing its job and causes a frightening "?UNDEF'D STATEMENT ERROR". Of course, according to Murphy's law(s), this bug will only be found when you are trying to impress people by demonstrating that your amazing program finally works ("no, wait, this time for sure!").

Thirdly, a program containing many GOTOs is likely to look like "spaghetti code", with branches going all over the place. This kind of code is very difficult to understand, and even harder to de–bug. This is because, as mentioned earlier, you never know what was executed before any given line in the program.

Finally, GOTOs are slow to execute, especially in long programs, since the BASIC interpreter must scan through each program line until it finds the target line number. Thus, our infamous spaghetti code program, besides being difficult to understand and de–bug, is also inefficient.

A Better Way

Now that I've convinced you never to use another GOTO again in your life, what's the alternative? Standard PET BASIC does not contain WHILE or UNTIL statements, and who wants to load up a new language every time the machine is turned on? Well, there is one control structure available in PET BASIC which can help us in a few not very obvious ways – the old faithful FOR. . .NEXT loop.

How can FOR. . .NEXT replace GOTOs? Consider the above keyboard get routine. Now look at the slightly strange code below.

```
100 FOR I=0 TO 1  : REM* LOOP *
110 : GET A$
120 : I=-(A$<>"") : REM* UNTIL A$<>""
130 NEXT I
```

These statements will wait until a key is pressed, and then continue, just like the earlier example. Line 100 ensures that the FOR. . .NEXT loop will terminate as soon as the index, in this case 'I', is one or greater. The index is set to zero or one by the little boolean expression in line 120. The bracketed expression will be set to –1 if it is true (A$<>"") or 0 if false (A$=""). The result is negated to yield values of 1 or zero. Thus, as long as A$="", or no key is pressed, 'I' equals zero and the FOR. . .NEXT loop keeps repeating. As soon as a key is pressed, the expression becomes false and 'I' gets set to one – the NEXT then allows the loop to end.
If you're now calling me a hypocrite after I just complained about hard–to–read code, note that once you know and understand this structure (as you hopefully do now), its use and comprehension in the future will be easy, if not automatic.

What are the advantages of using this weird technique? The main reason is, we don't need a GOTO. The virtue of that has already been explained. It may seem a disadvantage that the code to wait for a key using FOR. . .NEXT took four lines, while the equivalent code using a GOTO only required one, but if you're interested in conserving lines, consider this:

```
100 FOR I=0 TO 1 : GET A$ : I=-(A$<>"") : NEXT I
    : PRINT "KEY ENTERED: ";A$
```

A statement is performed after the key is pressed, on the same line. This could not be done using the GOTO method, since any statement placed after the GOTO would never be executed. That's another disadvantage of using GOTOs, by the way.

Sharp readers will point out that this make–shift WHILE loop is actually an UNTIL loop, since the code within the loop is executed at least once, even if the test condition is false to begin with. True enough, but the UNTIL structure is well suited to most situations. If a true WHILE loop is needed, where the condition is checked at the beginning of the loop, it may be simulated with GOTOs like this:

```
1500 IF (condition) GOTO 2010  '* WHILE LOOP *

  any code within while loop

2000 GOTO 1500
2010 REM— ENDWHILE —
```

A final note regarding FOR. . .NEXT loops: as reported in this issue's Bits & Pieces section, an infinite loop can be set up using a STEP 0 clause after the FOR statement, as in:

```
FOR I=0 TO 1 STEP 0.
```

The Exception

Unfortunately, using FOR. . .NEXT loops cannot eliminate GOTOs altogether. The IF. . .THEN. . .ELSE. . .ENDIF structure must be simulated using GOTOs. An example of this type of structure is:

```
IF A = 1 THEN
.
.(process to perform when a = 1)
.
ELSE
.
.(process to perform when a<>1)
.
ENDIF
```

We must use GOTOs, but there is a way to use them so that the problems discussed earlier will not cause too much anguish. Examine the following translation:

```
 10 IF A<>1 THEN 60
 20   .
 30   .(process to perform when a = 1)
 40   .
 50 GOTO 100
 60 REM — ELSE —
 70   .
 80   .(process to perform when a<>1)
 90   .
100 REM — ENDIF —
```

By making the target lines for both GOTOs REM statements, we no longer have to worry about re-arranging lines, since we know that the only lines which are targeted are the "ELSE" and "ENDIF" remarks. The only problem is, the target line numbers are not known at the time the GOTOs are written, but they can be filled in once the entire structure is completed. Now that we have WHILE. . .ENDWHILE and IF. . .ELSE. . .ENDIF constructs in our programming arsenal, we can tackle any programming problem.

An Example

Entire programs may be written without using GOTOs, as shown in the example below. This program counts how many words per minute you type, starting after the first character is typed. Pressing RETURN will give the WPM count and start over, and pressing up-arrow ends the program.

```
10 rem* count words per minute - cz 1984
11 rem* this program allows input of a number of
12 rem* words separated by any number of spaces,
13 rem* and followed by a carriage return.
14 rem* it then reports on the number of words,
15 rem* the time taken, and the resulting typing
16 rem* speed in words per minute (neglecting errors).
17 rem* the entire program is written without using
18 rem* gotos, using for. . .next loops to simulate
19 rem* 'until' loops.
100 :
110 for k = 0 to 1              :rem* main loop
115 : a0$ = chr$(0)             :rem* previous character
117 : c = 0                     :rem* character count
120 : wrds = 0                  :rem* word count
121 :
125 : for i = 0 to 1            :rem* "UNTIL" loop
126 : if c = 1 then bt = ti     :rem* start timer after first
                                      char.
130 : get a$
140 : if a$ = " " and a0$<>" " then wrds = wrds + 1
150 : print "R r[1left] ";a$;  :rem* print fake cursor
160 : if a$<>" " then a0$ = a$: c = c + 1  :rem* add to
                                      character count
170 : i = -(a$ = chr$(13) or a$ = "↑")  :rem* do until
                                      return or up-arrow
180 : next i
190 :
192 : tm = ti-bt: min = tm/60/60  :rem* convert jiffies
                                      to minutes
195 : if c>1 then wrds = wrds + 1  :rem* add last word
200 : print wrds " words, " min " minutes"
202 : wpm = int(wrds/min + .5)  :rem* calculate words
                                      per minute
204 : print wpm " words per minute."
205 : k = -(a$ = "↑")           :rem* do until up-arrow key
                                      struck
210 next k
220 end
```

Note the way the body of the loops are indented to show how they are nested. This indentation, along with the liberal use of spaces and tidy comments, makes for a very readable and maintainable program. Also, since no GOTOs or implied GOTOs are utilized, the line numbers are inconsequential, as long as they are in sequence.

A Final Thought

Using a structured style will make your programs easier to debug, and much easier to modify. When you must use GOTOs, use them judiciously, and label all GOTO target lines with comments. Structured programming in Commodore BASIC is possible, so (carefully) GOTO it!

Lincoln College Commodore Campers Key In on Computers

Robert D. Widmer, Associate Dean, Lincoln College
Lincoln, Illinois

America's first female vice presidential nominee has been named, but that historic event practically goes by unnoticed. Newspapers, radios, and televisions appear as alien as "ET."

Participants at Lincoln College's week-long, Commodore Computer Camp seem oblivious to the world beyond the college gate. They can count the number of times they left the college campus on two fingers. Through intense concentration they attempt to squeeze in every bit of learning possible about their passions – their Commodore computers.

Some 65 computer enthusiasts from across the country gathered in Lincoln, Illinois, last July for Lincoln College's second annual summer camp, the only workshop in the U.S. offered to adult Commodore computer users.

The participants were experienced computer users seeking to expand their knowledge for both work and hobby. Workshop leaders, nationally known Commodore experts, saturated the campers with information.

"I hear people saying they are getting such an enormous amount of information and understanding it at a very surface level. We'll have to leave and really do some work so that it soaks in," remarked camper Ellen Payson of San Antonio, Texas, on the second to last day of the camp. Payson is a public school teacher who trains other teachers to use computers.

Workshop topics included maintaining and improving computer equipment, introductory and advanced disk handling techniques, machine language, assembly language programming and introduction of Commodore 64 sound and graphics. The campers selected a two and a half hour morning session and a different afternoon class and stayed with those classes the entire week.

Instructors included Jim Butterfield, an internationally known Commodore "expert" and author of numerous books about Commodore computers; Jim Strasma, a Lincoln College computer science instructor and editor of Midnite/Paper; and Dick Immers, known in Commodore circles as "the disk doctor" because of his expertise with the Commodore 1541 disk drive.

Other camp leaders included Jim Tucker, a certified Commodore technician; Steve Michael, a computer science

teacher from Sauk Valley College and Mike Todd, a computer hardware and software authority from England.

Camper Mike Spengel from Arlington, Virginia appreciated the spirit and tenor of the camp's atmosphere.

"This place is run very much the way graduate school should run because of the relationship between teachers and students," Spengel said. "I like the small classes and the 'we're in this together type of feeling' rather than the adversarial relationship of 'if you don't get this we're going to smear your career' type of attitude. Also, the students are being treated like adults and in many academic surroundings you don't find that."

Age and sex barriers found no place at the camp.

Todd Colacino, an eighth grader from Newark, N.Y., basked in the treatment he received as one of the youngest campers.

"I thought because there aren't too many kids here that they'd just pass over me in the classes, but they've given me time and they listen to my questions," he said.

Todd's father, Ron, noted, "The teacher-student relationship is fantastic. There is an intellectual exchange going on that is unbelievable here. There are no barriers anywhere."

Women accounted for more than one-fourth of the participants much to the surprise and the delight of many of the women.

"I thought there would be only one or two women besides me. In Phoenix, there is no one I can talk to about computers on the Commodore level," remarked Becky Boren of Phoenix, Arizona. "One of the reasons we're so tired (at the camp) is that we'll talk past midnight and all we'll talk about is computers."

This mother of six children sets up elementary school computer labs on a voluntary basis; she also teaches beginning programming skills.

"My children are proud of me and my work with computers. They say nobody's got a mom who is a computer mom," Boren noted.

Gail and Tim Perrin, a married couple from Milwaukee, Wisconsin, both plan to further their educations and careers by boosting their present computer literacy.

"I came to the camp looking for some background to help me if I go back to get a master's degree in computer education," said Gail, who operates a group home for mentally retarded adults.

Tim Perrin, an industrial designer who wants to return to college for a degree in robotics engineering, said the camp heightened his knowledge in the proper uses and concepts related to computers.

Besides the daytime classes, special lectures were conducted each night of the camp. Those lectures proved to be "the icing on the cake," as one camper observed. Topics ranged from a mini-refresher course on basic computer programming techniques such as the proper method of closing files to a session on new hardware and software "toys." Participants got another opportunity to quiz the experts and share information and experiences among each other.

"The evening programs make everything that much more delightful," said Ellen Payson. "You cannot help but feel a little confined by the two classes you are enrolled in that perhaps you are missing some of the other goodies being covered in other classes. The evening seminar gives you a little bit of what's been going on in the other classes."

While many of the campers admitted to scratching their heads while wondering "where is Lincoln, Illinois" when they signed up for the camp, they found the Lincoln College campus exactly to their liking. "The college bent over backwards to meet every request we had," one camper observed.

Lincoln itself, a community of some 16,000 residents located in Central Illinois surrounded by Springfield, Decatur, Bloomington and Peoria, provided a peaceful atmosphere with few distractions which allowed the campers to focus on the subject so dear to their hearts.

"The degree of concentration on the subject matter here is comparable to when a ship goes out to sea and stays there for three months. The amount of concentration you can put into your work peaks," Mike Spengel said.

So the swimming and tennis facilities available on campus went ignored. Most of the "free time" scheduled in each day found the campers still sitting in front of their computers or clustered in small groups talking shop.

In fact the only short-coming mentioned universally by the campers was a need for "six more hours in the day."

Speller: A Drill Program Using Vectors

Robert Drake
Brantford, Ont.

A VECTOR is an ordered n-tuple. This is the formal term and definition mathematicians and physicists use to describe what the rest of us call lists. Lists are easy things to work with on paper and just about as easy on your computer. To work with a list you first tell the computer that you are going to use a list. Then you 'load' the list. That is you will either read data (from data statements or input data (from the keyboard, tape or disk drives) to fill the list. Once the list is created, you can do almost anything you like with it, handling it exactly the way you use ordinary variables. Usually lists are processed in several ways, including:

* retrieving items from the list
* editing the list to change an item or add items
* searching the list to find an item
* sorting the list alphabetically or numerically

We'll look at only the first two of these here using a simple application.

My eight year old son has problems with reading and spelling. He needs practice with both. We found that the traditional spelling program showed the word for a short period of time and then asked him to re-type it. This just tested his memory. No comprehension of the word's meaning or definition was required. His spelling skills also weren't really being tested because he was shown the correct spelling and then just asked to repeat it. No reading was required. We sat down together and designed a little program to help him with both of his problems. The program works like this.

1. Print a title on the screen
2. Offer some instructions with a menu
3. Create the needed lists
4. Load the lists from data statements
5. Give a definition
6. Ask for a spelling
7. Based on the answer, give hints or congratulate
8. If the answer is right the first time, delete the word from the list
9. Repeat steps 5 to 9 until all the words are used up or until he wants to quit.
10. List the words there was trouble with and the number of attempts that were made with each. Also list the words that weren't attempted.
11. Go back and offer to repeat the program.

This is my normal way of implementing a program. Detail what you want to do and then program each step one at a time. Let's do these one at a time.

The first step is to print a title on the screen. Lines 100–260 actually do more than that. They print a title in lines 110–130, initialize the program in 130, and present a menu of choices in 140–240. Lines 130 and 230 use Boolean algebra. Basically, a value is "false" if it is zero and "true" if it is –1. So, line 130 says, if F1 is not true then gosub 300. Line 190 sets F2 to false, and line 230 says if F2 is true then goto 110. F1 and F2 are "flags" to the program. The flags are signals to the program to do or not do something. We don't want the program to do the initialization twice, thus the 'if not F1' in line 130. Line 230 is a flag which is set to true if the instructions are used. If we have gone to the instructions then on the return we have to clear the screen and reprint the menu. Lines 250 and 260 exercise the QUIT option. On the VIC will require some playing with the tabs to make it look right.

```
100 rem speller * copyright 1984 * robert drake * free
      to copy – not to sell
110 print chr$(14) " Sqqqqqq " tab(16) " rSPELLER "
120 print " q " tab(10) " Bob Drake – February 1984 "
130 if not f1 then gosub 270
140 print tab(14); " qr R Run the program "
150 print tab(14); " qr I Rnstructions "
160 print tab(14); " qr Q Ruit "
170 print " q " tab(12) " Press 'R', 'I', or 'Q' "
180 get a$
190 f2 = 0
200 if a$ = " r " then gosub 580
210 if a$ = " i " then gosub 400
220 if a$ = " q " then goto 250
230 if f2 then goto 110
240 goto 180
250 print " S "
260 end
```

Let's look at setting up the lists. The program uses 3 lists. They are called W$ (for words), D$ (definitions), and S% (score). Lists can be used with strings or words, for real numbers (including fractions and decimals), and for integers. The creation of the list is done in line 330. DIM (dimension) tells the computer to set aside space for the lists. If the list uses less than 10 values or words, then you don't have to tell the computer about it. It is good programming practice though to dimension your vectors regardless of length. Each of the three lists in this program is set at a maximum of 100 entries. If we try to use more than that, the computer will complain loudly with a ?BAD SUBSCRIPT ERROR. The computer will also complain if we try to change our mind about the number of elements later in the program. The error is a ?REDIM'D ARRAY ERROR.

Loops and lists go hand in hand. If you know the number of elements or cells in your vector use a FOR – NEXT loop. If, as in this program you don't know the number of words (or don't want to count them), then use a regular loop and an END OF DATA (EOD) value. The EOD here is *,*. The vector can be loaded with READs and DATA, or INPUTs from the keyboard, tape or disk drive.

The second loop which sets up S% uses a FOR NEXT because we know how many items there are. The –1's indicate that a word has not been attempted.

```
270 rem set up and load the lists
280 f1 = -1
290 print tab(14); " q Initializing Q "
300 dim w$(100), d$(100),s%(100)
310 read w$(i),d$(i)
320 if w$(i) = " * " then 350
330 i = i + 1
340 goto 310
350 for j = 0 to i-1
360 s%(j) = -1
370 next j
380 print " Q " ;
390 return
```

There is nothing difficult about writing simple instructions for a program. Why then do so many programs leave them out? If you are entering this program on a VIC, this is the second spot to look for. Adjust the instructions to fit comfortably on your screen.

```
400 rem instructions
410 print chr$(14) " Sr SPELLER "
420 print " q Hi. This program will give you some "
430 print " q practice with your reading and spelling "
440 print " q skills. I will give you the meaning of "
450 print " q a word and you type in the word that "
460 print " q matches. Press the 'RETURN' key when "
470 print " q you are finished.
480 print " q If you make a mistake, I'll give you a
490 print " q hint. All the words you get right on
500 print " q the first try will be erased. "
510 print " q You can quit by typing 'x' as the word. "
520 print " q Press the 'RETURN' key when you have "
530 print " r finished reading this. R "
540 f2 = -1
550 get a$
560 if a$ = chr$(13) then return
570 goto 550
```

The actual program isn't very long. The variable k counts the number of words presented and answered correctly. J picks a random number between zero and the number of words. This is where a list shows its real value. If you want to pick a question randomly, a list makes the job really easy. Pick a number, any number within the range of the vector. Then just PRINT or compare or do whatever you like with that cell. You can get at the cell with W$(J) or D$(J) or S%(J). If you're looking for something hard, I'm sorry because it isn't here. Line 610 looks for a period as the first character of the word selected. I add a period to those words correctly answered the first time so the program can recognize them as being removed from the list. Lines 620 and 630 print the definition on the screen. You might notice the arithmetic in the TAB in line 630. D is the length of the definition. Therefore, 40–D is the number of spaces left over. Dividing by two gives half the number of spaces and centres the definition. If you are using this program on a VIC, leave this out. The VIC's lines are too short to play with centering. If you have an 80 column screen, just change the 40 to 80. This works with any length of line. Lines 640 – 660 get an answer and compare it to 'x' and then the right answer. As with all programs of this type the answer must be an EXACT match, right down to punctuation and capitals.

```
580 rem the program
590 k = 0
600 j = int(rnd(1)*i)
610 if left$(w$(j),1) = " . " then 600
620 d = len(d$(j))
630 print " S " tab((40-d)/2) d$(j)
640 input " q What is the word " ;a$
650 if a$ = " x " then goto 800:rem exit early
660 if a$ = w$(j) then goto 750
```

If the answer doesn't match then first we count it in line 670 (wr – wrong) and we add one to the integer that matches the word (s%(j)). Line 690 controls the number of tries at a word. The program is set up for 4 tries. If the word is less than 4 letters long, there is one less attempt than number of letters. A three letter word gives two tries. If the word is only one letter long, then you have only one try.

```
670 wr = wr + 1
680 s%(j) = s%(j) + 1
690 if wr = 4 or wr = len(w$(j))-1 or len(w$(j)) = 1 then 730

780 goto 600
```

The program gives hints. Each time a try is made which isn't correct, the program prints the first letter or letters in the word and dashes to represent the other letters in the word. Line 710 will fit on one program line if you use ? for PRINT and leave out the spaces. Otherwise, you may find that you can't get it all into the 80 characters you are allowed. 700 print " Here's a hint. "

```
710 if wr then print " r " left$(w$(j),wr);
    :for l = wr + 1 to len(w$(j)): print " - ";: next: print
720 goto 640
```

If you run out of tries, the word is printed.

```
730 print " qq The word is : " w$(j)
740 get a$ : if a$ = "" then goto 740
```

If the try is right the first time then the program adds a period to the beginning of the word and adds increments k (adds one). S% is changed to zero to show that the word has been tried and was right ont the first try. The number of wrongs is reset to zero so that you start fresh on each word. And, if you have run out of words (k = i), the program skips to the scoring routine in line 800.

```
750 if wr = 0 then w$(j) = " . " + w$(j):k = k + 1:s%(j) = 0
760 wr = 0
770 if k = i then goto 800
```

The first thing done in the scoring routine is to remove the dots from the beginnings of the words using a FOR–NEXT on the vector W$. (Remember, I told you that loops are used a lot with lists.)

```
790 rem score
800 for j = 0 to i
810 if left$(w$(j),1) = " . " then w$(j) = mid$(w$(j),2)
820 next j
```

Set F2 to true so that the menu will print when we go back there. The loop in lines 840–910 does several things. First, it looks through S% to see if there were any wrong questions (values greater than zero). If not, line 920 is executed. Otherwise, 860 runs and m is set to one. Line 870 prints a heading – but only if there was a wrong attempt and only if m = 1. Lines 880 – 920 prints out the troublesome word or words and the number of attempts.

```
830 f2 = –1
840 forh = 0toi–1
850 if s%(h)< = 0 then goto 930
860 m = m + 1
870 if m = 1 then print " S Here are the words you had
    trouble with: "
880 print w$(h); "  ";
890 for l = 1 to 10 – len(w$(h))
900 print " . ";
910 next l
920 print s%(h) + 1
930 next h
940 print
950 if m = 0 then print " Sqq All correct. "
```

This second loop looks through S% one more time. Every –1 value in S% indicates a word not tried. As each value is checked, it is also reset to –1 so the program can be reused. On a VIC this will need a little reworking on the screen display.

```
960 for h = 0 to i–1
970 if s%(h)<>–1 then 1010
980 n = n + 1
990 if n = 1 then print " You didn't try these
    words: " chr$(13)
1000 print w$(h),
1010 s%(h) = –1
1020 next h
1030 print chr$(13) " q Press 'RETURN' to continue. "
1040 m = 0
1050 n = 0
1060 get a$
1070 if a$ = chr$(13) then return
1080 goto 1060
```

The last part of the program is the data. I've given you several examples of the data we used. There are only two catches. First put commas, semicolons, colons, and capitals inside quotes. Second, the last data value MUST be *,*. The program looks for the two asterisks as the EOD. 2000 rem data * words – definitions (use quotes to include commas, semicolons)

```
2010 rem * also use quotes if you are using capitals
2020 rem * the last data value must be *,*
2030 data fall, " autumn, to trip "
2040 data back, opposite of front
2050 data " I ", me
2060 data yes, opposite of no
2070 data good, opposite of bad
2080 data our, belonging to us
2090 data yellow, colour of a lemon
2100 data forget, not remember
2110 data hard, opposite of soft
2120 data hand, has five fingers
2130 data stand, not sitting
2140 data band, they play music
2150 data land, ground
2160 data her, 'his' for a girl
2170 data love, opposite of hate
2180 data yard, play in the back . . ..
2190 data milk, we get ---- from cows
2200 data and, also
2210 data get, we --- milk from cows
2220 data go, do this with a green light
2230 data " Spot ", a dog's name
2240 data new, opposite of old
2250 data *,*
```

This program was set up to help Cameron with his spelling and reading. This program has applications in other areas. If your use requires giving one piece of information and requesting another then you can use Speller. French to English translation, history questions (Who sailed in 1492?), geography, biology, English (What is the plural of 'matrix'?) can all be used within this program by only changing the data. I hope it helps you with the concept and application of lists and a few other programming techniques. Maybe it will help a few other children with their spelling.

As written it should run without change on most PETs (replace the PRINT CHR$(14) with POKE 59468, 14), the VIC 20 if you follow the hints given above, omit the centering routine, and with very little work virtually any other computer running Microsoft BASIC. The program can be improved. There is no reward procedure for right answers. Graphics, sound, and colour can be added for more zip and pizazz.

Helping The Handicapped

Philip J. Honsinger
Kitchener, Ontario

The special educational needs of handicapped children can make the usage of a computer system only a dream in the child's eyes. If his/her motor control has been affected by the handicap, then even the usage of a joystick may not be possible.

In an attempt to solve one such child's problem, I have designed and built a "joystick simulator" box. This device allows the movements of a joystick to be duplicated through a grouping of large push button switches on a specialized peripheral box.

Working with the teacher from the school, we identified some specialized design considerations, such as:

- make sure the box will fit between the arms of a wheelchair
- it should not be predominately left or right handed
- the push buttons used should be quite sturdy, but easy to push down
- it should have functional & plug-in compatibility with standard joysticks so that either a joystick or the box can be used with the same software
- the distance between the switches must be enough to allow small hands to palm-push a switch without accidentally pushing other buttons
- keep the design reliably simple and inexpensive.

Now that our requirements have been established, let's move on to the design. The box is about 21.5 cm. by 28 cm. by 5 cm. (8 1/2" by 11" by 2"), and has a 2 metre (6 foot) 6-conductor cable out of the back. The cable ends in a standard joystick plug. The six wires in the cable are one for ground, one each for up / down / left / right, and one for the fire button(s).

The 64's reference manuals tell us which wire is connected to which pin in the joystick plug. The wiring layout is:

Line	Pin
Ground	8
Fire	6
Up	1
Down	2
Left	3
Right	4

First mount the switches in the box, and solder separate wires on the 64 end of the cable to each of the above pins in the plug. Run the other end of the cable through a hole in the back of the box (not the bottom), and tie a knot on the inside to act as a strain relief. Leave about 15 cm. (6 inches) of cable to work with inside the box. Strip the ends of the wires and connect them to each lug of a six-lug terminal strip inside the box.

Now solder a wire from one of the two lugs from each of the switches to ground. Solder a wire from the other lug of each switch to the corresponding wire in the cable for the function that the switch is to perform. This sets up each switch to connect the proper wire to ground when the switch is pressed. I have arranged the switches on the top of the box in the following pattern:

fire fire
up
left right
down

My prototype box was built out of wood. If it is going to be used a lot, a metal cabinet may be a good idea.

The switches themselves were obtained from a local electronics supply company. They have a high impact plastic button that protrudes about 1.5 cm. (1/2 inch) above the box top. The action of the switch is firm enough to tell the child by touch that it has been pressed, but yet is not too difficult to depress. They are momentary contact, normally open switches (that is, they are ON only when pressed, and pop back up instantly when released).

The 9-pin plug for the joystick port can be purchased at Radio-Shack, or a suitable electronics supply store. Ask for D-Sub Connectors.

I have described the switch layout that I used for my application. This was set-up to represent the 4 major compass points plus two fire buttons for symmetrical operation. A direction such as north-east may be accomplished by pushing two buttons at the same time. Taking this further, we actually have more input flexibility than a joystick allows. Any (or all) of the switches can be pressed at the same time, and this could be incorporated into your own programs, if you need this type of flexibility.

You also do not have to use push-button switches. Using the same wiring scheme with on/off toggle type switches will produce an input device that "remembers" your selections. In this case, these switches stay on until switched off.

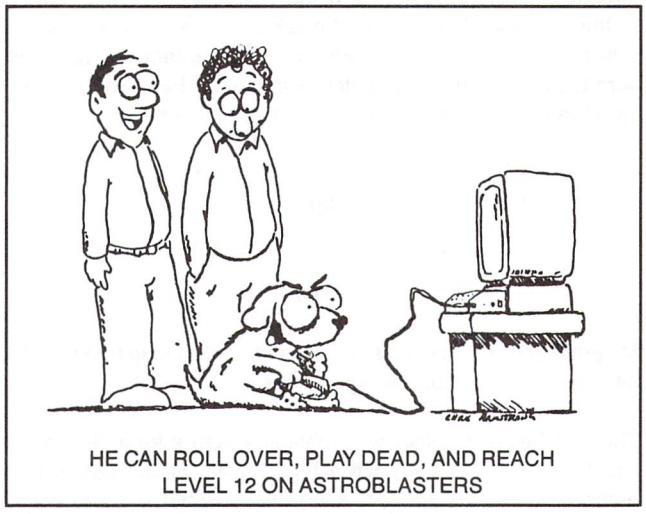
This type of an input peripheral has many uses, and can be adapted through layout choices and switch types to cover many applications. Try magnetic reed switches (closed by magnets) on doors or windows as simple security devices. Of course, these custom applications will require original software to service the input devices. Projects such as these would make good assignments for young programmers.

Getting back to the original reason that the box was built, we have provided the school system with a new tool for helping handicapped kids use computer systems. One of the best vehicles for

this is simple shoot-em-up games. They will give the student instant feedback, and keeps the experience lots of fun.

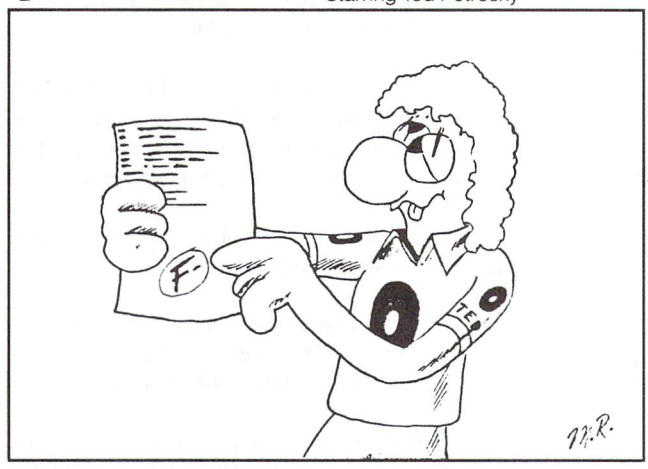
I tested the prototype box with my three year old son. He has an alphabet game which normally uses a joystick to "shoot down" letters, and also plays the ABC song at the beginning and end of each game. It is easy to play, but yet is interesting enough to give the child a good starting point with the box.

Our handicapped student at the school is in his early teens. He has Spastic Cerebral Palsy, and is both visually and physically impaired. He can not control his hand movements enough to accurately move a joystick, but his teacher thought he could bang on buttons with no problem. When he used the box for the first time, he discovered he actually could use a computer just like the other kids. He could not make out exactly which letters he was shooting down, but they were large enough to tell him if he was aiming properly. This program provides an opportunity for spatial awareness practice. Each time he would hit a letter, it would explode and he would get some sound effects. From what his teacher told me, he had never been as excited as he was that day. It made all the work worth-while.

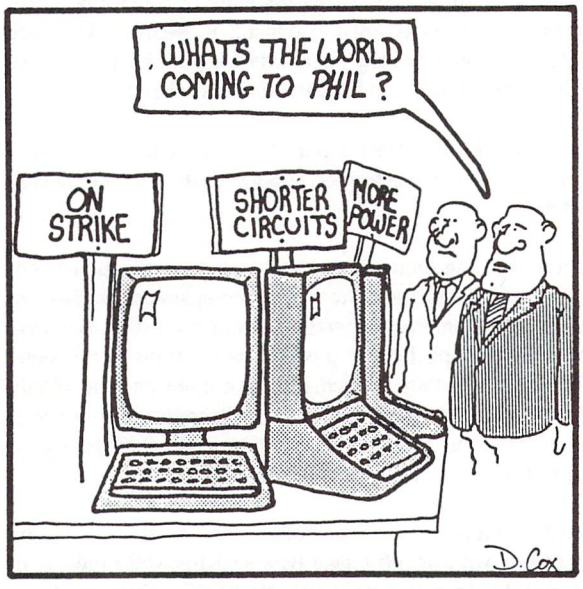


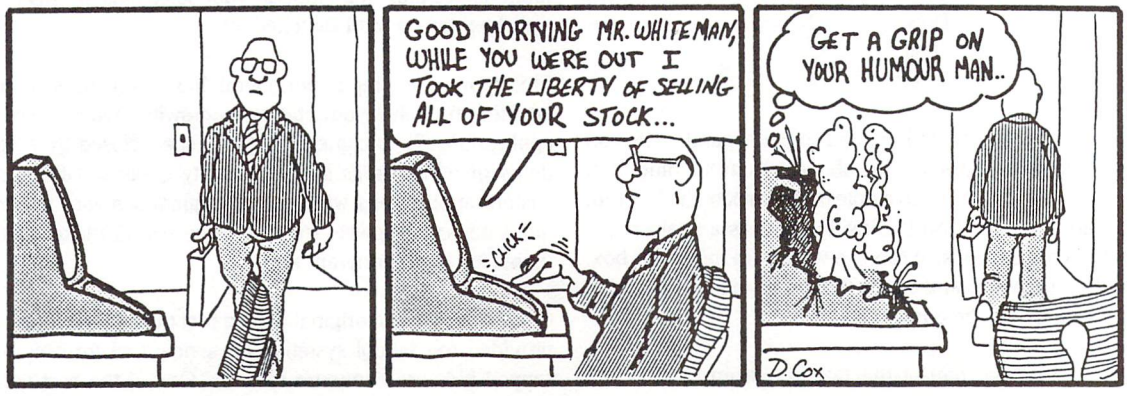
HE CAN ROLL OVER, PLAY DEAD, AND REACH LEVEL 12 ON ASTROBLASTERS

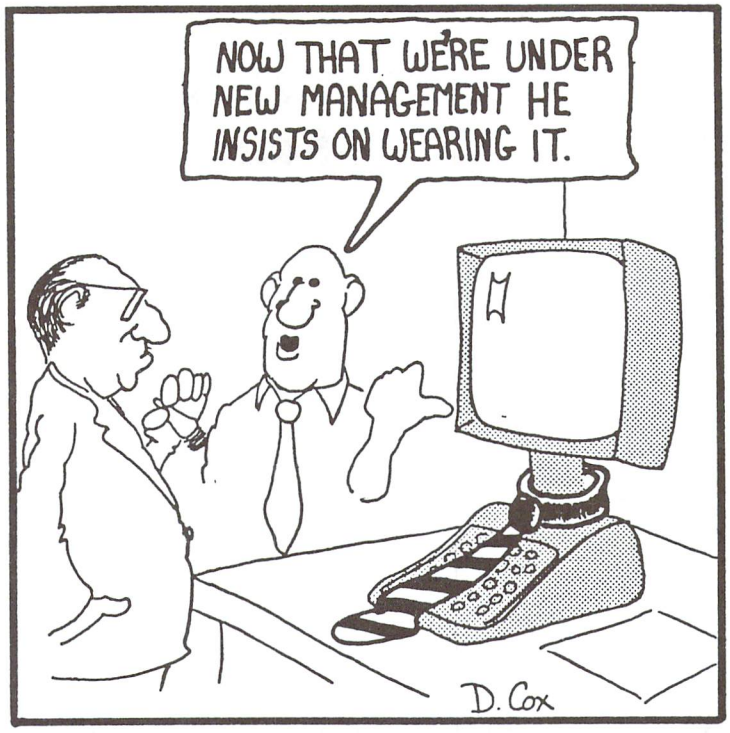
DATA-MANIA! Starring Ted Petrosky

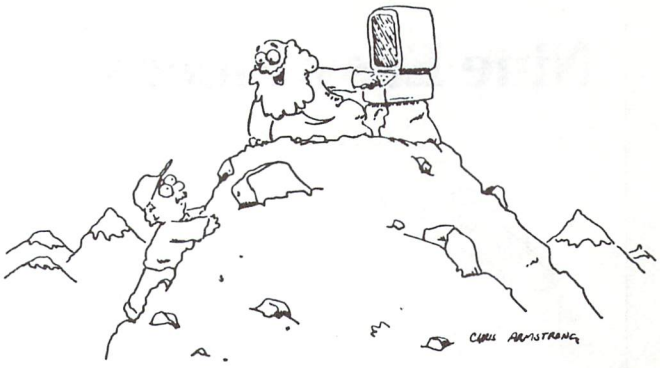


I gave my computer teacher the definition of a GOTO statement and he flunked me. So I gave him **my** definition of a GOTO statement and he kicked me out of school!









THE SECRETS OF THE UNIVERSE COME A LOT EASIER SINCE I GOT THIS THING

Nine Easy Pieces

Carl W. David
Storrs, CT

The Transition to Machine Language issue of The Transactor (Vol. 5, Issue 02), by its very appearance, indicates a desire on the part of Transactor readers to learn and become expert in machine language programming. This article presents "nine easy pieces", arranged in sequential order, which incrementally introduce absolute novices to C64 assembler language programming.

The assembler happens to be French Silk, but the reader with another assembler should have no difficulty in applying these programs to his/her own dialect. Each program is as short as I could make it.

Program 1

```
CHROUT   EQU 65490
START    EQU 49152
         LDA #'A'
NEXT     JSR CHROUT
         JMP NEXT
```

This first program is just about the shortest program possible that actually does something. It is an endless loop going from JuMPing to NEXT forever, while it sends what is in the A register, which happens to be the representation of the letter "A", to a subroutine that prints the "A" on the screen. That subroutine is called CHROUT and it starts at decimal location 65490 in ROM. It' function, as the name suggests, is to OUTput a CHaRacter.

Note: All subroutines used herein are described in the Commodore 64 Programmers Reference Guide. CHROUT is described in section B–5, on page 278. The serious machine language programmer needs this book!

Briefly, however, CHROUT sends the contents of the A register to the output device which is the screen *by default*. The BASIC "PRINT" command uses the CHROUT subroutine. Likewise, PRINT defaults to the screen. But if a CMD command is performed, PRINT will send characters to the printer, disk, etc. The point is, CHROUT sends the character no matter where it is going, and no matter where it came from. (PRINT#, INPUT prompt, etc.). For now though, we'll just be sending to the default output device; the screen.

To run the program, first assemble it. The first two lines are label assignment lines. They do not actually produce any code but rather tell the assembler to equate a specified value to a label so that we do not have to remember the value when we need to use it. This also makes it easy to modify the program since we need only change the assignment line and not every line of the program where the value may be used.

The assembler will deposit the numerical values representing the instructions into successive memory locations starting with the address specified by the START label. Essentially, the assembler puts values in memory as if we were to put them there with successive POKE commands:

```
POKE 49152, 169
POKE 49153,  65
POKE 49154,  32
POKE 49155, 210
POKE 49156, 255
POKE 49157,  76
POKE 49158,   2
POKE 49159, 192
```

Then SYS 49152 causes the computer to jump to 49152, where it finds the LoaD Accumulator instruction, and proceeds. This tiny program is doesn't have any exit so the computer must be powered off (or reset by some other means) in order to get out of the loop.

Program 2

```
GETIN    EQU 65508
CHROUT   EQU 65490
START    EQU 49152
NEXT     LDA #'A'
         JSR CHROUT
         JSR GETIN
         CMP #0
         BNE STOP
         JMP NEXT
STOP     BRK
```

Only 5 lines have been added, with one change. The first line equates the label "GETIN" to 65508 which is another subroutine in ROM that services the keyboard. (see page 283 in the 64 PRG) When GETIN is used, the Accumulator will contain a character corresponding to the ASCII (CBM ASCII) value of the key pressed, or zero if no key is being pressed.

This program, once assembled and SYS'ed to (use SYS 49152 again) will print A's forever, like last time, until you press any key. The three lines after the call to CHROUT do a test. After calling GETIN, the CoMPare command compares the Accumulator with zero. If a key is pressed the Accumulator will **not** contain zero and the Branch on Not Equal instruction will branch execution down to STOP where a BRK command will be executed. No need to turn off the computer to halt anymore.

Otherwise the branch will 'fail' and it JMPs back to NEXT. Notice NEXT has been moved up one line to the LDA instruction. Since GETIN alters the contents of the Accumulator, the value for 'A' must be re-loaded.

When the BRK instruction is reached, the program will halt. How it halts will depend if you have a machine language monitor installed. The BRK vector is an address that tells the computer where to go if a BRK command is executed. With no MLM the C64 BRK vector will be pointing at BASIC Warm Start – the screen will clear and print "READY.". If Supermon 64 or other MLM program is loaded the BRK vector will be altered to point at the MLM. When BRK is executed the screen will usually show the Register display that prints upon calling the MLM deliberately.

Program 3

```
GETIN    EQU 65508
CHROUT   EQU 65490
START    EQU 49152
NEXT     JSR GETIN
         JSR CHROUT
         CMP #13
         BEQ STOP
         JMP NEXT
STOP     BRK
```

Now, the character typed in (and found by GETIN) is echoed onto the screen (by CHROUT) until the user presses the RETURN key, represented by the value 13. BEQ is a Branch if EQual instruction, which says (after the CoMPare with 13) "if the character in the Accumulator is a Carriage Return, then Branch to STOP and execute the BRK. Otherwise, JMP back to NEXT, GETIN another character from the keyboard, etc, etc.

The BYT Directive

BYT is an assembler directive or "pseudo-op" that allows the user to place characters within a program that are not necessarily instruction codes. Here we'll use BYT to include a string as part of the program. (Note: Some assemblers use .BYT or .BYTE, and PAL requires .ASC for inserting characters as a string of letters; .BYT is used to specify values numerically as in the last line)

Program 4

```
CHROUT   EQU 65490
START    EQU 49152
         LDX #0
NEXT     LDA POINT,X
         JSR CHROUT
         CMP #13
         BEQ STOP
         INX
         JMP NEXT
STOP     BRK
POINT    EQU @
         BYT 'THIS IS A MESSAGE
ENDM     BYT #13
```

Here, the computer will write out the message "THIS IS A MESSAGE", and when it is done it will issue the BRK instruction. Instead of getting characters from the keyboard, this program gets them out of an area in memory that is just above the program. The computer translates the statement:

```
POINT EQU @
```

so that the label "POINT" equals the current address (@) with French Silk, other assemblers use *) which is immediately after the address of the BRK instruction. This is where the BYT directive will deposit the letters of the message when the program is assembled. The next BYT directive deposits a Carriage Return immediately after the message which will indicate the end.

When SYS 49152 is entered, the X register is loaded with zero. Next the Accumulator is loaded:

LDA POINT, X

. . .will load the character from an address that equals POINT plus the contents of the X register. Since the X register starts at zero and continues incrementing in value (by the INcrement X instruction) this line will successively load the letters of the message which are printed by CHROUT.

Each time a character is printed here, it is tested for the value 13. When the Accumulator loads this value, it too will be printed as a Carriage Return. If you don't want a Carriage Return the test must be performed before CHROUT is called.

The only thing wrong now is that BRK instruction. When the program gets there, the screen is cleared. Change the BRK to an "RTS". RTS means ReTurn from Subroutine. This will print "READY." without clearing the screen if you don't have a machine language monitor.

Program 5

```
GETIN    EQU 65508
CHROUT   EQU 65490
START    EQU 49152
READ     JSR  GETIN
         CMP #0
         BEQ READ
         JSR  CHROUT
         CMP #13
         BEQ STOP
         INX
         JMP READ
STOP     RTS
```

Here is a small program that is much like Program 3 with some minor rearrangements. It will echo what you type on the screen until you type RETURN. There is absolutely nothing new here except that the routine waits until a key is actually pressed before attempting to print it. Program 3 would call CHROUT whether a key was pressed or not. But because "no key" results in zero, CHROUT would try to print a CHR$(0) which has no effect on the screen.

Now we'll add to it.

Program 6

```
PLOT     EQU 65520
GETIN    EQU 65508
CHROUT   EQU 65490
START    EQU 49152
         LDX #23
         LDY #24
         CLC
         JSR  PLOT
READ     JSR  GETIN
         CMP #0
         BEQ READ
         JSR  CHROUT
         CMP #13
         BEQ STOP
         INX
         JMP READ
STOP     RTS
```

Just for fun we've added a call to the subroutine "PLOT" at location 65520 in ROM (see page 290 of 64 PRG). PLOT will allow the cursor to be positioned to any row and column on the screen. This program positions to row 23 (in the X register) and column 24 (in the Y register). Feel free to change these to any value. The CLC, CLear Carry, is required by the PLOT routine before calling it.

Program 7

```
STOP     EQU 65505
CHROUT   EQU 65490
START    EQU 49152
         LDX #0
         LDY #24
         CLC
         JSR  PLOT
READ     JSR  GETIN
         CMP #0
         BEQ READ
         JSR  CHROUT
         JSR  STOP
         BNE READ
         RTS
```

This is merely a modification to the last program. Instead of halting with a Carriage Return the routine will now end when you press the STOP key. This is done with a call to the subroutine 'STOP' at 65505 in ROM (see page 301 of 64 PRG). This routine does not actually do anything except set up a situation that can then be tested. If the STOP button is not pressed, the BNE command will cause execution to go

back to READ the keyboard.

Program 8

```
STOP    EQU 65505
CHROUT  EQU 65490
START   EQU 49152
        LDX #0
        LDY #24
        CLC
        JSR PLOT
READ    JSR GETIN
        CMP #0
        BEQ READ
        CLC
        ADC #1
        JSR CHROUT
        JSR STOP
        BNE READ
        RTS
```

This program is a "gibberish generator". After getting a character, 1 is added to its value with the ADd with Carry command. When you type A you get B, type B you get C, etc.

Program 9

```
READST  EQU 65463
CLOSE   EQU 65475
CLALL   EQU 65511
OPEN    EQU 65472
STOP    EQU 65505
CHKIN   EQU 65478
CHRIN   EQU 65487
SETNAM  EQU 65469
SETLFS  EQU 65466
GETIN   EQU 65508
CHROUT  EQU 65490
START   EQU 49152
STEP    LDA COUNT
        LDY #>FILE
        LDX #<FILE
        JSR SETNAM
        LDA #3
        LDX #8
        LDY #3
        JSR SETLFS
        JSR OPEN
        BCS ERROR
        LDX #3
        JSR CHKIN
READ    JSR CHRIN
        JSR CHROUT
        JSR READST
        AND #64
        BEQ READ
L1      JSR STOP
        BNE L1
        LDA #3
        JSR CLOSE
        JSR CLALL
        RTS
ERROR   LDA #'E'
        JSR CHROUT
        JMP L1
FILE    BYT '0:NINTH.SRC,S,R
COUNT   BYT #15
```

This last program isn't so simple. The first part imitates an:

```
OPEN 3, 8, 3, "0:NINTH.SRC,S,R"
```

as if it were done in BASIC (see page 289 of 64 PRG). This requires that we first set up a name for the file, and then set up a logical number for that named file.

```
LDY #>FILE
```

loads the Y register with the high byte part of the address of FILE, while:

```
LDX #<FILE
```

loads the X register with the low byte part of the address of FILE. "3,8,3" is obvious from the code. READST (see page 292 of 64 PRG) senses if there was any problem with the OPEN, and raises the carry flag if there was. Branch on Carry Set tests for this error condition. CHRIN (see page 277 of 64 PRG) imitates a GET#3 as if done by BASIC. After the last read from disk, READST will return with a value of 64 indicating end of file. CLOSE is a subroutine that acts just like the BASIC CLOSE, and CLALL CLears ALL the I/O channels so that everything is back to normal; input from the keyboard and output to screen.

And that's all there is to that! Experiments based on these nine easy pieces have shown novices become experts in no time. Machine Language is no different than any other language, it just take practice like any other language. Best of luck on your 10th piece!

Editor's Note:

Carl may not use the assembler you're familiar with, and if that's the case this will be a good exercise for you. It won't be always that you read code written in a familiar dialect. Becoming familiar is another skill you should acquire.

Interrupt Driven Code On The Commodore 64

Bruno Degazio
Toronto, Ont.

This article will demonstrate some of the features of interrupt driven programming that make it the versatile and efficient tool it is in all sorts of real time control applications. In particular, a set of assembly language routines for the Commodore 64 will, I hope, lead you to consider some applications of your own.

The C64 is a pretty good machine from the standpoint of hardware interrupts since CBM has thoughtfully provided us with a full box of interesting electronic gadgets inside that brown plastic case. The CIA chips that handle all input/output processing in the 64 are very advanced and versatile devices that are not used to full advantage by many programmers. Specifically, CIA chip #1 is used by the operating system to set up a real time clock, scan the keyboard for possible user input, and handle various other "housekeeping" duties. This is done on a regular and frequent basis by means of the timer interrupt facilities on the CIA chip, which the operating system sets up to clock 1/60th second intervals. That is, every sixtieth of a second one of the timers in CIA #1 "calls" an interrupt, which causes the microprocessor to stop its current task and begin executing the operating system routines whose start address is stored in $FFFE and $FFFF at the top of ROM.

Now, if we want a job done on a regular basis without necessarily requiring a complete self-running program of its own, we might consider replacing the housekeeping routines with routines of our own. One in particular, that I thought would drive my friends crazy, would be a routine to sound a beep with every keystroke on my C64. Since I knew the operating system scanned the keyboard 60 times per second, it seemed a little inelegant, to say the least, to set up my own program loop to check the keyboard and emit a beep if a key was pressed.

Therefore, I went through the ROM system routines beginning with $FFFE. The address contained there is $FF48. An IRQ causes an indirect jump through $FFFE which effectively jumps to $FF48. The routine at $FF48 does some stack handling and then another indirect jump through $0314 which contains the address $EA31. When it gets there, the housekeeping duties begin.

When the various levels of indirection begin to add up it gets a little confusing so I'll explain that again. The timer interrupt (or IRQ) causes the processor to stop what its doing and begin executing instructions at the address contained in $FFFE and $FFFF - that action is built into the hardware of the 6510 microprocessor. All this does little more than cause a jump to $FF48 because that's the address it found in $FFFE/F. $FF48 is the main IRQ entry point. After a couple of preliminary commands, it causes another indirect jump through $0314. This is no different than the first indirect jump, except that it's in RAM instead of ROM, and the address found there is $EA31. The code at $EA31 is where the actual work starts.

Why all the jumps from ROM to ROM to RAM and back to ROM? It's for our own convenience really. The first jump uses $FFFE/F because the microprocessor is built that way. This is known as a vector. No matter what piece of hardware it's designed into, the processor will look there and use the contents as the vector which it will jump to for an IRQ - $FF48 in the 64. Why $FF48? In most Commodore machines the first few bytes of the IRQ routines are identical. This "common" part of the IRQ routine has been stored here in the 64 and it looks like Commodore will try to maintain $FF48 as the start address of the IRQ routines in future machines, which is also convenient.

The second jump happens just after $FF48. It jumps through RAM because unlike ROM, RAM can be changed. When the 64 is turned on, the power up routines load the address

$EA31 into locations $0314 and $0315. Since this address can be altered to any one we choose, we can have the timer interrupt drive our own custom routines instead of the ROM routines. Of course, these routines must be firmly in place before we change the "RAM vector" or the machine will most likely crash.

Most of the time we will not want to re-write all the little machine language routines that handle the ever vital house-keeping chores. So all we need to do is have the processor jump to those routines (ie. at $EA31) **after** it has taken care of our own work. This technique is known as *pre-interrupt* coding, for obvious reasons, and can be used for all kinds of interesting applications.

A few of the little tricks necessary to handle this technique are illustrated below by means of the keyboard beeper program. If you have an assembler aor machine language monitor you can try these out for yourself. Good place to put the program are in the cassette buffer at $033C and in the large normally unused area beginning at $C000.

Note: The 4K block of RAM starting at $C000 is a revolutionary feature introduced by the 64. Essentially, it is an "activity free zone". If there is no other program requiring use of this space (like a cartridge) you can be sure that nothing else in the machine will touch it. It is yours for experimenting, storing data, or whatever you desire with absolutely no side effects.

Notice in the initialization routine titled "PRE", the instruction SEI is executed first. This ensures that an interrupt will not occur while the program is changing the contents of the vector. Imagine the potential disaster if an interrupt were to occur between changing the low and high bytes – only half of the address would be reliable. The other half has a 1 in 256 chance of being correct, but we can't take that chance. SEI means SEt Interrupt mask which disables the timer interrupt. Later we will re-enable it.

After changing the vector, this little routine will then set up the correct registers in the SID chip to provide a pleasant beep later when we tell it to (we hope). With the envelope, frequency, and volume controls adjusted, the routine ends with a CLI (CLear Interrupt mask) and an RTS. The timer interrupts are now re-enabled.

The RESTORE routine is very similar to PRE, except in reverse. It first turns off the SID chip and then replaces the contents of $0314 and $0315 with their original values.

```
PRE     SEI
        LDA #<SERVICE
        STA  $0314  ;replace low byte of system routine
                    with low byte of "SERVICE" routine
        LDA #>SERVICE
        STA  $0315  ;replace high byte of system routine
                    with high byte of "SERVICE"
        LDA #$00
        STA  $D405  ;SID envelope generator for voice 1 -
                    attack and decay values
        LDA #$F8
        STA  $D406  ;EG1 sustain and release values
        STA  $D401  ;SID oscillator 1 frequency value
        LDA #$0F
        STA  $D418  ;volume control set to maximum
        CLI
        RTS
;
RESTORE SEI
        LDA #$00
        STA  #D418  ;set volume to zero
        LDA #$31
        STA  #0314  ;reset irq low byte to $31
        LDA #$EA
        STA  #0315  ;reset irq high byte to $EA
        CLI
        RTS
```

The BEEP routine is fairly straightforward. Location $00C5, according to the Commodore 64 Programmers Reference Guide, holds the value of the current key pressed and contains zero if no key is pressed. This value is used both as a flag to turn the beep on or off, and as a frequency value to give each key a unique pitch when pressed.

```
BEEP    LDA $C5     ;accumulator gets contents of $C5 -
                    the ASCII value of the current key
                    pressed
        AND#$BF     ;is it an ASCII character or a zero?
        BEQ OFF     ;a zero - no key currently pressed
        STA $D401   ;otherwise store the value in SID's
                    frequency register
        LDA #$11
        STA $D404   ;turn SID gate on
        RTS         ;and return to SERVICE routine
;
OFF     LDA #$10
        STA $D404   ;turn SID gate off
        RTS         ;and return to SERVICE routine
```

The SERVICE routine is typical of pre-interrupt handlers. If first saves all the registers in a safe place (on the STACK) through use of the PHA (PusH Accumulator on the stack) instruction, and then calls the BEEP subroutine. The internal registers are then restored through use of the PLA (PulL the top stack item into the Accumulator) instruction so that it can continue with whatever job it was doing when it was

interrupted in the first place. Note that the registers are restored in the reverse order from the way they were stored. Also note that all three internal registers are stored and restored even though the little BEEP routine used only one, the accumulator. This is purely in the interest of generality and to allow the routine to be easily adapted to your own jobs. The final important note is that the pre-interrupt routine does not end with an RTI (ReTurn from Interrupt) instruction but transfers control to the normal ROM interrupt handler routine at $EA31, which you may trust to preserve carefully all internal registers before going about its chores. This routine does eventually end with an RTI, transferring control back to the processor's main task, a BASIC program for instance.

```
SERVICE  PHA          ;save all processor
         TYA          ;registers on stack
         PHA
         TXA
         PHA
         JSR  BEEP
         PLA          ;restore all
         TAX          ;processor registers
         PLA          ;in reverse order
         TAY
         PLA
         JMP $EA31    ;jump to normal IRQ service rou-
                       tines
```

Thsi routine is fully relocatable except for the placement of the interrupt vector at locations $0314 and $0315. The bytes put in these locations by the PRE routine **must** be the low and high bytes, respectively, of the start address of the SERVICE routine. If the program was assembled at address $C000, the address of the SERVICE routine will be $C046. However, the assembler will calculate this for you no matter where it ends up and the program can be invoked from BASIC with a simple

```
SYS 49152
```

The program is disabled by RESTORE using:

```
SYS 49183
```

The BEEPER program though fun and illustrative of the fascination of "background" programming with interrupts, is not particularly useful. However, the PRE and RESTORE routines are quite general and can be used to insert any sort of processing task you wish. One rather useful application would be to send data to the printer from a disk file (a "Spooler" routine) or collect data at a moderate rate from the User Port.

The task need not be as short and simple as the BEEP routine either. Even at sixty time per second, the processor has lots of time between calls to execute other tasks, such as a BASIC program that might be running. As the pre-interrupt routine gets longer and longer however, you will notice the "foreground" task slow down more and more. At a certain point, the background task will require all the processor time; just as the interrupt is finished, another interrupt occurs. This boundary is pretty high though – the interrupt routine can be a total of about 6000 instructions before you reach it. So you can see there is really quite a bit of room to insert your own code.

Editor's Note:

The SERVICE routine that Bruno presents here is much more complete than it needs to be. Bruno saves the A, X, and Y register before the JSR to BEEP and restores them afterward. This is not necessary and will not affect the general behaviour of your computer.

When an IRQ occurs, these registers are all stored for you by the code just previous to the indirect JMP through $0314. They are restored for you in the service routine somewhere beyond $EA31. When you insert a pre-interrupt routine, you need **not** be concerned what values are left in these registers before transferring execution back to $EA31. They may contain any values since the housekeeping routines will not use them in any way. Therefore, the routine called SERVICE could become:

```
SERVICE  JSR  BEEP
         JMP $EA31
```

with no nasty side effects. In fact, the SERVICE routine could be eliminated by merely changing the routine "PRE" to swap in the low and high bytes of BEEP instead of SERVICE, and change the RTS to read JMP $EA31.

Bruno's method may not be all too wasteful though. . . there is one situation where you might want to push and pull all the registers before reaching $EA31. Some I/O registers (like in the 6520/22/26 Interface Adapters and ACIA communications chips) are reset upon read. Inotherwords, their contents are altered just because they have been read, or "un-loaded". This is a handy feature when speed is of the essence – MOS Technology has deliberately and carefully built this in. But if two pre-interrupt routines will require information from this type of register, it may be wise to store the registers for future use by upcoming code. This situation happens rarely at best and usually with communications programs. So omitting the extra stack handling is a safe procedure in most cases. – M.Ed.

AUTOSWAP: A Multiprocessing System on the C64

Chris Zamara, Technical Editor

Partition Switching

In the last issue, we printed a program by Daniel Bingamon called Quadra 64. Quadra 64 allowed program development in one of four independent memory partitions, and used "wedge" commands to switch among partitions. Quadra 64 keeps all BASIC variables local to each partition, since text and variable pointers are switched when a new partition is entered.

If you wish to have indpendent partitions, and will always be switching from one to another in direct mode (not while running a program), then Quadra 64 will meet your needs. Sometimes, though, partitioning is not enough. If, for example, you wish to switch from one program while it is running, enter a new program, then switch back to the first one and have it continue where it was interrupted, what is needed is independent *environments*.

Environment Switching

As far as the operating system in the 64 is concerned, there is more to life than BASIC text and variable pointers. There are the pointers for CHRGET, current line being executed, current variable address, expression evaluation, etc. There are flags to indicate quote mode, direct mode, key repeat, cursor flash, and dozens more. There are hundreds of bytes containing things like character colour, cursor position, number of inserts outstanding, number of files open, and miscellaneous temporary storage used by various ROM routines. In addition, we can't forget the stack, screen memory, colour memory, or the input buffer. In other words, lots of stuff.

What this means is that an environment-switching system must have independent sets of all of these bytes stored somewhere in memory. When a switch to an alternate environment is to take place, the current state of the variables must be stored, and another environment's variables brought in to take their place. This is the approach taken by the accompanying program, "AUTOSWAP".

To be useful, the environment-switching technique should be controlled by direct keystrokes, and should take place during an IRQ (hardware Interrupt ReQuest). AUTOSWAP is set up so that there are two environments, which may be alternately selected by hitting the F3 key at any time. When the environment switches, it's like getting up from behind your computer and going to another one (with less memory). On your new computer, you are free to do whatever you like, without regard for what was going on over at computer 1. When you hit the F3 key again to switch back, your first computer hasn't changed at all, unless you did some POKEs to hardware registers while in the second environment. The screen is the same, the cursor is in the same place, programs continue from where they were interrupted, and of course, no variables have been clobbered.

With AUTOSWAP, you could enter or load a program while in environment 1, then run it. While it is running you could switch to environment 2, load another program and run it. Now, by pressing the F3 key – swapping environments – you can alternate execution between the two routines. The logical extension of this poligamous processing is automatic swapping every fraction of a second, excluding the screen swap. This amounts to multiprocessing, something which micros are very infrequently called upon to do. But it *is* possible, and "AUTOSWAP" makes provision for it (it isn't called auto-swap for nothing). Mutiprocessing is sort of like feeding steroids to your 64: it'll work twice as hard for the same money.

Multiprocessing With AUTOSWAP

Multiprocessing is accomplished by pressing the "auto mode" key, F5. When F5 is struck, automatic environment switching begins to take place every three interrupts. This amounts to a swap about every 0.05 seconds, but the value is variable. A swap every interrupt makes the system very inefficient, since there is too much overhead: most of the CPU's time is spent just switching environments. Swapping too infrequently, on the other hand, makes execution of each program very irregular: run for a while, stop. . ..run. . ..stop. . . – you get the idea. As with any multiprocessing system , there is a trade-off between system overhead and actual processing. That, in fact, is AUTOSWAP's alias: "An Unavoidable Trade-Off: System Work And Processing". (It's really an acronym for: A Useful Technique Of Switching With Alternating Parameters).

To run two programs at once, first set them up as in the above example: run one at a time so that either one may be selected with

the swap key, F3. Then press F5 to engage multiprocess mode. The screen won't be swapped back and forth between environments, but you may look at either environment's screen by pressing the swap key. You can also work in direct mode in either environment while in multiprocess mode – this allows you to edit one program while running another in the background. You may look in on the running program at any time by selecting its screen with the swap key, and then switch back. The F5 key acts as a toggle, so pressing it again turns off multiprocess mode and leaves the system in the environment currently being viewed.

You can also control AUTOSWAP from within a program, without having to press the function keys. The location $02F1, or 753 decimal, is used to simulate the depressing of one of the function keys. Just poke the key code of the desired function key into this location, and that key's function will be performed. Specifically, to simulate F3, the swap key:

```
POKE 753,5
```

To toggle multiprocess mode, performed by F5:

```
POKE 753,6
```

If you wish, you can simulate F1, the "save out" key (used in installation – explained later) with:

```
POKE 753,4
```

Here's a table for clarification:

Function	Direct mode	Program Mode
Save environment 1	F1	POKE 753,4
Switch environments (switches screens in m/p mode)	F3	POKE 753,5
Toggle multiprocess mode	F5	POKE 753,6

The Methodology and the Problems

The principal of operation has already been explained: swap sets of parameters alternately during interrupts. Accomplishing that deviously simple-sounding aim is a little bit tricky. Imagine that the CPU is courting two girlfriends (environments) at the same time; if either one finds out about the other, there's gonna be trouble.

First of all, as mentioned previously, there are a lot of variables to swap. That's not the hard part, since most of them live in a contigous chunk of memory from 0000 to 0400 (hex). Furthermore, the screen resides at 0400, and a second screen is set up at 0800; it's easy to point the operating system and the video chip at either screen block, so the screen memory needn't be physically swapped like the other parameters. Besides that, the only other thing to swap is colour memory, but that's another story and will be addressed later.

So what's the problem? Well, as many have probably noted, the stack makes its home in page one, which is included in our zone to be swapped. You probably know that the stack doesn't like to be messed with. In fact, terrible things often happen to those ill-fated souls who inadvertently clobber the stack. The stack holds the vital return addresses for subroutines and interrupts, as well as providing a temporary storage area for any code that wants to use it. Thus, we must include the stack with the parameters to be switched; each environment would otherwise declare, "this stack ain't big enough for the two of us", and then proceed to crash in some strange way. The stack has to be saved and restored along with the other parameters, and the same holds true for the stack pointer, which is actually a register internal to the CPU. An extra bit of code in AUTOSWAP takes care of that.

We've ascertained that strange things happen when unorthodox stack manipulation is performed, and this situation is no exception. The original version of AUTOSWAP used a general-purpose memory transfer subroutine to swap memory in and out of pages 0 through 4 (which includes the stack in page 1). The code looked flawless, but it didn't quite work. Your confused and frustrated author, scratching his head, had to eventually turn to a programmer for help when all else failed (thanks, Rico). A few hours of mutual head-scratching finally revealed the obscure bug. When the entire stack, including the stack pointer, is replaced, the effect is somewhat like pulling the rug out from under the system's feet. This is basically what was happening: The program would call the subroutine to move new memory into the stack. When the subroutine ended, it attempted to return to the calling address with an RTS instruction, in the usual fashion. RTS would go about its business, pulling the return address from the stack – which now had a different return address in it. Since the new stack represented the state of the system at the previous swap, the program would return to where it was at that time; a crash wouldn't occur, but things just wouldn't work quite right when switching during BASIC program execution. It sounds confusing, and it is. AUTOSWAP had to be modified so that it didn't use a subroutine for memory transferring.

Explanation of that little oddity will have to serve as apology for the bizzarre looking code which resulted: it's self-modifying, and does not follow an easily traceable path of execution. It may be educational, however, to look at the way it avoids using JSR or RTS instructions to execute a subroutine.

The section of code which transfers a range of memory acts as a subroutine, and is passed its paramters (source and destination start pages) through the A and X registers. Instead of ending with an RTS, though, it ends with a JMP instruction whose argument is set up before the "subroutine" is executed. Thus, the calling sequence involves setting up the return address of the final JMP instruction, then executing a JMP to the start of the routine. When the routine is finished, it just jumps back to where it was told. This simulates what happens with JSR and RTS instructions, only the stack is not utilized this way. When the transfer subroutine is used for purposes other than filling in the lower four pages, the calling routine replaces the JMP instruction with an RTS and uses it like a normal subroutine, then changes the RTS back to a JMP.

Another problem that had to be worked around was the use of indirect addressing through zero page registers. All of zero page is written over during the transfer process, so registers in that area can't be used as pointers. Again, self-modifying code had to be used to point to source and destination addresses for memory transferring.

In theory, switching in and out the variables will allow multiprocessing. But, as usual, theory and practice do not see eye to eye. The main problem with multiprocessing using this technique shows up when it comes to editing in direct mode while also in multiprocess mode. Some characters typed on the keyboard will go to the screen you see, but others will go to the other environment, and will only show up when you switch over using the F3 key. Not good. What we need is a local keyboard as well as a local screen for each environment. In order to do that, a bit of mucking about with the kernal (operating system) is in order.

Luckily, the C64 has the unusual capability of allowing ROM and RAM to exist in the same address space, and allowing either to be selected by software. This capability can be used to change the operating system in ROM: just copy the existing ROM into the underlying RAM, select the RAM and disable the ROM, then make changes to the RAM. With this technique we can change the operating system depending upon which environment we are in and which screen we are viewing.

The problem with patching into the ROMs in this way is that the resulting code may not be compatible with all of the different ROM releases (there have been three that I know about for the 64 to date). AUTOSWAP works with the most common ROMs, not the early ones or the very late ones. I haven't tested it with other ROM versions, but if the program doesn't work on your machine, use a machine language monitor to check your ROMs against the ROM patch in the source listing (listing 2). The code at lines 4470–4540 should appear in the ROM starting at $EB34. You serious hackers out there should be able to find the equivalent patch locations for different ROM versions. We'd be glad to print these as a follow up in a future issue. If you do have different ROMs, you can still use AUTOSWAP: just delete lines 1730 to 1760 in the source listing, or make the indicated change in the basic loader program (listing 1). This will disable the ROM patches; AUTOSWAP will still work, but typing in direct mode while in multiprocess mode will be problematic.

To explain how each environment is given its own keyboard buffer, first consider all the different combinations of environments and screens:

Environment 1, screen 1
Environment 1, screen 2
Environment 2, screen 1
Environment 2, screen 2

To give a local keyboard to each environment, AUTOSWAP uses the following logic. If the environment and the screen are the same, then characters are sent to the keyboard buffer at $0277 as usual. If, however, the processor is in one environment, and the screen being viewed is that of the other, then the characters are sent to the *image* of the keyboard buffer in the storage area for the processor's environment. For example, suppose that the CPU is in environment 1, meaning that all the environment 1 parameters are occupying pages zero to four. At the same time, the screen being viewed (and thus the keyboard to be enabled) corresponds to the second environment. In this case, the interrupt routine, instead of putting the characters from the keyboard into the regular keyboard buffer, puts them into the storage area for environment 2. The next time the swap takes place, the storage area for environment 2 will be moved into pages 0–4, replacing the keyboard buffer. Thus, all keystrokes that took place while the CPU was in the non-visible environment will appear when it switches back to the visible one. It's pretty tricky, and again, makes the program a bit difficult to visualize, but this is a non-typical application, so programming must be done accordingly.

That's roughly how localized keyboards are accomplished, although there are a few more details involved which may be seen in the source code (listing 2). With the keyboard problem taken care of, it should work fine, right? Unfortunately, there are a few more problems which have to be solved to make AUTOSWAP useful.

Besides localizing the keyboard buffer, the STOP key must be handled independently as well. If a program is being worked with in environment 1 while another one is simultaneously running in 2, pressing STOP should only affect the program in environment 1. This is taken care of by using the "check STOP key" vector to trap the stop, and ignore it unless the environment and screen are the same. A minor problem, easily fixed. It also means, though, that the new stop key code is being continuously executed; changing AUTOSWAP after it has been initialized could cause a system crash. Make sure to disable AUTOSWAP with a RUNSTOP/RESTORE before doing any playing about with the code (like re-assembling new source).

Another problem that sadly has no easy fix is colour memory. Ideally, colour memory would be local to each environment just like screen memory. That would mean that switching environments, changing all the character colours on the screen, and switching back, would change nothing. Unfortunately, implementing such a scheme is a problem because colour memory must always start at the same address, $D800. The VIC video chip and the operating system in ROM both assume it to be there. The ROMs we could change, but the chip's memory addressing we couldn't – short of performing some major surgery to the circuitry inside the 64 (put down your soldering iron; we're not going to do it). So what to do about colour memory?

Well, colour memory is not that important – AUTOSWAP will usually be used when editing and debugging programs, or for other primarily text applications where only one character colour is desired. We can't just ignore colour memory, however, because whenever the operating system clears a line of text on the screen, it fills the corresponding line in colour memory with the background colour. This behaviour varies with different ROM releases: If you can clear the screen on your machine, then see the effects of POKEing to screen memory, the problem doesn't exist, and you

can get rid of lines 1840 to 1870 in the source code. With most 64s though, the free-of-charge colour change means that a program running in environment 2 could interfere with screen 1: whenever the screen scrolls (which could happen any time a line is printed), the bottom line is cleared before the new line is printed. Since screen 1 and screen 2 both share the same colour memory, characters would seem to magically disappear wherever the colour memory was set to background colour.

The solution? Well. . . ahem . . . I don't have a very good one. Barring massive ROM rewrites and prodigious memory transfers every swap, there doesn't seem to be a practical solution that will give totally local colour memory to each environment. So I cheated, and just changed the ROM code that puts the backgound colour into colour memory. Changing two bytes makes it put in the current character colour instead. It makes the system workable, as long as you stick to one character colour. Using a different colour in each environment can gives unexpected results.

Installing AUTOSWAP in the System

With most programs, installation is simple: just LOAD and RUN. AUTOSWAP, on the other hand, must set up the second environment so that it has something to switch to. This is accomplished by having two entry points in the program; the main entry point intializes AUTOSWAP, and the entry point three bytes later sets up environment 2. To install AUTOSWAP, load the object code into memory or run the BASIC loader program in listing 1. Type SYS 49152, and then NEW (make sure the loader program has been previously saved). This initializes AUTOSWAP itself, but you can't switch to another environment yet because no image exists in memory. Now press the F1 key. This is a special key used for installation only, and simply saves the current environment (contents of pages 0-4) out to its image in high memory, without bringing in a new environment. Now you can set up the new environment by typing SYS 49155. The screen should clear, and The system will now be totally operational. You are in environment 2 at this point; enter NEW before you begin programming here. To recap:

1) RUN loader (listing 1) or LOAD object created from source in listing 2
2) SYS 49152
3) NEW
4) press F1 (or POKE 753,4)
5) SYS 49155
6) NEW

Installation is now complete: swap environments with F3 or enable multiprocess mode with F5. Use F3 to change screens while in multiprocess mode.

Applications

What's AUTOSWAP good for? Well, once more, the old cliché, "you're only limited by your imagination" applies. But if your imagination needs a little help, here's a few possible applications.

Two programmers could compete against each other by programming a sprite as a killer robot, using common machine language subroutines for firing, "radar", etc. Each program would be in BASIC and live in one of the environments. Multiprocess mode would be used so that both programs could run simultaneously, and the sprites could be watched in action on the screen. To enhance the game, additional BASIC keywords could be supplied, and a "Robot Control Language" developed. After each robot was supplied with the logic it needed for seeking and destroying, both players could just sit back and watch their creations battle it out. It's something I've always wanted to do, but never had a way to run 2 programs at the same time. Now I do.

Controlling AUTOSWAP from within programs as previously explained opens up all kinds of possibilities. For example, a main program in environment 1 could call a subroutine in environment 2 by switching environments (with POKE 753,5). The subroutine would switch back when it was finished by executing the same POKE to swap again. The advantage of performing subroutines this way is twofold: first of all, all variables will be local, so any program could be used as a subroutine without worrying about conflicts with the main program. Second, the screen would be local, so that the subroutine could be used to introduce a menu, help screen, spreadsheet, note-pad, etc., without disturbing the original screen. Additionally, multiprocess mode may be enabled or disabled via program control, from either environment (with POKE 753,6). This feature could be used to execute a background subroutine; on completion, the subroutine could turn off multiprocess mode, and switch back to the main program.

Background processing. You can edit or enter a program while another one is running in the alternate environment. This may be useful, for example if you're running a long sort. While you're waiting for the sort, you want to test out a short program, or look at some disk files, or something like that. Normally, you'd have to wait for the sort to finish because you don't want to have to stop it and start it all over again. If you had the presence of mind to install AUTOSWAP before running the sort, however, it's easy: hit the multiprocess key (F5), and then the swap key to view the alternate environment. Do whatever you want there, and the sort will continue running while you do. You can later switch back to the sort screen if you wish, and shut off multiprocessing to bring it back up to full speed.

And of course, AUTOSWAP can be used for the same reason as Quadra 64, to develop programs in separate partitions. AUTOSWAP has only two such partitions, but that is usually all that is required. Just over 16K is available in each partition. Of course, there's no reason that AUTOSWAP couldn't be modified to use any number of environments. The idea remains the same, but additional logic would have to be coded to select one of N environments. In fact, AUTOSWAP started out life as a 4-environment system, but was pared down to 2 when it became too cumbersome, and would have been too large to print in the magazine.

Limitations

Well, it all sounds wonderful to have a computer split in two, but in reality there are a few flaws in this theoretical ideal. The colour memory problem has already been discussed, but there is also the matter of the I/O pages. The video and sound chip, for example, are global to both environments. That means if the border colour is changed in environment 1, it will also change in 2. Likewise, high-res mode, sound effects, and sprites are all common to both environments. This can be used to advantage, as with the killer robot application, but it generally keeps the environments less isolated from each other than is sometimes desired.

Another problem is the obvious physical limitation of having only one screen, keyboard, and serial port. If a program is background processing and requires keyboard input at some point, it will have to wait until you swap into its environment to supply it. And I don't even want to think about what happens when two programs try to access a disk file at the same time. But I suppose if what you really want is two computers, you would be reading the classified ads instead of this article.

Enhancements

As with many programs I present in the magazine, AUTOSWAP is just a bare bones system. The idea of presenting a program in the pages of Transactor is to expose techniques, and to supply a program which may be used as it stands or modified by the reader for his own needs. As it stands, AUTOSWAP is fairly long to type in (especially the source), but it could grow a whole lot. Here are a few possible improvements that a reasonably proficient programmer could add:

- A priority could be assigned to each environment, so that CPU time could be distributed unevenly between them. For example, an unimportant program running in the background (It may just check the time and act as an alarm clock) could be swapped in every 100 interrupts, for a period of 2 interrupts.
- More than 2 environments could be supported.
- a keyword or a "wedge" could be added to BASIC which would simultaneously RUN programs in all environments.
- The number of environments, and the partition size of each environment could be selected from a start-up menu
- The installation procedure could be simplified
- What about that "Robot Control Language" (RCL)? I'd love to hear from anyone completing such an application - it would make a great article!
- An improved colour memory scheme could be implemented

Usage Notes

Using AUTOSWAP becomes quite natural after a while, since it doesn't change the basic operating characteristics of the machine. There are however a few points you should keep in mind when programming in an AUTOSWAP-equipped system.

- In environment 1, screen memory starts at it's usual address, $0400 or 1024 decimal. In environment 2, it starts at $0800 or 2048. You can use this fact to communicate between screens, i.e. you could POKE into screen 2 while looking at screen 1.

- BASIC starts at $0C00 (3072) in environment 1, and at $4E00 (19968) in environment 2 (it normally starts at $0800). Any pure BASIC program may be loaded and run normally, but some programs with built-in machine language expect to load into $0800 and will die if loaded and run with AUTOSWAP in place.

- Any programs which use the function keys will have to be used with care: you may find yourself switching environments or entering multiprocess mode at an inopportune time.

- AUTOSWAP is interrupt-driven, and thus will not work with other interrupt-driven software (unless you link them together by pointing AUTOSWAP's exit address to the other routine).

- The 64's operating system is changed to RAM, so it is subject to clobbering. Be careful if POKEing about from BASIC or monitor. To bring the normal ROM back into place, just POKE 1,55. This will prevent multiprocess mode from working properly, but re-initializing AUTOSWAP with SYS 49152 will bring the RAM back in again.

- AUTOSWAP uses 2K of memory from $9000 to $97FF to store the page 0-4 image for each environment. The BASIC partition in environment 2 ends at $9000 so that this space is not stepped on. (The very top 2K, from $9800 to $9FFF is free for future expansion of autoswap to 4 environments.) If you want to run a package that normally deposits itself in high memory, either install autoswap first, or protect the top 4K of memory before loading the high-memory program. To protect $9000 to $9FFF, enter POKE 56,144, then NEW before loading.

- There are two locations in low memory used by AUTOSWAP; $02F0 (752) and $02F1 (753). The former is used to store the stack pointer, and the latter the last key pressed (used for program control of function keys). Stepping on these locations could be disastrous. If you need them, change AUTOSWAP to use some other locations instead (The equates are the first two in the source listing).

Finally

I hope that the AUTOSWAP application served to show you that your C64 is capable of more than just simple processing. If you find it useful, that's great. If you learned something, that's even better. Because the reason the computer enthusiast pushes his machine to greater and greater limits is not for the program itself, it is for fun. Why is it fun? It's not often spoke of, but the motivating force that keeps us bashing away on a keyboard until the wee hours, can be said in a word: EDUCATION.

Listing 1.

This is the BASIC loader program for AUTOSWAP: enter it, and when you get it to run without data errors, SAVE it. See text for start-up instructions.

The first bold-face value should be changed to 55 if you have different ROMs, and the second controls the frequency of swaps – set to one every three interrupts here. See text for details.

```
  100 rem** autoswap basic loader **
  110 :
  120 cs=0: rem* checksum *
  130 for i=49152 to 49766
  140 read a: poke i,a: cs=cs+a
  150 next i
  160 if cs<>84702 then print " data error " : stop
  170 print " Ok, program is in place. "
  180 print " See article for startup instructions. "
  190 end
  200 :
 1000 data  76,  50, 192, 120, 169,   8
 1010 data 141, 136,   2, 169,   0, 141
 1020 data   0,  78, 169,   1, 133,  43
 1030 data 169,  78, 133,  44, 169,   0
 1040 data 133,  55, 169, 144, 133,  56
 1050 data 169,   2, 141, 184, 192, 141
 1060 data 185, 192, 169,  39, 141,  24
 1070 data 208, 169, 147,  32, 210, 255
 1080 data  88,  96, 120, 169, 186, 141
 1090 data  20,   3, 169, 192, 141,  21
 1100 data   3,  32,  62, 194, 169,  87
 1110 data 141,  40,   3, 141,  40, 147
 1120 data 141,  40, 151, 169, 194, 141
 1130 data  41,   3, 141,  41, 147, 141
 1140 data  41, 151, 169, 53, 133,   1
 1150 data 141,   1, 144, 141,   1, 148
 1160 data 169,  76, 141,  52, 235, 169
 1170 data  43, 141,  53, 235, 169, 194
 1180 data 141,  54, 235, 169, 134, 141
 1190 data 219, 228, 169,   2, 141, 220
 1200 data 228, 173, 177, 192, 141, 176
 1210 data 192, 169,   4, 141, 174, 192
 1220 data 169,   1, 141, 185, 192, 141
 1230 data 184, 192, 169,   0, 141, 171
 1240 data 192, 141, 241,   2, 169,   0
 1250 data 141,   0,  12, 169,   1, 133
 1260 data  43, 169,  12, 133,  44, 169
 1270 data   0, 133,  55, 169,  78, 133
 1280 data  56,  88,  96,   0,   0,   0
 1290 data   4,   0,   0, 3,   0,  16
 1300 data 237, 246,   2,   2,   1,   1
 1310 data 173, 241,   2, 208,  33, 169
 1320 data   0, 141, 175, 192, 173, 171
 1330 data 192, 201,  64, 208,   8, 169
 1340 data   0, 141, 173, 192,  76, 236
 1350 data 192, 173, 173, 192, 208,  20
 1360 data 169,   1, 141, 173, 192, 173
 1370 data 171, 192, 201,   4, 240,  75
 1380 data 201,   5, 240,  36, 201,   6
 1390 data 240,  38, 173, 178, 192, 240
 1400 data  14, 206, 176, 192, 208,   9
 1410 data 173, 177, 192, 141, 176, 192
 1420 data  76,  34, 193, 165, 197, 141
 1430 data 171, 192, 169,   0, 141, 241
 1440 data   2,  76,  49, 234,  32, 237
 1450 data 193,  76,  34, 193, 173, 178
 1460 data 192,  73, 255, 141, 178, 192
 1470 data 173, 184, 192, 205, 185, 192
 1480 data 240, 221, 169,   1, 141, 175
 1490 data 192, 165, 197, 141, 197, 144
 1500 data 141, 197, 148, 173, 184, 192
 1510 data 201,   1, 240,   7, 201,   2
 1520 data 240,  72,  76, 255, 192, 186
 1530 data 142, 240,   2, 169,  96, 141
 1540 data 234, 193, 162,   0, 160, 144
 1550 data  32, 199, 193, 173, 175, 192
 1560 data 240,  45, 174, 182, 192, 142
 1570 data  46, 194, 142,  58, 194, 232
 1580 data 232, 142,  54, 194, 169,   2
 1590 data 141, 184, 192, 169,  76, 141
 1600 data 234, 193, 169, 123, 141, 235
 1610 data 193, 169, 193, 141, 236, 193
 1620 data 162, 148, 160,   0,  76, 199
 1630 data 193, 174, 240,   2, 154,  76
 1640 data 255, 192, 186, 142, 240,   2
 1650 data 169,  96, 141, 234, 193, 162
 1660 data   0, 160, 148,  32, 199, 193
 1670 data 173, 175, 192, 240,  45, 174
 1680 data 183, 192, 142,  58, 194, 142
 1690 data  46, 194, 232, 232, 142,  54
 1700 data 194, 169,   1, 141, 184, 192
 1710 data 169,  76, 141, 234, 193, 169
 1720 data 192, 141, 235, 193, 169, 193
 1730 data 141, 236, 193, 162, 144, 160
 1740 data   0,  76, 199, 193, 174, 240
 1750 data   2, 154,  76, 255, 192, 142
 1760 data 218, 193, 140, 221, 193, 174
 1770 data 174, 192, 160,   0, 140, 217
 1780 data 193, 140, 220, 193, 185,   0
 1790 data   0, 153,   0,   0, 200, 208
 1800 data 247, 238, 218, 193, 238, 221
 1810 data 193, 202, 208, 238,  76,   0
 1820 data   0, 173, 185, 192, 201,   1
 1830 data 240,  23, 169,   1, 141, 185
 1840 data 192, 169,  16, 141, 179, 192
 1850 data 169, 144, 141, 182, 192, 169
 1860 data   0, 141, 183, 192,  76,  31
 1870 data 194, 169,   2, 141, 185, 192
 1880 data 169,  32, 141, 179, 192, 169
 1890 data   0, 141, 182, 192, 169, 148
 1900 data 141, 183, 192, 173,  24, 208
 1910 data  41,  15,  13, 179, 192, 141
 1920 data  24, 208,  96, 138, 174, 198
 1930 data   0, 236, 137,   2, 176,   7
 1940 data 157, 119,   2, 232, 142, 198
 1950 data   0,  76,  66, 235, 169,  32
 1960 data 141, 174, 192, 169,  96, 141
 1970 data 234, 193, 162, 160, 160, 160
 1980 data  32, 199, 193, 162, 224, 160
 1990 data 224,  32, 199, 193,  96, 173
 2000 data 184, 192, 205, 185, 192, 208
 2010 data   3, 108, 180, 192, 169, 255
 2020 data 201, 127,  96
```

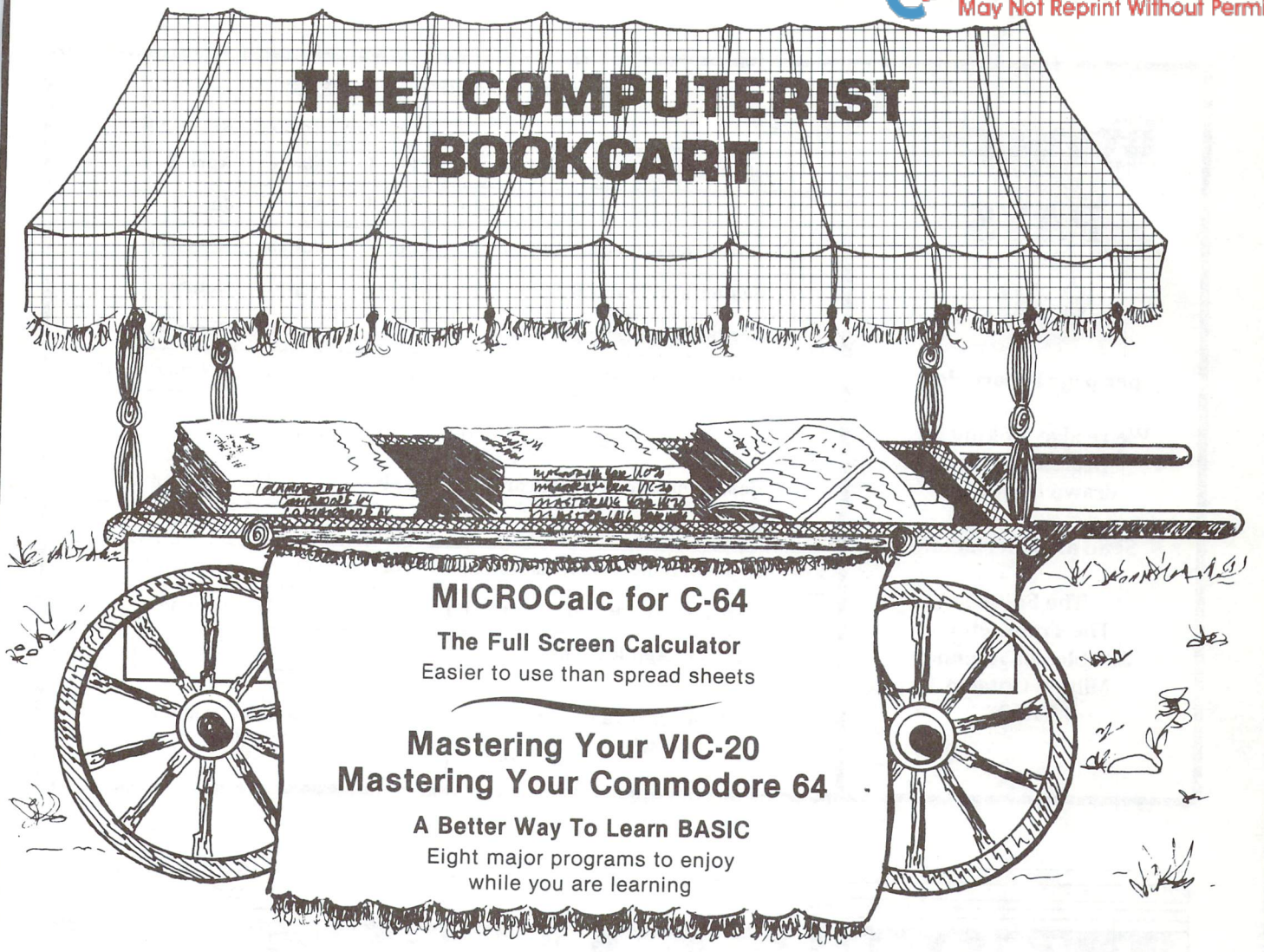
Listing 2.

The source sode for AUTOSWAP, written using the PAL 64 assembler. It should be compatible with most other assemblers.

```
1000 sys700                    ;assembled on pal 64
1010 .opt oo
1020      *      = $c000
1030 sp          = $02f0       ;stack pointer
1040 prgkey      = $02f1       ;last key detected
1050 keybd       = 197         ;key pressed
1060 pntrs       = $002b       ;basic pointers
1070 scrnpag     = $0288       ;screen page
1080 romptch1    = $eb34       ;kbd buffer irq code
1090 romptch2    = $e4db       ;colour memory
1100 stopvec     = $0328       ;check stop key
1110 chrcolor    = $0286       ;character colour
1120 ;
1130 e1start     = $0c01       ;env 1 basic start
1140 e1end       = $4e00       ;env 1 basic end
1150 e2start     = $4e01       ;env 2 basic start
1160 e2end       = $9000       ;env 2 basic end
1170 image1      = $9000       ;env 1 storage
1180 image2      = $9400       ;env 2 storage
1190 ;
1200 ;main intitialization routine
1210             jmp  irqinit
1220 ;
1230 ;entry point to initialize env 2
1240             sei
1250             lda  #8       ;screen memory page
1260             sta  scrnpag  ;at $0800 in env 2
1270 ;set up basic partition
1280             lda  #0
1290             sta  e2start-1
1300             lda  #<e2start ;basic start low
1310             sta  pntrs
1320             lda  #>e2start ;start high
1330             sta  pntrs+1
1340             lda  #<e2end  ;end low
1350             sta  pntrs+12
1360             lda  #>e2end  ;end high
1370             sta  pntrs+13
1380 ;
1390             lda  #2
1400             sta  envno    ;environment #2
1410             sta  scrno    ;screen #2
1420             lda  #$27     ;set video chip
1425             sta  $d018    ;for $0800
1430             lda  #147     ;clear screen
1440             jsr  $ffd2    ;chrout routine
1450             cli
1460             rts           ;end of setup routine
1470 ;
1480 ;
1490 irqinit     = *
1500 ;initialize autoswap and env 1
1510             sei
1520 ;redirect irq vector
1530             lda  #<intrtn
1540             sta  $0314
1550             lda  #>intrtn
1560             sta  $0315
1570 ;
1580 ;copy rom into underlying ram
1590             jsr  romstor  ;secure rom
1610 ;
1620 ;change stop vector
1630             lda  #<newstop
1640             sta  stopvec
1650             sta  image1+stopvec
1660             sta  image2+stopvec
1670             lda  #>newstop
1680             sta  stopvec+1
1690             sta  image1+stopvec+1
1700             sta  image2+stopvec+1
1710 ;
1720 ;select ram under rom
1730             lda  #$35
1740             sta  1
1750             sta  image1+1
1760             sta  image2+1
1770 ;stuff in rom patches
1780             lda  #$4c     ;jmp instr
1790             sta  romptch1
1800             lda  #<patch  ;patch address low
1810             sta  romptch1+1
1820             lda  #>patch  ;patch address high
1830             sta  romptch1+2
1840             lda  #<chrcolor;change colour for clear
1850             sta  romptch2 ;(not required on all rom
1860             lda  #>chrcolor;versions)
1870             sta  romptch2+1
1880 ;
1890 ;initialize variables
1900             lda  swpfreq  ;frequency of swaps
1910             sta  intno
1920             lda  #4       ;# of block for transfer rtn
1930             sta  blkno
1940             lda  #1       ;screen 1, environment 1
1950             sta  scrno
1960             sta  envno
1970             lda  #0
1980             sta  key      ;previous key pressed
1990             sta  prgkey   ;program control
2000 ;
2010 ;set up basic partition
2020             lda  #0
2030             sta  e1start-1
2040             lda  #<e1start ;basic start low
2050             sta  pntrs
2060             lda  #>e1start ;start high
2070             sta  pntrs+1
2080             lda  #<e1end  ;end low
2090             sta  pntrs+12
2100             lda  #>e1end  ;end high
2110             sta  pntrs+13
2120 ;
2130             cli
2140             rts
2150 ;
2160 ;
2170 ;variables follow
2180 ;
2190 key         .byte 0
2200 prevkey     .byte 0
2210 keyflag     .byte 0
2220 blkno       .byte 4
2230 swapflg     .byte 0
2240 intno       .byte 0       ;interrupt count
2250 swpfreq     .byte 3       ;swap frequency
2260 autoflg     .byte 0       ;auto swap mode
2270 scrnloc     .byte $10     ;screen location
2280 oldstop     .word $f6ed
2290 kbuf1       .byte 2
2300 kbuf2       .byte 2
2310 envno       .byte 1       ;environment 1 or 2
2320 scrno       .byte 1       ;screen 1 or 2
2330 ;
2340 ;
2350 intrtn      = *
2360 ;interrupt service routine
2370             lda  prgkey   ;program control
2380             bne  onkey    ;been poked, perform
2390             lda  #0
2400             sta  swapflg
2410             lda  key      ;last key detected
2420             cmp  #64      ;64 = no key pressed
2430             bne  keydn
2440             lda  #0
2450             sta  keyflag
2460             jmp  autochk  ;no key; autoswap"?
2470 keydn       = *
2480             lda  keyflag
2490             bne  autochk
2500 ;key pressed
2510             lda  #1
2520             sta  keyflag
2530             lda  key      ;which key pressed"?
2540 onkey       = *
2550             cmp  #4       ;f1, save out
2560             beq  mout
2570             cmp  #5       ;f3, swap
2580             beq  switch
2590             cmp  #6       ;f5, toggle auto mode
2600             beq  automode
2610 ;check for auto swap
2620 autochk     = *
2630             lda  autoflg  ;multiprocess mode"?
2640             beq  out      ;no, get out
2650             dec  intno    ;time to swap"?
2660             bne  out      ;no, get out
2670             lda  swpfreq  ;yes, reset counter
2680             sta  intno
2690             jmp  swap     ;perform env swap
2700 ;
2710 out         = *           ;irq exit
2720             lda  keybd    ;key pressed
2730             sta  key
2740             lda  #0
2750             sta  prgkey
2760             jmp  $ea31
2770 ;
2780 switch      = *           ;swap key routine
2790             jsr  scrnswap ;switch screen
2800             jmp  swap     ;switch environment
2810 ;
2820 automode    = *           ;multiprocess key rtn
2830             lda  autoflg  ;multiprocess flag
2840             eor  #255     ;toggle (flip bits)
2850             sta  autoflg
2860             lda  envno    ;leave system in
2870             cmp  scrno    ;..visible environment
```

```
2880            beq  out        ;exit if screen = environmen
2890 ;
2900 swap       =    *          ;swap environments
2910            lda  #1
2920            sta  swapflg
2930            lda  keybd      ;keep keystrokes alive
2940            sta  image1 + keybd
2950            sta  image2 + keybd
2960 mout       =    *
2970            lda  envno      ;environment # (1 or 2)
2980            cmp  #1         ;in env 1"?
2990            beq  envo1      ;yes, switch to 2
3000            cmp  #2         ;in env 2"?
3010            beq  envo2      ;yes, switch to 1
3020            jmp  out        ;neither, do nothing
3030 ;
3040 ;
3050 envo1      =    *
3060 ;first save out environment 1
3070            tsx             ;save stack pointer
3080            stx  sp
3090 ;
3100            lda  #$60       ;rts instruction
3110            sta  rtrn       ;for transfer subrtn
3120 ;
3130            ldx  #$00       ;source high
3140            ldy  #>image1   ;dest  high
3150 ;
3160            jsr  transfer   ;save pages 0-4 out
3170            lda  swapflg    ;should env 2 move in"?
3180            beq  nswap1     ;no, just f1 key 0ressed
3190 ;
3200 ;change roms to put chars in right buffer
3210            ldx  kbuf1
3220            stx  patch1 + 2
3230            stx  patch3 + 2
3240            inx
3250            inx
3260            stx  patch2 + 2
3270 ;
3280            lda  #2
3290            sta  envno      ;set to env #2
3300 ;
3310            lda  #$4c       ;jmp instruction
3320            sta  rtrn
3330            lda  #<bagn2    ;return address low
3340            sta  rtrn + 1
3350            lda  #>bagn2    ;return address high
3360            sta  rtrn + 2
3370 ;
3380            ldx  #>image2   ;source high
3390            ldy  #$00       ;dest  high
3400 ;
3410            jmp  transfer   ;move env 2 in
3420 bagn2      =    *
3430            ldx  sp         ;restore stack
3440            txs             ;..pointer
3450 nswap1     =    *
3460            jmp  out        ;finished
3470 ;
3480 ;
3490 envo2      =    *
3500            tsx             ;save stack pointer
3510            stx  sp
3520            lda  #$60       ;rts
3530            sta  rtrn
3540 ;
3550            ldx  #$00       ;source high
3560            ldy  #>image2   ;dest  high
3570 ;
3580            jsr  transfer   ;save out env 2
3590            lda  swapflg
3600            beq  nswap2
3610 ;rom changes for key diversion
3620            ldx  kbuf2
3630            stx  patch3 + 2
3640            stx  patch1 + 2
3650            inx
3660            inx
3670            stx  patch2 + 2
3680 ;
3690            lda  #1         ;set to env #1
3700            sta  envno
3710 ;
3720            lda  #$4c       ;jmp instruction
3730            sta  rtrn
3740            lda  #<bagn4    ;return address low
3750            sta  rtrn + 1
3760            lda  #>bagn4    ;return address high
3770            sta  rtrn + 2
3780 ;
3790            ldx  #>image1   ;source high
3800            ldy  #$00       ;dest  high
3810            jmp  transfer   ;move env 1 in
3820 bagn4      =    *
3830            ldx  sp         ;restore stack
3840            txs             ;..pointer
3850 nswap2     =    *
3860            jmp  out
3870 ;
3880 ;
3890 transfer   =    *
3900 ;memory move routine
3910            stx  srcptr + 2 ;source block
3920            sty  destptr + 2 ;dest block
3930            ldx  blkno      ;# of blocks to transfer
3940            ldy  #0
3950            sty  srcptr + 1
3960            sty  destptr + 1
3970 transbyt   =    *
3980 ;may I be struck down by lightning for
3990 ;doing this, but the next two instructions
4000 ;are (gasp) self-modifying.
4010 ;apologies - it was the only way.
4020 srcptr     lda  !*-*,y
4030 destptr    sta  !*-*,y
4040            iny
4050            bne  transbyt
4060            inc  srcptr + 2
4070            inc  destptr + 2
4080            dex
4090            bne  transbyt
4100 rtrn       jmp  *-*
4110 ;
4120 ;
4130 scrnswap   =    *
4140 ;swap screen pointer
4150            lda  scrno
4160            cmp  #1         ;in screen 1"?
4170            beq  to2        ;yes, switch to 2
4180            lda  #1         ;no, switch to 1
4190            sta  scrno
4200            lda  #$10       ;bit for vic chip
4210            sta  scrnloc
4220            lda  #>image1   ;set up rom patches
4230            sta  kbuf1
4240            lda  #$00
4250            sta  kbuf2
4260            jmp  switchit
4270 ;
4280 to2        =    *
4290            lda  #2         ;set to screen 2
4300            sta  scrno
4310            lda  #$20
4320            sta  scrnloc
4330            lda  #$00
4340            sta  kbuf1
4350            lda  #>image2
4360            sta  kbuf2
4370 ;
4380 switchit   =    *
4390            lda  $d018      ;make vic chip look
4400            and  #$0f       ;..at current screen
4410            ora  scrnloc
4420            sta  $d018
4430            rts
4440 ;
4450 ;
4460 patch      =    *          ;rom patch
4470            txa
4480 patch1     ldx  !$00c6
4490            cpx  $0289
4500            bcs  back
4510 patch2     sta  $0277,x
4520            inx
4530 patch3     stx  !$00c6
4540 back       jmp  $eb42
4550 ;
4560 ;
4570 romstor    =    *
4580 ;copy rom into underlying ram
4590            lda  #$20       ;move $20 blocks
4600            sta  blkno
4610            lda  #$60       ;jsr
4620            sta  rtrn
4630 ;
4640            ldx  #$a0       ;source high
4650            ldy  #$a0       ;dest high
4660            jsr  transfer
4670 ;
4680            ldx  #$e0
4690            ldy  #$e0
4700            jsr  transfer
4710 ;
4720            rts
4730 ;
4740 ;
4750 newstop    =    *
4760 ;stop vector
4770            lda  envno
4780            cmp  scrno
4790            bne  nostop     ;is screen = env"?
4800            jmp  (oldstop)  ;yes, normal stop vector
4810 nostop     =    *          ;no, ignore stop
4820            lda  #$ff
4830            cmp  #$7f
4840            rts
```

THE COMPUTERIST BOOKCART



Mastering Your VIC-20
Mastering Your Commodore 64

The 8 programs, "run-ready" on disk (C-64) or tape (VIC-20) and explained in the 160-192 page book, each demonstrate important concepts of BASIC while providing useful, enjoyable software. Programs include:

- Player — compose songs from your keyboard, save, load and edit for perfect music
- MicroCalc — display calculation program that make even complex operations easy
- Master — a one or two person guessing game
- Clock — character graphics for a digital clock

VIC-20 with tape & book ..... **just $19.95**
C-64 with disk & book (avail. Sept.) ..... **just $19.95**

Look for us at the
International Software Show
Toronto, September 20-23

MICROCalc for C-64

This on-screen calculator comes with diskette and 48-page manual offering a wide variety of useful screens, and a great way to learn BASIC expressions if you don't already know them.

- Unlimited calculation length & complexity
- Screens can be linked and saved on disk/cassette
- Build a library of customized screens
- Provide formatted printer output

Diskette & 48-page manual ..... **just $29.95**

For the Freshest Books, Buy Direct!

- No prehandled books with bent corners
- Books come direct to your door
- No time wasted searching store to store
- 24 hours from order receipt to shipment
- No shipping/handling charges
- No sales tax (except 5% MA res.)
- Check, MO, VISA/MC accepted (prepaid only)

The Computerist Bookcart
P.O. Box 6502, Chelmsford, MA 01824
For faster service, phone: 617/256 – 3649.

The Transactor
The Tech/News Journal For Commodore Computers

PAYS $40

per page for articles

We're also looking for professionally drawn cartoons!

Send all material to:

The Editor
The Transactor
500 Steeles Avenue
Milton, Ontario
L9T 3P7

Volume 5 Editorial Schedule

Issue#	Theme	Copy Due	Printed	Release Date
1	Graphics and Sound	Feb 1	Mar 19	April 1
2	The Transition to Machine Code	Apr 1	May 21	June 1
3	Software Protection & Piracy	Jun 1	Jul 23	August 1
4	Business and Education	Aug 1	Sep 17	October 1
5	Hardware and Peripherals	Oct 1	Nov 19	December 1
6	Programming Aids & Utilities	Dec 1	Jan 19	February 1/85

Volume 6 Editorial Schedule

1	Communications & Networking	Feb 1	Mar 21	April 1/85
2	Languages	Apr 1	May 20	June 1
3	Implementing The Sciences	Jun 1	Jul 18	August 1
4	Hardware & Software Interfacing	Aug 1	Sep 21	October 1
5	Real Life Applications	Oct 1	Nov 19	December 1

Advertisers and Authors should have material submitted no later than the 'Copy Due' date to be included with the respective issue.

What is the ULTIMATE TEST of TRIVIAL KNOWLEDGE?

6400 questions.
3 levels of difficulty for 1–8 players.
Count down clock **and** Sound & Graphics.
6 different categories.

For the Commodore 64 ™
"The Game designed to Flex the Mind!"

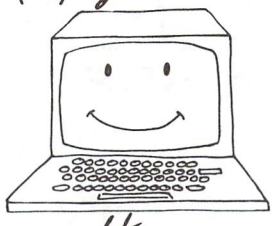
QUIZARD 6400

Send $37.95 (Can.)*
Pedlar Discount
84 Jarvis Street, Fort Erie, Ontario, L2A 2S6

Terms of Payment: cheque; money order; Master Charge or VISA.
* Ontario Residents please add 7% Provincial Sales Tax.
Commodore 64 is a trademark of Commodore Business Machines Inc.

Dealer Inquiries Invited

(M)agreeable™



software

STOCK HELPER™

Commodore 64 and VIC-20

Stock HELPER is a tool to maintain a history of stock prices and market indicators on diskette, to display charts, and to calculate moving averages. Stock HELPER was designed and written by a "weekend investor" for other weekend investors.

Stock HELPER is available on diskette for:

Commodore 64	$30.00	($37.00 Canadian)
VIC-20 (16K)	$27.00	($33.25 Canadian)

plus $1.25 shipping ($1.55 Canadian)

The VIC-20 version only charts 26 bi-weekly periods rather than 52 weekly periods.

(M)agreeable software, inc.
5925 Magnolia Lane • Plymouth, MN 55442
(612) 559-1108

(M)agreeable and HELPER are trademarks of (M)agreeable software, inc.
Commodore 64 and VIC-20 are trademarks of Commodore Electronics Ltd.

WATCOM

HANDS-ON WORKSHOPS

Special Weekend Sessions

For Microcomputer Users

Hardware Design and Interfacing

- Get over the fear of handling circuits and chips.
- Understand what happens inside the micro.

November 23-25, 1984 - Waterloo

Pascal for Microcomputers

- Learn one of the most popular and powerful languages.
- Structured Programming is emphasized.

November 16-18, 1984 - Waterloo

WATCOM is the developer of language systems for the C64, SuperPET, IBM PC, ICON and has been offering computing workshops for over a decade.

Call Sharon Malleck at
(519) 884-2700 or mail the
coupon for a detailed brochure.

WATCOM Seminars
415 Phillip Street
Waterloo, Ontario
Canada N2L 3X2

Send more information on the following workshops:

___ Hardware Design and Interfacing
___ Pascal for Microcomputers

Name: _____
Address: _____
City,Province: _____
Postal Code: _____
Telephone: (___) _____

YOUR COMMODORE 64 AND THE POWER OF DESIGN

We know you adore your Commodore 64 but did you know about its power of design? its musical capabilities? If you want to know all there is to know about your Commodore 64, including its graphics and sounds, let *Howard W. Sams*, a leader in computer publishing, tell you everything.

Commodore 64 Starter Book
Christopher A. Titus, David G. Larsen, and Jonathan A. Titus
1984, spiralbound, 382 pages.

An enjoyable introduction to programming and understanding the Commodore 64 computer. Especially written for quick learning by first-time computer users, it contains many programming experiments and practice questions, covers accessories, and explains how to intelligently select packaged programs.
0-672-22293-0 $25.50

Learn BASIC Programming in 14 Days on Your Commodore 64.
Gil M. Schechter
1984, spiralbound, 192 pages.

A fun and easy mini programming course designed especially for beginners 13 years and older. Consists of 14 understandable lessons including over 130 programs and examples.
0-672-22279-5 $18.50

Commodore 64: Graphics and Sounds.
Timothy Orr Knight
1984, paper, 128 pages, plus cassette and disk

Forty-seven programs that reveal the powerful graphics and sounds available from the Commodore 64.
0-672-22278-7 Book $12.95
0-672-26186-3 Book, cassette & disk $27.95

Mostly BASIC Applications for Your Commodore 64 Books 1 and 2
Howard Berenbon

Book 1 1984, spiralbound, 192 pages
Brings you 38 chapters filled with fun and serious BASIC programs that help you save money on energy usage, bar-chart your business sales, dial the telephone, learn a foreign language, and more. 42 easy-to-use programs in all.

Book 2 — 1984, spiralbound, 264 pages
A second collection of Commodore 64 BASIC programs which includes dungeons, memory challengers, a student grader, phone directory, monthly budget, ESP tests, and more. 87 easy-to-use programs in all.

Book 1 0-672-22355-4 $18.50
Book 2 0-672-22356-2 $20.95

To order, fill out this handy coupon and return to:

The Transactor, 500 Steeles Ave.
Milton, Ontario. L9T 3P7 (416) 876-4741

Please send me:

_____ Commodore 64 Starter Book, $25.50; _____ Learn BASIC Programming in 14 Days on your Commodore 64, $18.50; _____ Commodore 64: Graphics and Sounds. Book, $12.95; _____ Commodore 64: Graphics and Sounds Book, Cassette, Disk, $27.95; _____ Mostly BASIC Applications for Your Commodore 64, Book 1, $18.50; _____ Book 2, $20.95. Add 5% for shipping and handling per order. I enclose _____ Cheque, _____ Money Order (no COD's or cash), charge _____ VISA, _____ Mastercard.

Card No. Expiry Date

Signature

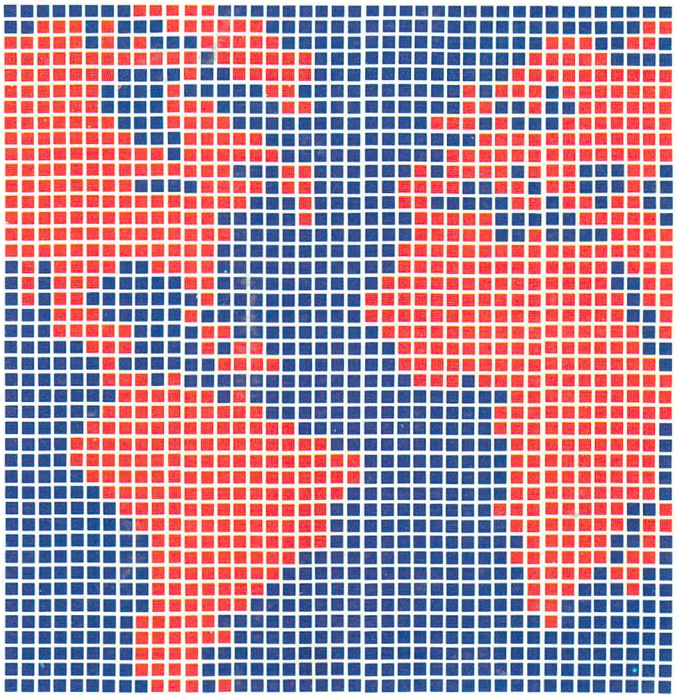
Name

Address

Prov/State Postal/Zip Code

Prices are subject to change without notice

INTERNATIONAL CENTRE, TORONTO
NOVEMBER 29 & 30, DECEMBER 1 & 2, 1984



THE WORLD OF COMMODORE II

The Company that had the foresight and imagination to design and build more computers for home, business and education than any other will be presenting the most farsighted and imaginative show to date with exhibitors from around the World.

The 1983 Canadian World of Commodore Show was the largest and best attended show in Commodore International's history. Larger than any other Commodore show in the World and this year's show will be even larger.

World of Commodore II is designed specifically to appeal to the interests and needs of present and potential Commodore owners.

Come and explore the World of Commodore.

world of commodore II

A HUNTER NICHOLS PRESENTATION.
FOR MORE INFORMATION CALL (416) 439-4140