Commodore Computers' Largest Users' Group's Magazine

TPUG Magazine

October 1984

$2.95

Commodore 64
VIC 20
PET/CBM
SuperPET

Automating Home Comfort

The Beginner and the Disk. Part IV

Accurate time on the C-64

User Port programming and hardware

Laser Art and Technology

Canada Post Postage paid
Postes Canada Port payé
Bulk third class
En nombre troisième classe
E12430
Scarborough

Introducing the Batteries Included FAMILY of Computer Products

PaperClip
The finest word processor available for the Commodore 64 now offers the ability to switch from a 40 to an 80 column display without additional hardware. PaperClip's state of the art design allows free movement of columns, phrases, sentences or paragraphs from one location to another within the text. This outstanding addition to your Commodore 64 system is now available with SpellPack.

Home Organizer Series
These programs, which are available individually, are specifically designed modules for your record keeping activities. Programs available include stamps, recipes, addresses, mailing lists, personal banking, music and book libraries, photo and home movie collection and household inventory. Each module is so easy to use even a novice can achieve professional results.

The Consultant
The Consultant, a database management system, is like a computerized filing cabinet with a brain. With it you can design a layout for any kind of information that needs filing, allowing you various methods for searching, sorting and analyzing your data quickly.

BusCard II
An improved version of the original BusCard which allows you to add any Commodore-compatible disk drive, including hard disk, and virtually any printer on the market. Just plug it in and you can mix and match your peripherals without fear of software incompatability.

B.I.-80
Another plug in module that provides an 80 column, crystal clear display. Interference is eliminated even with a full screen of characters. It is readily switched from 40 to 80 column display at anytime and can be used with Commodore 1701 and 1702 color monitor or any monochrome video monitors.

SpellPack
This program contains a dictionary of over 20,000 commonly used words in the English language. SpellPak will check each word, in its context, on the entire document in 2-4 minutes and highlight those not found in the dictionary. You may then correct the spelling if required or instantly add the word to the SpellPack dictionary with a single key command. In this manner the dictionary may be expanded by over 5,000 terms for any technical or scientific vocabulary your work may demand.

BATTERIES INCLUDED

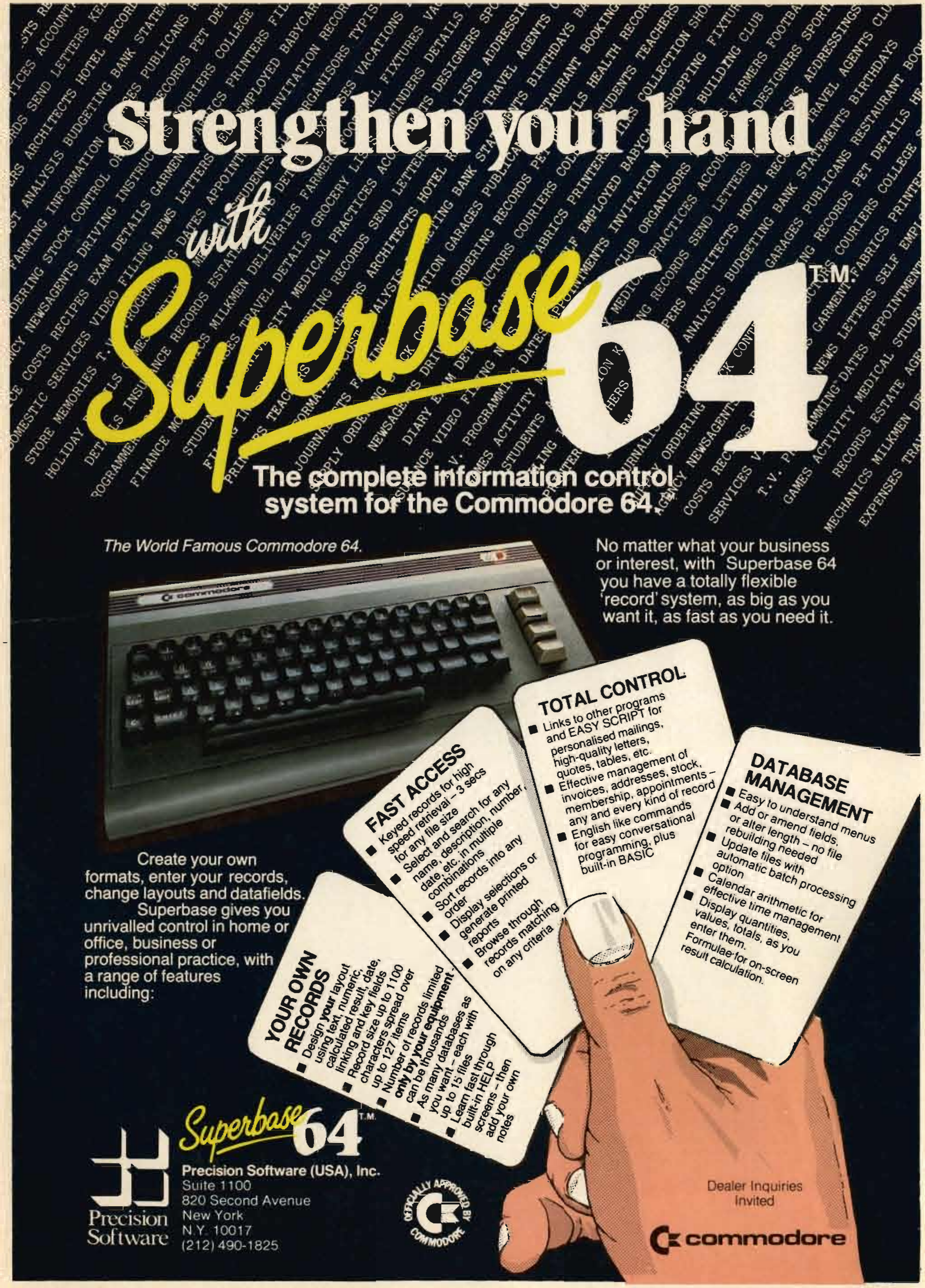
"Excellence in Software"

For a full color brochure write to

Batteries Included, 186 Queen Street West, Toronto, Canada M5V 1Z1 (416) 596-1405/3303 Harbor Blvd., Costa Mesa, CA 92626 (714) 979-0920 handic

software inc.

520 Fellowship Road, Suite B206; Mount Laurel, NJ 08054 (609) 866-1001


Strengthen your hand
with
Superbase 64 T.M.
The complete information control system for the Commodore 64.™
The World Famous Commodore 64.
No matter what your business or interest, with Superbase 64 you have a totally flexible 'record' system, as big as you want it, as fast as you need it.
Create your own formats, enter your records, change layouts and datafields. Superbase gives you unrivalled control in home or office, business or professional practice, with a range of features including:
YOUR OWN RECORDS
■ Design your layout using text, numeric, calculated result, date, linking and key fields
■ Record size up to 1100 characters spread over up to 127 items
■ Number of records limited only by your equipment - can be thousands
■ As many databases as you want - each with up to 15 files
■ Learn fast through built-in HELP screens - then add your own notes
FAST ACCESS
■ Keyed records for high speed retrieval - 3 secs for any file size
■ Select and search for any name, description, number, date, etc. in multiple combinations
■ Sort records into any order
■ Display selections or generate printed reports
■ Browse through records matching on any criteria
TOTAL CONTROL
■ Links to other programs and EASY SCRIPT for personalised mailings, high-quality letters, quotes, tables, etc.
■ Effective management of invoices, addresses, stock, membership, appointments - any and every kind of record
■ English like commands for easy conversational programming, plus built-in BASIC
DATABASE MANAGEMENT
■ Easy to understand menus
■ Add or amend fields, or alter length - no file rebuilding needed
■ Update files with automatic batch processing option
■ Calendar arithmetic for effective time management
■ Display quantities, values, totals, as you enter them.
Formulae for on-screen result calculation.
Superbase 64 T.M.
Precision Software
Precision Software (USA), Inc.
Suite 1100
820 Second Avenue
New York
N.Y. 10017
(212) 490-1825
OFFICIALLY APPROVED BY COMMODORE
Dealer Inquiries Invited
commodore

Contents

A=all, C=C-64, V=VIC 20, P=PET/CBM, S=SuperPET, *=none
G=General, B=Beginner, I=Intermediate, A=Advanced

Departments

TPUG *magazine*

Publisher
Chris Bennett

Editor
David Williams

Assistant Editor Production Manager
Sandra Waugh

Director of Advertising Sales
Louise Redgers
(416) 782-9804

Production Team
Astrid Kumas
Marya Miller
Monica Bodirsky

Graphic Design
Leslie Smart and Associates, Toronto, Ont.

Printing
Delta Web Graphics, Scarborough, Ont.

Cover Photo
Roberto Portolese Studio, Toronto, Ont.

TPUG yearly membership fees:

Regular member (attends meetings)	—$30.00 Cdn.
Student member (full-time, attends meetings)	—$20.00 Cdn.
Associate (Canada)	—$20.00 Cdn.
Associate (U.S.)	—$20.00 U.S.
	—$25.00 Cdn.
Associate (Overseas—sea mail)	—$30.00 U.S.
Associate (Overseas—air mail)	—$40.00 U.S.

TPUG Inc.,
1912-A Avenue Rd., Ste. #1
Toronto, Ont.
M5M 4A1
Telephone numbers:
Business Office (416) 782-8900
(416) 782-9252
Magazine Office (416) 782-1861
Advertising (416) 782-9804

TPUG Magazine is printed in Canada. Mailed at Toronto, ON and Buffalo, NY. Send change of address to: TPUG Inc., Address Changes, 1912-A Avenue Rd., Toronto, ON, Canada M5M 4A1

Subscription: 15,500
Newsstand: 10,000
ISSN #0825-0367

Distributed by:

Compulit Distributors
PO Box 352
Port Coquitlam, BC
V3C 4K6
(604)464-1221

Micron Distributing
409 Queen Street W.
Toronto, ON
M5V 2A5
(416)593-9862
Toll Free Order Desk
1-800-268-9052

Subscription-related inquiries are handled ONLY by TPUG.

TPUG CONTACTS

TPUG OFFICE

(416) 782-8900

(416) 782-9252

Board of Directors		
President	Michael Bonnycastle	416/654-2381
Vice-President	Chris Bennett	416/782-9252
Vice-President	Gord Campbell	416/492-9518
Treasurer	Carol Shevlin	c/o 416/782-8900
Recording Sec.	John Shepherd	416/244-1487
Directors	Bruce Beach	519/925-5376
	Rosemary Beasley	416/787-8432
	Gary Croft	416/727-8795
	Mike Donegan	416/639-0329
	John Easton	416/251-1511
	Gerry Gold	416/225-8760
	Louise Redgers	416/447-4811
TPUG Inc.		
Business Mgr.	Chris Bennett	416/782-8900
Asst. Business Mgr.	Doris Bradley	416/782-8900
TPUG Magazine		
Publisher	Chris Bennett	416/782-1861
Editor	David Williams	416/782-1861
Asst. Editor/Prod. Mgr.	Sandra Waugh	416/782-1861
Ad Director	Louise Redgers	416/782-9804
Meeting Co-ordinators		
Brampton Chapter	Garry Ledez	c/o 416/782-8900
Central Chapter	Michael Bonnycastle	416/654-2381
C-64 Chapter	Louise Redgers	416/447-4811
Comal Chapter	Donald Dalley	416/742-3790
	Victor Gough	416/677-8840
Communications	David Bradley	416/782-8900
	Richard Bradley	416/488-4776
Eastside Chapter	Kelly Grinton	416/839-7284
	Joyce Topley	416/683-4898
Forth Chapter	David Williams	416/782-1861
Hardware Chapter	David Williams	416/782-1861
Machine Language	Jim Carswell	416/531-9909
SuperPET Chapter	Gerry Gold	416/225-8760
VIC 20 Chapter	(Doris Bradley)	416/782-8900
Westside Chapter	John Easton	416/251-1511
	Al Farquharson	519/442-7000
Librarians		
Commodore 64	David Bradley	416/782-8900
	Richard Bradley	416/488-4776
French	Baudouin St-Cyr	c/o 416/782-8900
PET	Mike Donegan	416/639-0329
SuperPET	Bill Dutfield	416/224-0642
VIC 20	Chris Covell	416/925-9296
Bulletin Board	Richard Bradley	c/o 416/782-8900
Conference	Doris Bradley	416/782-8900

editorial

I feel it is sad that there are now so many people who have been born too late to have experienced the 1960's. Having spent their entire sentient lives within the last fifteen years or so, most of them must see the world as a far more drab and boring place than do we whose memories go back to that magical decade.

As a science student whose friends included many energetic and artistically creative people, I spent much of the '60's blithely ignoring the supposed cultural division between the arts and sciences. In doing so, I was far from alone. Every Friday night, some of us would gather in a basement in London, England, to entertain each other in every way we could, including with a lot of intriguing technological devices. We called ourselves and our venue "The UFO Club". Among us were some experimental rock musicians who were trying out new electronic techniques of music-making. They were quite successful. As the "Pink Floyd", they later achieved world-wide fame. Others of us, including myself, were more interested in kinetic visual effects. Armed, mainly, with modified slide projectors in which the slide-holders had been replaced with various means of producing moving coloured patterns, we decorated the walls of the club with brilliant, swirling, bubbling, writhing shapes. As the Floyd played, we endeavoured to provide appropriate visual accompaniments to their music. Some of our light-machines were capable of being "played" as para-musical instruments. A light-artist would sit with his fingers on a set of buttons, pressing them in time with the music. In response, the machine would vary the projected patterns.

By the standards of the '80's, of course, our efforts were extremely primitive. Nothing we did then could come close to matching the effects which can be produced today. We projected patterns onto walls. The laser-artists of 1984 etch their images in stark clarity into the fabric of the cosmos.

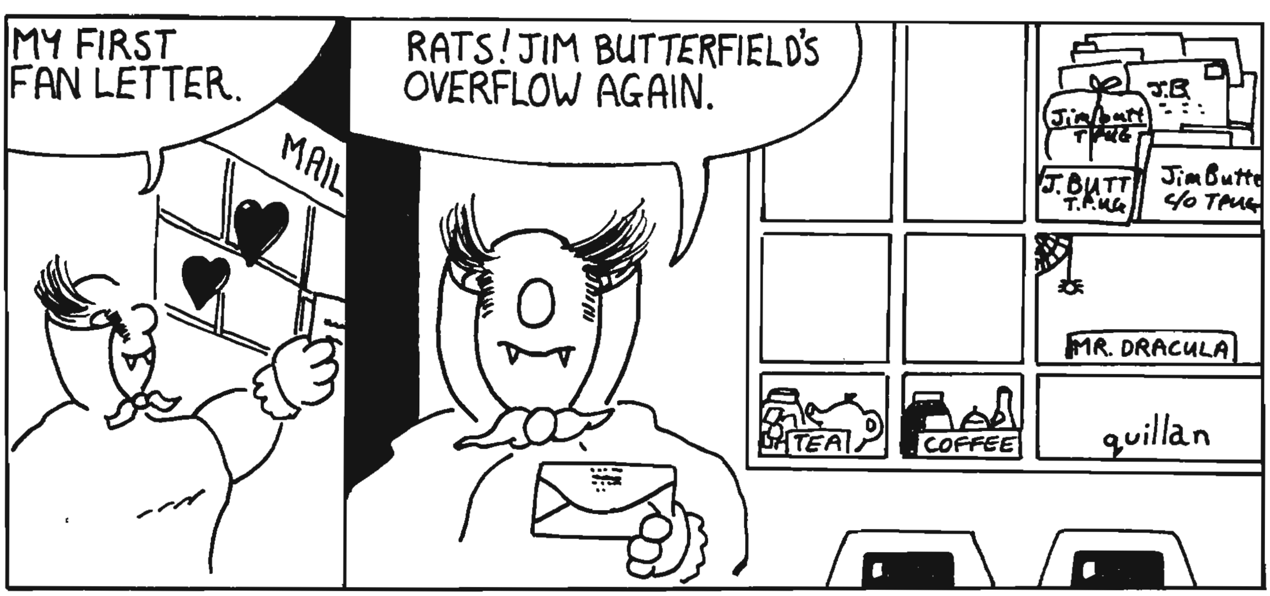
The last sentence is, of course, an exaggeration in a physical sense. But it is totally inadequate to describe the sensation of watching a laser display projected upon the dome of a planetarium. Using equipment which incorporates an astonishing array of differing technologies, a laser-artist can transport his audience into a dazzling new area of experience. Space itself becomes the arena in which luminous coloured shapes, some abstract and some realistic, some crystal-sharp and others tenuously ghostly, float and whirl in synesthetic harmony with the music.

Since the technology involved in laser art includes the use of such things as microprocessors and digital memories, it is clearly an appropriate topic for this magazine. In this issue, you will find an article by Greg Lowry, who is the artist responsible for the laser displays at the McLaughlin Planetarium, in Toronto. His art is shown in some of our photographs, including those on the front cover. Yet, in a static, low-contrast medium such as the pages of a magazine, it is impossible to convey more than a faint shadow of what he actually does. For those readers who have attended laser shows, his article should illuminate some of their technical facets. But for those who have not yet seen this art-form, we cannot claim that this magazine offers a worthwhile substitute.

Greg's article should also stimulate the imagination of readers who want to explore new uses for their computers. Although none of the Commodore machines with which we are all familiar can rival the sophistication of the equipment which Greg uses, they can be used to control external devices, including lighting systems. Readers who feel up to building some hardware, as well as writing programs, should be able to produce some very interesting displays of synchronized music and light. I know this is a field which I, myself, intend to explore! *TPUG*

David Williams

MR. DRACULA



THIS & THAT

Doris Bradley
Assistant Business Manager

MARCA Conference

July 28th and 29th found me in Hershey, Pennsylvania, attending the First Annual Commodore Users' Conference and Computer Show of the Mid-Atlantic Regional Commodore Association. Hats off to the co-ordinating committee which put together this successful combination of speakers and dealer display. Congratulations especially to Mindy Skelton, who handled registration. As the one who co-ordinated registration for TPUG's Third Annual Conference this May, I can fully appreciate all the work that she did in preparation for July 28th. I must say I enjoyed being a participant rather than an organizer. I attended a great many sessions including Pre-Beginner Machine Language with Lou Sander, Better BASIC with Jim Butterfield and Women in Computing with Ellen Strasma.

I now realize how spoiled we are here in Toronto. Just about any month a Regular or Student member can hear Jim Butterfield expound as only Jim Butterfield can. I had to attend his session in Hershey to fully appreciate how much others appreciate him and just how lucky we are!

I expect my Hershey experience will be useful to me as co-ordinator of TPUG's Fourth Annual Conference next May.

Another Riddle

What does a person who does word processing like to drink?

TPUG Bulletin Board

By the time you read this, the Bulletin Board will have moved again. The new number is 416-782-9534. Hopefully, this is the last move for a while!

VIC Programming Contest

I trust that your entry to the VIC programming contest is in! The deadline is October 2, 1984. The two sections to the contest are (1) programs written in BASIC and (2) programs written in another language.

Tape Terminal Programs

Gord Campbell has put together a set of terminal programs which will work on tape. As yet, this "group" has not been properly organized for the library and therefore does not have an official library name. However, if you wish to purchase this tape in its present form (cost $6.00), just ask for the "tape terminal programs".

Associate Club Chapters

The list of groups who have taken advantage of the offer to have fifteen or more of their members join TPUG at a saving of $5.00 per member is growing all the time. The latest additions are: Southern California 64 Users Group, Eagle Rock Commodore Computer Club of Idaho Falls, Idaho; Phoenix Arizona Commodore Club; Mid-City Commodore Club of Fort Worth/Arlington Texas; Mountain Computer Society, Utah; and Anchorage Commodore Users Group, Alaska. We now have fifty-one associated clubs!

TPUG Conference 1984

Have you sent in the Appraisal Sheet which was on page 58 of the August/September TPUG Magazine? No? — please take a few minutes and do it today. Meetings to plan the next Conference are already under way.

Coming Events

A Machine Language course taught by David Williams (Editor of *TPUG Magazine*) will be offered in the Toronto area. 10 classes from 5:00 to 7:00 pm. on Tuesday evenings, starting October 2nd. Location — Lakeshore Collegiate (Kipling Avenue, near Lakeshore). For further information call The Continuing Education Department, Etobicoke Board of Education (626-4360) and ask about course # 1815. Total course cost — $20.00.

Storywriter for the C-64

Stop the presses! Don Whitewood of the Toronto Board of Education has succeeded in getting version 11 of Storywriter for the PET converted to the Commodore 64. Knowing Don, this is THE version to have. There are a few other versions out and about, but unfortunately they are based on earlier versions of the PET program. At the time of writing, I do not know which C-64 disk Storywriter will appear on. If you can't wait for next month's magazine before ordering, just send in an order for "the disk with Storywriter for the 64", and we'll do the rest.

NOS Translator

In the July issue it was stated that the NOS Translator programs had been given to the TPUG Librarians for inclusion in the club library. I have asked that these programs appear on the September releases for the VIC 20, PET and C-64.

Tape Mastering

TPUG has purchased a Sony high-speed tape duplicator to copy Commodore Educational tapes, plus any future C-64 and VIC tapes. The arrival of this machine will speed up our ability to make these tapes available to members. Gone forever are the long waits for tapes!

At long last we are having success with our disk-to-tape program and intend to produce any PET tapes that are ordered by this method.

Tape Drives

Chris Bennett has spent many hours testing the alignment of datasettes. We recently purchased two brand new datasettes from a CBM dealer — one was okay, the second was way out of alignment; — some programs would load, others would not. By persevering, Chris has mastered aligning a tape drive without sending it to Commodore — and without using an oscilloscope. He is now working on developing a tape alignment kit for about $5.00. If he can get the special screwdriver he needs, it may be free with an order of four tapes. More news on this later. . .

Annual General Business Meeting

As mentioned in the Calendar of TPUG Events, the Annual General Business Meeting will take place on Thursday, October 4 at Leaside High School, Bayview & Eglinton Aves., Toronto, at 7:30 p.m. I am taking the space to repeat the information again here because it is important that 'Regular' Members in particular take note, mark the date on their calendars and make a point of coming out and participating. The financial statement for the fiscal year will be discussed, the executive for the next year will be elected, etc.

The vitality of any democratically-organized group can be measured by the active involvement of its members. Just as it is your right (and I would say, your duty) to take an informed part in a political election, so it is your right (and I would say, your duty) to take an active interest in your club. If you can't come in person, read over the "Notice Of Meeting" package which Regular members will receive in the mail, and make sure that your views are being represented by using your proxy form.

Answer

A word processor drinks TAB. *TPUG*

HELP !

Do you have anything for this column? The three headings are:

(1) Helpful Hints,

(2) Who's Got the Answer? and

(3) "PET" Pals Wanted.

Just send your contributions (including answers to any questions which have appeared) to:

Toronto PET Users Group
Dept. Help
1912A Avenue Rd., Ste. 1
Toronto, Ontario, Canada
M5M 4A1

Please let us know if you wish your full address published.

S.O.S. from Doris Bradley: If you follow the HELP column regularly you will be aware that there are many people out there with questions. On the other side of the coin, we do have some very knowledgeable and helpful members who pitch in and provide information, technical help etc. Just recently Clayton W. Dewey K8CKD volunteered to answer questions from HAM operators. His offer got me thinking about one of the biggest problems I have—where to find someone who can answer a specific question. What I'd like to do is to set up a Manager file of those members who could be called upon occasionally to provide some assistance (indicating their area(s) of expertise). If you might consider volunteering, drop me a line or give me a call and let's discuss it. I would only provide you with one inquiry a month (unless you indicated otherwise).

Helpful Hints

This month the **Hints** section takes on a new look. Below you will find questions along with answers which have been provided by some of our experts in the Toronto area. As much as possible, we will try to deal in this way with some of the more interesting questions which probably have wide appeal.

(Q) I would like to address a problem that we have encountered and I'm sure other members have also come across. When you acquire new programs in the Commodore Educational Series, you incorporate them in alphabetical order. This causes the member who previously owned all the programs to do without, or order a whole new set. I understand that this is the easiest and most organized way to list these sets. If I may be so bold as to suggest also incorporating new diskettes consisting of these newly released programs: i.e. "ED ADDENDUM #1", so that the member can keep a full set without too much trouble and expense.

William F. Besterman
Nautilus Computer Club
Groton, CT 06340

(A) *I understand fully the problem you are faced with, but unfortunately TPUG is not the originator of this series, and so does not have the opportunity to incorporate the programs as we would wish. If we had the time, we would like to take the Commodore Educational Software library and reorganize it according to grade level. As a matter of fact, if David Bradley, one of the Commodore 64 librarians, succeeds in reorganizing all the current monthly disks for the C-64 into category disks before the summer is over, he intends to do just that. In addition, if you were to supply a list of the newly released programs, he might also be able to do as you suggest.*

(Q) It would be greatly appreciated if you can explain how I can include your Lockdisk 64 program (or something equal) to my disks, since it is in ML and I don't have a working knowledge of it at this time, and am still confused tremendously even though I have the (C)E1 disk. Any help you can provide in clearing the snow of ML programing would be greatly appreciated.

Also in your copy programs: File Copy and Copy All have SYS #'s in them which prevent making comments in the program or otherwise modifying them. Could you explain why, for what purpose and how they could be implemented in any programs I write.

I appreciate your efforts in writing all these various programs and would hope that you would start a series on programing, mainly in the area of how to use the various routines that are currently imbedded in the C-64 for assisting in writing programs for us newer-type people who have only had a year or so with computers.

Another request I have of you is for a Christian Brother who teaches computers at St. Joseph H.S. here in Westchester. His request is for a PRINT USING command which he got used to on an Apple. The high school at present has 12 C-64's and 4 1541 drives

James M. Ardovitch
Chicago, Illinois

(A) *Lockdisk 64 is a BASIC program which has locked itself. It can be copied by using any readily available copy program such as 1541 Backup or Unicopy.*

The SYS #s call up Machine Language parts of the program. The Machine Language is used for speed. You will need a Machine Language monitor to list this, plus a little knowledge.

There is a brief PRINT USING program in the library which will format numbers neatly for output in columns.

(Asst. Business Mgr.'s Note: Jim Butterfield has a book on Machine Language programming which was just recently published. It is called "An Introduction to Commodore Machine Language Programming" and is published by Robert J. Brady Co.)

(Q) I am the "Computer Lady" here at the Dubin Center. We own and use 2 Commodore PETs, 1 CBM 8032, and 1 Commodore 64.

Quite often we have had the situation where there are some programs on the 1541 disk format that we wish to have also on our PETs or CBM. I have tried many times to save these programs onto tape, but these tapes will not run or list on either the PETs or the CBM.

Can you help me? What will make these programs run and list on the other machines?

Heather Albright
Santa Barbara, California

(A) *To make the programs loadable on the PET, you must make the C-64 pretend temporarily that it is a PET too. The following POKEs on the C-64 will do it:*

POKE 56576,5:POKE 53272,4:POKE 648,128: POKE 1024,0:POKE 44,4:POKE 56,128:NEW

These should all be given in one line. The resulting screen will be messy so clear the screen. Now load the programs that you wish *to transfer one last time, save on a fresh tape or disk and these newly saved programs will load correctly into the PET. WARNING! If the program uses special features of the 64 such as colour or sound, this procedure will get it to load but you still will not be able to run it.*

(Q) I am trying to obtain a utility that will do the following: To make a M/L Code program on a HIRES or NORMAL screen, step one instruction at a time, so at any point I can read the instruction last executed, its location in memory, and the A X Y registers.

Ideally, the utility should reside from $C000 upwards and be able to read programs up to $BFFF after lifting the ROM at $A000.

I have been told that your EXTRAMON may be able to do this but I cannot get any information on it. If you know of a program that will do what I require would you let me know where I can obtain it with instructions (1541 disk if possible).

Roger K. Walsome
Feltham, England

(A) *Supermon 64 is available, but it does not contain a single step version. For debugging I would recommend inserting the break command. The source programs for Supermon have been given to the Commodore 64 librarians, but I am not sure if they have appeared in the library yet.*

(Q) Apparently Commodore U.K. are about to release a Z80 cartridge together with CP/M. Is this available in Canada yet?

I am very interested in Logo. By that I mean a full use/implementation, not just Turtle Graphics. "The surface of Logo has yet to be scratched – my theory!"

Alistair O'Connell
Cornwall, England

(A) *As far as CP/M is concerned, yes; there is a limited CP/M available. It is "limited" since, first of all, the C-64 has only 40 columns, while most CP/M programs expect more. Secondly, the version we have seen operates in 48K of memory and many CP/M programs expect 56K. Finally, Commodore disks have a different format than most CP M disks. It seems likely that the Commodore version is a training CP M rather than a fully-fledged system.*

Commodore Logo is first-class. It is written by Terrapin and may be the most full-featured Logo ever placed on a micro.

Questions

I have a PET 2001 with a graphic ROM, and a Gemini printer with an ADA 1800 interface. Is there a commercial or public domain program for downloading the PET graphic set to the printer? I know there is one by Cardco for the C-64, but what about the 2001?

Candy Jens
Wall, New Jersey

As a high school teacher, I have been considering some independent micro tutoring. The '84 educational activities-computer programs brochure from 'Educational Activities, Inc.' shows promise. True/False?

Ray Hendle
Innisfail, Australia

I have tried many places to find out how I can make my CBM display the characters necessary for foreign languages. (I am using a TEC 1500 Starwriter Printer.) This includes writing to many of the Commodore Computer Stores in French-speaking Canada and writing to those addresses in Commodore's Encyclopedia advertising the equipment necessary for this ability. In desperation, I asked my local computer store to help me, but alas, it all seems in vain. Has anyone got any helpful suggestions?

Clinton S. Cummings
40011 McDowell Cr. Dr.
Lebanon, Oregon 99355

Is there a word processor for use with French text which will work with a C-64, 1541 disk drive, 1702 monitor and Gemini-10X printer?

Warren Pollans
South Carolina State College
Orangeburg, SC 29117

I am looking for programs for our 4032/4040 system which would give us general ledger capabilities; data-base management; Bible research. I know that a lot of programs of these types are available for CP/M. Is it worth while, in light of Ron Kushnier's article, to buy a Z-RAM so that we could use these CP/M programs?

Rev. J. Paul Morris
Long Beach, CA

Is there anyone who has successfully modified the Macrotronics M650 RTTY + ASCII terminal program for 4.0 BASIC on the 8032 (not 4032)?

P. J. Rovero
Monterey, CA

I have a Commodore 64 and a 1650 Automodem. I have been using 'Autoterm/1650.c' and 'Term64.d' to access Compuserve, DJNS, etc. Ian Wright, in his review of the C-64 Automodem, seemed to have no trouble using this terminal software. However, I cannot figure out how to download data from these sources. My main interest is to download text from their encyclopaedia, news reports, etc. I have been logging on via DATAPAC in Winnipeg. If this software is not adequate could someone suggest a suitable one? I should add I have a 1541 disk drive but do not have a datasette. The program that came with the modem is on cassette and I can't use it.

Dr. S. A. Randeree
Emerson, Manitoba

I teach physics, mathematics and some electronics at university. After considerable evaluation of equipment and software I have purchased and am presently using the C-64, DATA 20 Eighty column plug-in and Gemini 10 printer for word processing. The word processor is from Protecto. In the sciences one often needs to type superscripts and subscripts, greek symbols, and mathematical symbols. Of course, descenders are also necessary for appearances sake. It would also be nice to be able to compose the work on the screen in 80 columns and print in exactly the same format. Does someone know of a system that will do all of this for a reasonable price?

Robert Speers
Huron, Ohio

I am a science/maths teacher at the local high school. Although I am using micros for introducing BASIC only at the moment, it is my intention to develop its business capacities (a 'course' available for the community), its micro tutoring (generally for younger people), and specific programming courses such as LOGO. I am also in the process of evaluating Multiplan, The Manager, and SuperBase 64 as business tools. No doubt there are members who have progressed beyond the early stage which I mention here. Perhaps I could receive some much needed assistance by referral to texts, software, ideas. . .

R. J. Hendle
P.O. Box 1072
Innisfail, 4860
Queensland, Australia

continued overleaf

PET Pals

I am trying to locate "Hairdressing School" as listed in the Commodore Encyclopedia of Software. Can anybody help?

Robert Everett
155 Castlefield Ave.
Toronto, ON M4R 1G6

I have a VIC 20 and a VIC-1525 printer. My printer always writes over the previous line. How can I fix it? Also, I need some help with telecomputing. I use the Vic-modem, with Victerm 1.

Liesel Radke
Box 1382
Rocky Mtn House, AB T0M 1T0

I would like to communicate with someone or some group with similar equipment to mine. I have an 8032 with an 8050 dual disk drive, and a Madison Z-RAM board with a Z-80A microprocessor, and a copy of CP/M version 2.2.

Stanley K. Johns
60 Guilford Lane, Ste. B
Williamsville, NY 14221

I am interested in trading ideas and the like with anyone who currently lives in a foreign country. A requirement is that you either speak English or Chinese as these are the only two languages I know.

Patrick Grote
11706 Devonshire Ave.
St. Louis, Missouri 63131

I am French-speaking and living in Australia. I would like to be in contact with any French-speaking Commodore 64 users.

Guy Coppens
Australian Computer Education Ass'n
P.O. Box 194
Corinda 4075, Australia

HAM OPERATORS

Name	Location	Call Sign
Canada		
R.E. Pettit	Brandon, MB	VE4OP
R.M. Johnston	Rothesay, NB	VE1EY
Michael R. Henwood	Moncton, NB	VE1BXV
William S. Townson	Ottawa, ON	VE3KIF
Maria G. Townson	Ottawa, ON	VE3KIP
Harold A. Dietrich	Waterloo, ON	VE3EJP
Rick Whitcomb	Verona, ON	VE3NWT
Ron Hall	Lakefield, ON	VE3CRT
Jack Millage	Lakefield, ON	VE3BLL
Ron Norris	London, ON	VE3FXX
Jack Becker	Downsview, ON	VE3NZJ or K4PBH
J. Allan Smith	Trenton, ON	VE3GI
Derek V. Stott	North Bay, ON	VE3KAA
Peter Hayes	White Spruce, SK	VE5ACY
International		
Peter J. Archer	Nelson, New Zealand	ZL2TYP
Juhani Leinonen	Turku, Finland	OH1FS
United States		
Cameron G. Pierce, Jr.	Palo Alto, CA	N6TQ
J. Carter Courtney	Lyme, CT	WA1FOK
Alfred Kornfeld	Manchester, CT	KM1U
Lyman Duggan	Tampa, FL	VE3ILW/W4
John McDowell	Crest Wood, IL	N9CVK
Dick Wasserberg	Lincolnwood, IL	K9RRF
Ronald MacIntyre	Newbury Port, MA	KA1LCZ
Carl A. Kolenda	Troy, MI	WB8BTY
John C. Sutherland	Livonia, MI	W8MOB
R. M. Stricker	Charleston, MO	W0UX
Robert McKinley,Jr	Tington Falls, NJ	W2OMR
Geoff Krauss	Latham, NY	WAZGFP
K.P. Perry	Kingsport, TN	N4DRG
Robert J. Hennings	Plano, TX	N5ECM
Michael Melum	Alexandria, VA	WA0CJI
Marvin Lowman	Inwood, WV	WB8NQB

CALENDAR OF TPUG EVENTS

Fall Schedule

ANNUAL BUSINESS MEETING — Thursday, October 4, at Leaside High School, Bayview & Eglinton Aves., at 7:30 p.m. in the auditorium. Regular members are voting members.

Please note: The exceptions to the "rule" for the designated date for a meeting (e.g. 2nd Thursday) are shown in bold.

BRAMPTON CHAPTER — Central Peel Secondary School, 32 Kennedy Rd. N., on the second Thursday of the month at 7:30 in the Theatre.

Thu. Sept.13	Thu. Nov. 8
Thu. Oct. 18	Thu. Dec.13

CENTRAL CHAPTER — Leaside High School, Bayview & Eglinton Aves., on the second Wednesday of the month, at 7:30 p.m. in the auditorium, for **PET/CBM**.

Wed. Sept.12	Wed. Nov. 14
Wed. Oct. 10	Wed. Dec. 12

COMAL GROUP — York Public Library, 1745 Eglinton Ave. W., (just east of Dufferin) on the last Thursday of the month at 7:30 p.m. in the auditorium.

Thu. Sept.27	Thu. Nov. 29
Thu. Oct. 25	Thu. Dec. 27

Commodore 64 CHAPTER — York Mills C.I., 490 York Mills Rd., (east of Bayview) on the last Monday of the month, at 7:30 p.m. in the cafetorium.

Mon. Sept.24	Mon. Nov. 26
Mon. Oct. 29	**Mon. Dec. 17**

COMMUNICATIONS GROUP — York Public Library, 1745 Eglinton Ave. W., (just east of Dufferin) on the first Wednesday of the month, at 7:30 p.m. in the Story Hour Room (adjacent to the auditorium).

Wed. Sept. 5	Wed. Nov. 7
Wed. Oct. 3	Wed. Dec. 5

EASTSIDE CHAPTER — Dunbarton High School, (from the traffic lights at Highway 2 and Whites Rd. — go north on Whites Rd. to next traffic lights — turn left to parking lots) on the second Monday of the month at 7:30 p.m. in Rooms 327 and 329.

Mon. Sept.10	Mon. Nov. 12
Mon. Oct. 15	Mon. Dec. 10

FORTH CHAPTER — York Public Library, 1745 Eglinton Ave. W., (just east of Dufferin) on the second Tuesday of the month, at 7:30 p.m. in the Story Hour Room (adjacent to the auditorium).

Tue. Sept.11	Tue. Nov. 13
Tue. Oct. 9	**Tue. Dec. 4**

HARDWARE CHAPTER — York Public Library, 1745 Eglinton Ave. W., (just east of Dufferin) on the first Friday of the month, at 6:30 p.m. in the Story Hour Room (adjacent to the auditorium).

Fri. Sept. 7	Fri. Nov. 2
Fri. Oct. 5	Fri. Dec. 7

MACHINE LANGUAGE CHAPTER (6502) — Call Jim Carswell at 416/531-9909 for additional information.

SuperPET CHAPTER — York University, Petrie Science Building (check in Room 340). Use north door of Petrie to access building. On the third Wednesday of the month at 7:30 p.m.

Wed. Sept.19	Wed. Nov. 21
Wed. Oct. 17	Wed. Dec. 19

VIC 20 CHAPTER — York Public library, 1745 Eglinton Ave. W., (just east of Dufferin) on the first Tuesday of the month, at 7:30 p.m. in the auditorium.

Thu. Sept. 6	Tue. Nov. 6
Tue. Oct. 2	Tue. Dec. 4

WESTSIDE CHAPTER — Clarkson Secondary School, Bromsgrove just east of Winston Churchill Blvd., (south of the QEW) on the third Thursday of the month, at 7:30 p.m. in the Little Theatre for **PET/CBM/VIC 20/Commodore 64.**

Thu. Sept.20	Thu. Nov. 15
Thu. Oct. 18	Thu. Dec. 20

Are you interested in organizing some other interest group in the Greater Toronto area? Please let the club office know, by mail, phone, or TPUG bulletin board.

TPUG Associate Club Chapter Meetings

CANADA

Edmonton Commodore Users Group
– meets at Archbishop Jordan High School, Sherwood Park, on the last Friday of each month at 7 p.m.
Contact Bob Kadylo 403-465-3523

Guelph Computer Club
– meets at Co-operators Insurance Assoc. on the 2nd Wednesday of each month at 7:30 p.m.
Contact Brian Grime 519-822-4992

London Commodore Users Club
– meets at Althouse College of Education, main auditorium, on the third Monday of each month at 7 p.m.
Contact Dennis Trankner 519-681-5059

Sarnia C-64 Users Group
– meets at Lambton College on the first Sunday of each month at 7:30 p.m.
Contact J. C. Hollemans 519-542-4710

Commodore Users Club of Sudbury
– meets at Lasalle High School in the cafeteria, on the last Thursday of each month at 7 p.m.
Contact Tim Miner 705-566-9632

PET Educators Group (Windsor)
– meets at Windsor Separate School Board Media Centre, 1485 Janette Ave. on the 3rd Wednesday of each month (not July & August) at 7 p.m.
Contact John Moore 519-253-8658

UNITED STATES

Boston Computer Society/Commodore Users Group
– meets at Minute Man Tech High School. Rt 2A (just off Rt 128) in Lexington, MA, every 2nd Monday of the month, at 7p.m.
Contact Harvey W. Gendreau 617-661-9227

Commodore Houston Users Group (Texas)
– **Clear Lake Chapter** – Nassau Bay City Hall, NASA Road #1, on the 1st Wednesday of each month at 7 p.m.
– **Central Chapter** – Farrish Hall, University of Houston main campus
– **NW Chapter** – Bleyl Jr. High School, 10,000 Mills Rd. (Cypress-Fairbanks SD), on the 3rd Thursday of each month at 7:30 p.m.
– **Klein Chapter** – Hildebrandt Middle School, 22,800 Hildebrandt Rd. (Klein ISD), on the 3rd Tuesday of each month (except July & August) at 6:30 p.m.
Contact Mary F. Howe 713-376-7000

Genesee County Area PET Users Group (Michigan)
– meets at Bentley High School on Belsay Rd. on the 3rd Thursday of each month at 7 p.m.
Contact Gordon Hale 313-239-1366

Greater Omaha Commodore 64 Users Group
– meets at South Omaha campus of the Metropolitan Technical Community College, 27th and Q Streets, in Room 120 of the Industrial Training Center, on the first Thursday of the month at 7 p.m.
Contact Bob Quisenberry 402-292-2753

Manasota Commodore Users Group (Florida)
– meets on the 2nd and 4th Thursdays of the month at 7 p.m.
Contact Roberto O. Bronson 813-747-1785

Michigan's Commodore 64 Users Group
– meets at Warren Woods High School in Warren, on the 3rd Tuesday of each month at 7 p.m.

Mohawk Valley Commodore User's Group (New York)
– meets at the Clara S. Bacon School in Amsterdam, NY, at 7 p.m. on the second Tuesday of the month.
Contact William A. Nowak 518-829-7576

Russellville CUG, Inc. (Arkansas)
– meets at Oakland Heights Elementary School on the 3rd Thursday of each month at 7 p.m.
Contact Bob Brazeal 501-967-1868

Sacramento Commodore Computer Club (California)
– meets at Kit Carson High School on the 4th Monday of each month at 7 p.m.
Contact Geoff Worstell 916-961-8699

Southern Minnesota Commodore Users Group
– meets at Mankato State University on the first Thursday of each month at 7:30 p.m.
Contact Dean Otto 507-625-6942

Westmoreland Commodore User's Club (Pennsylvania)
– meets at Westmoreland County Community College (Youngwood PA) on the 3rd Friday evening of each month
Contact Bob McKinley 412-863-3930

INTERNATIONAL

Baden Computer Club (West Germany)
– meets at CFB Baden-Soëllingen on the 2nd Sunday of each month at 7 p.m.
Contact Ben Brash

LASER PROJECTIONS

A Meeting Of Art And Technology

Photos by Roberto Portolese

Greg Lowry
Toronto, ON

Introduction

A dynamic art of pure projected light may have been one of mankind's ancient dreams, induced by the flickering campfire glow as it played across the rock walls of our ancestors' cave-dwellings. The constant flux of projected light forms must have been a powerful stimulus to the imagination and emotions of primitive man.

During the seventeenth, eighteenth and nineteenth centuries, optical theories and experiments performed by Newton, Goethe, Chevreul and others gradually laid foundations for the concept of a fine art of projected light. Later, impressionists concerned themselves with the technology of vision (albeit superficially). Their art reflected an interest in light and motion, and in the sciences which sought to explain them. Their philosophies gave rise to the notion that art might someday have as much to do with the traditions of science as with the traditions of art.

Traditionally, the arts and sciences have been considered mutually exclusive endeavors. The scientist, working within natural laws applied to hard devices and machines, had little to do with artistic experimenting as applied to ephemeral forms. Arts and sciences are so close today that there is an interplay of disciplines. Their convergence has been stimulated by the advances of our scientific and industrial technology, with its offerings of new and better means of generating and manipulating light, and novel methods of storing and processing information.

In the late 1960's a novel form of light display combined a variety of projection media — film, slides and overhead projectors. The "light show" flourished as a visual accompaniment to the emergent rock music of that era. Scientists and artists in those years began to speculate that the ancient vision might be realized by using one of the greatest technological innovations of the age — the laser.

Today, lasers are being widely applied in the sciences. The word "laser" has become associated with much "state of the art" technology such as nuclear fusion, holography, and data recording and transmission. Against this extensive utility, it is not surprising that a potential for art was recognized.

The Laser

The laser works somewhat like a fluorescent light bulb. An electric current is sent through a gas-filled glass tube. When enough power is applied, the gas begins to glow. However, in a laser, partially silvered mirrors at each end of the tube bounce the light particles back and forth until they pack enough power to shine through the mirrors. The result is an extremely thin high-energy beam of photons blasting tightly in the same direction.

In today's marketplace, there are a variety of ion lasers. The HeNe (Helium Neon) laser produces only one ruby red spectral line at a wave-length of 632.8 nm. A mixed gas KrAr (Krypton Argon) laser will produce several lines of varying intensity from red to ultra violet. A Krypton laser is most efficient in red. With soft coated optics, however, it can also reveal yellow/green and blue lines. An Argon ion laser has six lines within the blue/green portion of the spectrum.

Color Separation

A common means of gaining access to individual colors is to beam light from a Krypton or Krypton-Argon laser into a littrow or equilateral prism to extract spectral components. Separate beam colors can then be controlled with simple On/Off shutters, acousto-optics or beam intervention techniques.

Projection Methods

A) Scanned Image Generation

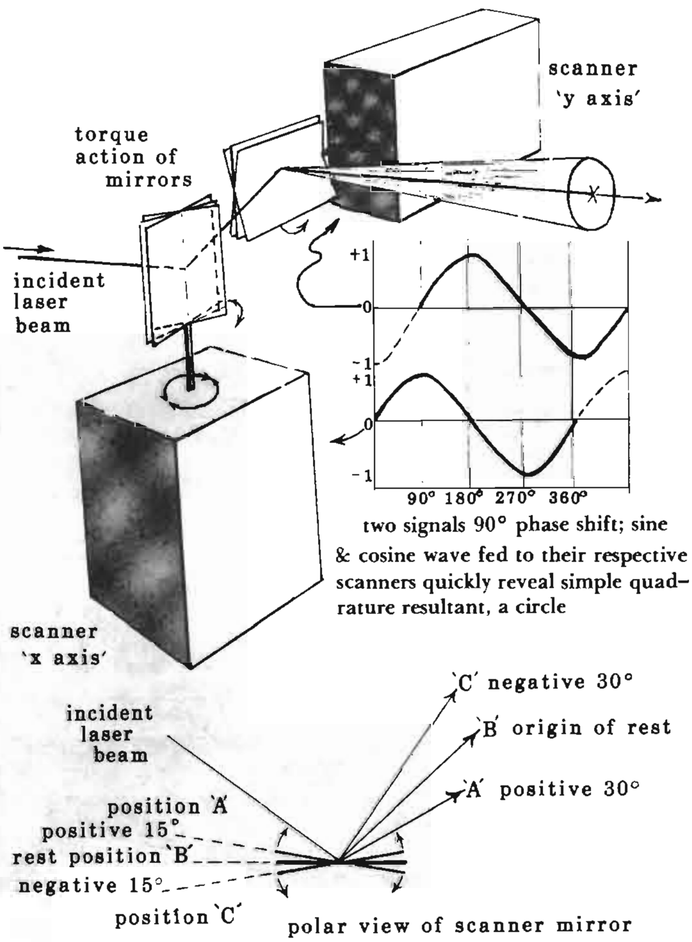
There are a variety of laser effects machines on the market which generate two-dimensional patterns through the use of dual axis galvanometer scanners. This electromechanical device rotates a small mirror which is attached to its shaft. In order to get two orthogonal (horizontal and vertical) axes, two scanners are positioned at right angles to one another so that the beam is reflected off both mirrors. Simultaneous movement of both scanners will move the laser beam in complex trajectories. In 'reality' at any given instant there is only a single dot moving so rapidly that the pencil beam (due to the phenomenon called 'persistence of vision') appears as a solid line image. *(See Graph 1)*

Open loop scanners are capable of covering sixty degrees of visual angle. Position feedback (closed loop) scanners use signals corresponding to both current position and velocity. These permit extreme accuracy in translation of input signal into mirror displacement. They do not suffer from the non-linearities and open-loop distortions associated with other scanners. A typical closed loop scanner obtains 2 KHz bandwidths over a wide scan angle and 3.5 KHz at lesser deflections. This means that the mirrors on the scanners can be oscillated at these frequencies.

If the feedback path is broken, and an appropriate computer and peripherals are placed therein, a memory of images becomes available, as I will describe later. As well, actual letter writing, logos and representational imagery are possible. Laser graphics require the design of a system where the bandwidth of scanners places the upper bound on the complexity of the imagery while still allowing for real-time transformations. Bandwidth limitation is due to the inherent mechanical limitations of the electro-magnetic galvanometer, such as the inertia of its mirror.

The most familiar method for image synthesis is accomplished through the use of standard electronic waveform generators operating scanners to produce Lissajous figures. For example, two equal amplitude sine waves of the same frequency (90 degrees out of phase with one another) form a circle centered at the origin. *(See Graph 2)*. This is the basic quadrature oscillator. More complex abstract designs, with a circular motif, can be achieved by summing a number of waveforms from oscillators tuned at harmonic frequencies, followed by control signals from a rotator circuit.

Graph 1

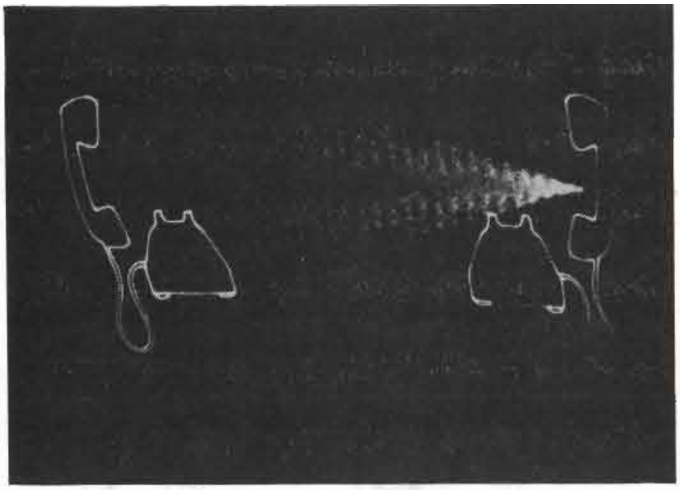


two signals 90° phase shift; sine & cosine wave fed to their respective scanners quickly reveal simple quadrature resultant, a circle

***note . . . the projection angle is twice the deflection angle relative to the incoming beam or plane of light**

Representational images can also be produced. The ability to enter hand-drawn graphics by bit pad, and to do frame animation by sequential accesses of stored points, immediately makes a digital system which is more versatile than an analog system. This technique of image design is very similar to sprite programming on the Commodore 64. See photos for examples of these images.

Animated sequences can be stored on magnetic tape with the use of an FM encoding system. The X and Y positions of the laser beam are stored as two signals on the tape. Once recorded, any projected shape can be repeatably reproduced whenever the tape is played. A third signal, in addition to the X and Y, blanks the laser beam. This makes it possible to display images with broken or discontinuous lines. If the magnetic tape also carries recorded music, on separate tracks from the animation signals, exact synchronization between the music and visual images is easily achieved.



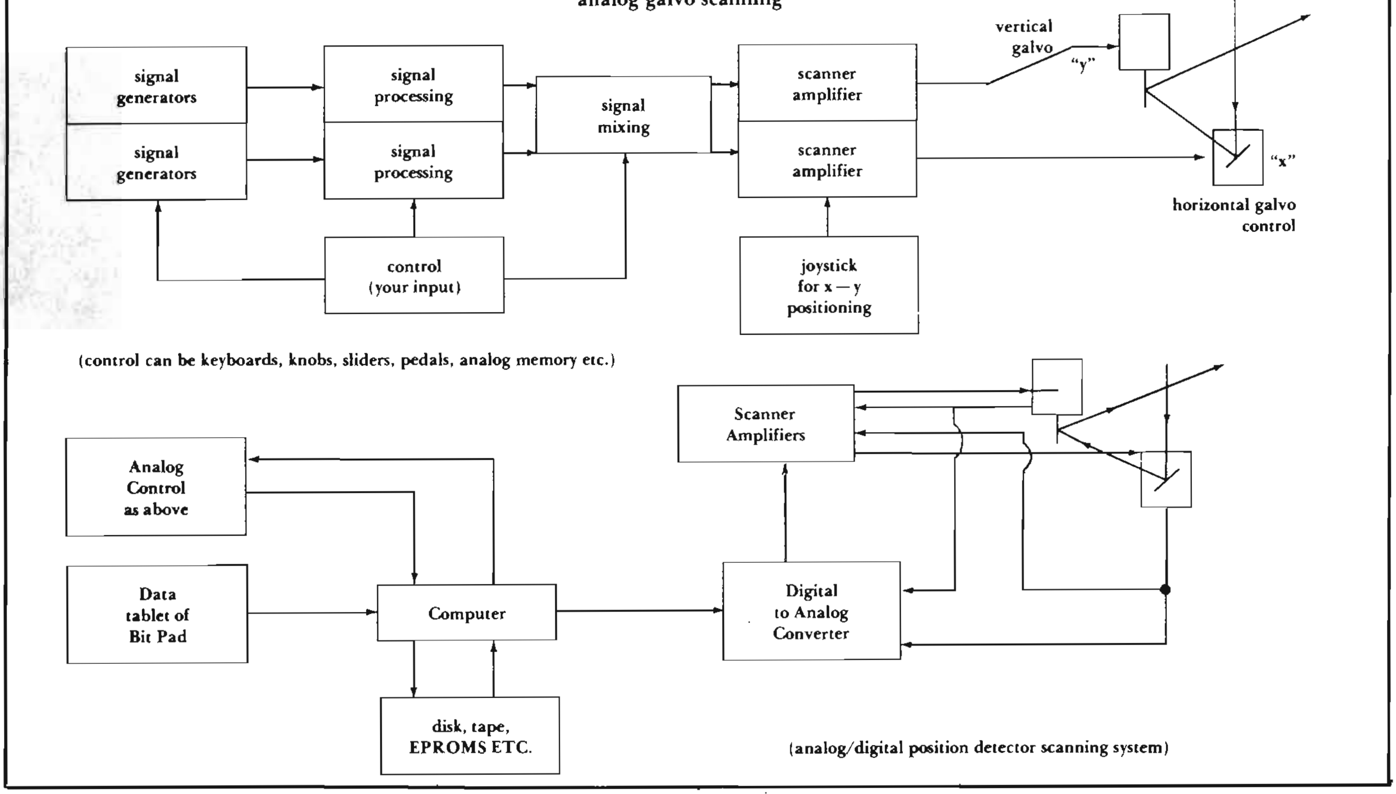
B) Raster-Scanned Images

It is possible to scan lasers to generate a raster which complies with N.T.S.C. standards. A rotating polygonal mirror provides the fixed-frequency 15.75 KHz horizontal deflection, while a precision scanner steps through the 60 Hz vertical staircase wave form. Combinations of red, green and blue allow for any color from soft pastels to vibrant primaries anywhere within the image. In the 1920's and '30's, when various methods of producing television pictures were being tested, this type of mechanical scanning was tried. At that time, lasers did not exist. Without them, mechanically-scanned images could not be produced with good definition, and this technique was discarded in favour of the cathode-ray tube. Nowadays it is being revived as one of several possible methods of producing very large TV pictures.

C) Non-Scanned Image Formation

Another method of image generation depends on transmission modulators which interact with the laser beam's phase coherence. I regard these as compositional instruments which are unlike any scan technique. This class of imagery may involve reflection, refraction, diffraction or interference dispersion. Translucent materials act to portray complex interference patterns such as galaxies, nebuli, clouds, and a myriad of others. Diffraction gratings also provide a technique of image multiplication with various densities and arrangements.

GRAPH 2
some system block diagrams
analog galvo scanning



Digital Techniques

Digital computer technology plays a part in several stages of the production of a laser display. For example, outlines of three-dimensional objects are often needed. These can be generated digitally by feeding a computer with a set of co-ordinates representing the three-dimensional shape of the object, then processing these to project a two-dimensional outline. This can be done repeatedly to portray the object in various orientations, and the sequences of images can be recorded on magnetic tape, as described earlier. When the tape is played through the laser projection system, an animated, moving image of the object is produced.

Digital storage can also be used, in place of magnetic tape, to hold the instructions for the production of images. A sequence of numbers, representing successive X and Y positions of the laser beam, is burned into an EPROM. The projection equipment can read these numbers back to produce the image. The system which I use contains many of these pre-programmed images, which I can call up whenever I need them.

Conclusion

A dry, technological description of the processes which are used in laser projection can do little to portray the artistic aspect of my work. In the creative process, I experience a sense of intimate connection, a human interaction with software and hardware. This linkage of the thought and emotion of the inner world with the high technology of the outer world is, to me, a revelation which creates a vision of a new reality. *TPUG*

(Editor's Note: The use of Commodore computers to drive lighting equipment similar to that which Mr. Lowry has described is a topic which I hope the Hardware Chapter will investigate next spring).

BENNETT'S TUTORIAL - Relative Files

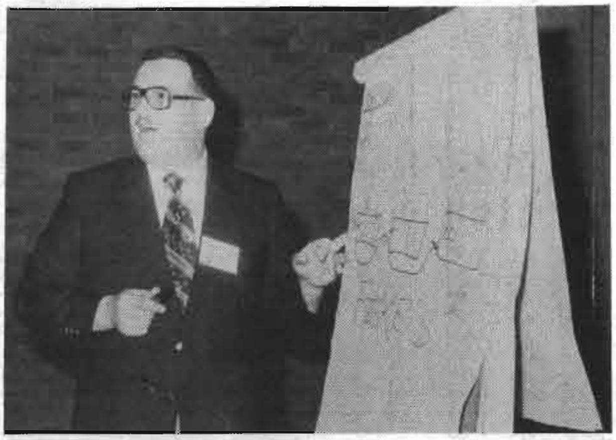


Photo by John Easton

Chris Bennett
TPUG Business Manager

One of the most common questions I am asked at meetings is: "How do I use Relative files?". Because of this, I presented two sessions on relative files at the Annual TPUG Conference in May 1984. They proved very popular. This article is basically a summary of what I presented at those sessions.

Let's first discuss the relative merits of sequential files versus relative files. Sequential files are easy to use and pack a lot of data in a small space. However, they must be read sequentially from beginning to end. If the data you want is near the end of the file, all information before it must be read and bypassed. If you wish to update information in a sequential file, you must either read it all into memory, update, and then write it all back out, or you will have to read the old file and write out a new file as you update the information.

Relative files (or direct access files) are not used this way. Once a relative file has been created, any record within that file can be read in, updated and written out. The one major restriction is that all records in a relative file must be the same length. This can cause a waste of space since you must set the record length to equal the largest piece of information that you will be processing.

In the listing included with this article is a sample program that creates, reads from and writes to a Relative File. It is written for BASIC 2.0 machines such as the VIC 20, Commodore 64 and BASIC 2.0 PETs. At the end, I will show you how to change it for the BASIC 4.0 machines (ie. 8032, 4032, PLUS/4 and Commodore 16).

The first thing you must do is OPEN the command channel (line 210) and the last thing you do is CLOSE it (line 370). The command channel is used to send commands to the disk drive and to read error messages from the disk drive. Closing the command channel at any time will also close all other files.

Using relative files is usually done in two parts. First, the relative file must be created. The number of records needed and the length of the records are specified in this creation process. The program logic needed to create 40 records of 100 characters each can be found in lines 1000 to 1110. The OPEN command in line 1010 sets up the parameters but does not actually create any records. The name of the relative file is TEST. The characters ",L," must follow the name within the quotes and the record length of 100 is specified by +CHR$(100) at the end of the OPEN. GOSUB 9010 is a subroutine that reads the error channel and returns the error number in the variable DS. Always check this variable after each disk operation. To set up 40 records, we must point to the 40'th record in the file and write something to it. Lines 1040 to 1060 set the record number to 40 and check the disk status (in this case, it must be 50). Lines 1070 to 1090 print data to that record and check that the disk status is zero. You will notice a short pause for a few seconds as the disk drive creates the space for those 40 records on your diskette. The file is then closed and we have finished creating the relative file.

Now we are ready to read or write to the file. The subroutine at line 2000 to 2190 gets some data from the keyboard and writes it out to the file TEST. Note the format of the OPEN command in line 2010. We no longer have to tell the disk drive the length of the records, as this is stored as part of the relative file. Line 2020 and 2030 read the error status and make sure that it is zero. Lines 2040 to 2100 get the record number we are going to write to, plus four pieces of data. Lines 2110 to 2130 set the record pointer. Line 2140 builds a string of the data we are going to write. Each field is followed by a carriage return C$ (set in line 230 at the beginning of the program). Lines 2150 to 2170 write the data to the record. A carriage return at the end of the record is not needed as the disk drive can tell where the end of a record occurs.

The subroutine to read is found at lines 3000 to 3220. The OPEN command is the same. In my example, I open and close the file for each read and write of a record. This is not necessary. The file should only be opened once at the start of processing and closed when everything is finished. The OPEN statements at lines 2010 and 3010 will open the file for *both* input and output. The IF statement at line 3120 is needed to check for a record that is empty. If you INPUT from a null record, you will get a value of CHR$(255) returned.

The one tricky part to relative files is the setting of the record pointer. This is done in lines 8000 to 8050. A set of six characters are sent to the disk drive over the command channel to tell it where to position itself. First, we send a lower-case 'p'. Next, we send CHR$(96+2) where 2 is the secondary address used in the OPEN. The secondary address is the third parameter and can vary from 2 to 14. If the open command were OPEN 5,8,4"... " then we would use CHR$(96+4). The third and fourth characters sent represent the record number. This number can be from 1 to 65535 and requires 16 bits (2 bytes) to represent it. The low order 8 bits first, followed by the high order 8 bits. The last values sent are the starting position within the record, followed by a carriage return.

BASIC 4 PROGRAMMING

If you have BASIC 4.0, you may make a couple of changes to simplify the program. First, the error channel is not needed (Delete lines 210, 370 and 9000 to 9020, plus any reference to GOSUB 9010). Replace line 1010 with DOPEN#1,"TEST",D0,L100 and replace lines 2010 and 3010 with DOPEN#1,"TEST",D0. Finally, replace lines 8010 to 8040 with RECORD#1,(R). *TPUG*

continued overleaf

```
100 REM ******************************
110 REM *                            *
120 REM *     RELATIVE FILES 2.0     *
130 REM *                            *
140 REM *             BY             *
150 REM *                            *
160 REM *       CHRIS BENNETT        *
170 REM *                            *
180 REM *                            *
190 REM ******************************
200 :
210 OPEN 15,8,15,"I0"
230 C$=CHR$(13): REM CARRIAGE RETURN
240 Q$=CHR$(34): REM QUOTE
250 T$="REC#:NAME:ADDR:PHON:MEM#:"
260 PRINT"❐";
270 PRINT"▩▩0 - EXIT"
280 PRINT"▩▩1 - CREATE FILE"
290 PRINT"▩▩2 - WRITE  FILE"
300 PRINT"▩▩3 - READ   FILE"
310 INPUT"▩▩SELECT OPTION    *▮▮▮▮";A$
320 IF A$="*" THEN 260
330 IF A$="1" THEN GOSUB 1010
340 IF A$="2" THEN GOSUB 2010
350 IF A$="3" THEN GOSUB 3010
360 IF A$<>"0" THEN 260
370 CLOSE 15
380 END
1000 :
1001 REM **  CREATE RELATIVE FILE  **
1002 REM **  40 RECORDS IN FILE     **
1003 REM **  RECORD LENGTH OF 100   **
1004 :
1010 OPEN 1,8,2,"0:TEST,L,"+CHR$(100)
1020 GOSUB 9010
1030 IF DS<>0 THEN CLOSE1: STOP
1040 R=40: GOSUB 8010
1050 GOSUB 9010
1060 IF DS<>50 THEN CLOSE 1: RETURN
1070 PRINT#1,CHR$(255);
1080 GOSUB 9010
1090 IF DS<>0 THEN CLOSE 1: STOP
1100 CLOSE 1
1110 RETURN
2000 :
2001 REM **  WRITE RELATIVE RECORD **
2002 :
2010 OPEN 1,8,2,"0:TEST"
2020 GOSUB 9010
2030 IF DS<>0 THEN CLOSE 1: RETURN
2040 INPUT"❐▩▩▩▩▩REC#  1▮▮▮▮";R$
2050 R=VAL(R$)
2060 IF R<1 OR R>40 THEN 2040
2070 INPUT"▩NAME   *▮▮▮▮";NA$
2080 INPUT"▩ADDR   *▮▮▮▮";AD$
2090 INPUT"▩PHON   *▮▮▮▮";PH$
2100 INPUT"▩MEM#   *▮▮▮▮";MM$
2110 GOSUB 8010: REM SET RECORD PTR
2120 GOSUB 9010
2130 IF DS<>0 THEN CLOSE 1: STOP
2140 F$=R$+C$+NA$+C$+AD$+C$+PH$+C$+MM$
2150 PRINT#1,F$;
2160 GOSUB 9010
2170 IF DS<>0 THEN CLOSE 1: STOP
2180 CLOSE 1
2190 RETURN
3000 :
3001 REM **  READ RELATIVE RECORD  **
3002 :
3010 OPEN 1,8,2,"0:TEST"
3020 GOSUB 9010
3030 IF DS<>0 THEN CLOSE 1: RETURN
3040 INPUT"❐▩▩▩▩▩REC#  1▮▮▮▮";R$
3050 R=VAL(R$)
3060 IF R<1 OR R>40 THEN 3040
3070 GOSUB 8010: REM SET RECORD PTR
3080 GOSUB 9010
3090 IF DS<>0 THEN CLOSE 1: STOP
3100 FOR I=1 TO 5
3110 : INPUT#1,D$(I)
3120 : IF D$(I)=CHR$(255) THEN I=5
3130 : GOSUB 9010
3140 : IF DS<>0 THEN CLOSE 1: STOP
3150 NEXT I
3160 PRINT"▩▩"
3170 FOR I=1 TO 5
3180 : PRINT MID$(T$,I*5-4,5),D$(I)
3190 NEXT I
3200 GET A$: IF A$<>CHR$(13) THEN 3200
3210 CLOSE 1
3220 RETURN
8000 :
8001 REM **  SET RECORD NUMBER     **
8002 :
8010 P$="P"+CHR$(96+2)
8020 P$=P$+CHR$(R AND 255)+CHR$(R/256)
8030 P$=P$+CHR$(1)
8040 PRINT#15,P$
8050 RETURN
9000 :
9001 REM ** READ THE ERROR CHANNEL **
9002 :
9010 INPUT#15,DS,E2$,E3,E4
9015 IF DS<>0 AND DS<>50 THEN PRINT DS,E2$
9020 RETURN
```

Important message to all BBS users

The TPUG BBS has moved again!
The new telephone number is. . .

(416)782-9534

Its operating hours are. . .
24 hours per day.
7 days per week.
The password is. . .

Humidity

THE BEGINNER AND THE DISK - Part IV

David A. Hook
Barrie, ON

We proceed with the series designed to address elementary disk handling techniques. **Part I** (June issue) looked at diskette care, formatting and initializing. **Part II** (July issue) introduced the DOS Wedge, disk error checking, and the BASIC 4.0 disk commands (found on PET/CBM, and also as part of VIC 20/C-64 "extensions"). Last issue, we concentrated on loading and saving programs, and mentioned "wild cards" and "pattern matching". This month, we'll continue the discussion of DOS (Disk Operating System) features with a look at VALIDATE and SCRATCH.

To demonstrate what is happening, we need an example, so let's use the following disk "directory" to work from:

```
0  "DATA DISK FOUR          " D4 2A
20    "ACCOUNT.A"              PRG
12    "ACCOUNT.B"              PRG
8     "ACCOUNT.C"              PRG
11    "ADD.CUST"               PRG
11    "ADD.INDX"               PRG
42    "ACC.CUST"               PRG
23    "ACC.INDX"               PRG
12    "RATAFRATZ"              PRG
11    "RATATATZ"               PRG
8     "ADMIN.A1"               PRG
16    "ADMIN.B1"               PRG
37    "STATS.PRG"              PRG
14    "STATS.TXT"              PRG
20    "STATS.INS"              PRG
13    "ADDITION"               PRG
1     "FILE"                   SEQ
0     "MESSED UP"             *PRG
405 BLOCKS FREE.
```

As a refresher, this was obtained with:

```
LOAD "$0",8     (directory is a PROGRAM)
>$0             (using the WEDGE utility)
@$0             (alternate WEDGE command)
CATALOG D0      (BASIC 4.0 command only)
DIRECTORY D0    (alternate with BASIC 4.0)
```

Only in the first example would you have lost the current program in memory. It was a LOAD command, remember! That's why you have become a devoted "WEDGER", isn't it? (See **Part II** for more evangelism on this topic).

Anyway, this is the "disk" that we will be using to show the effects of some of the DOS exercises. Refer back to the above directory to study what has taken place after each exercise below.

Validate

Occasionally this disk command gets described as "verify", which is unfortunate. Confusion with the BASIC command of this name would make my description futile. So, excise "verify" from your mind, and concentrate on VALIDATE. And don't think this is an obscure command that you will never need!

The directory above needs an application of the VALIDATE medicine. Where, you say? Look at the last entry, the one with the nasty "*" in front of the "PRG" designation. We have here an unclosed file, a ticking time bomb, as I may have said at least once before. Its file size is zero blocks—another flag that something is amiss. The attempted SAVE of the program failed, perhaps due to disk drive problems, a diskette flaw or a "DISK FULL" situation. With a data file, it's possible that you aborted the operation without closing the file. (Since we haven't mentioned data files yet, that's a topic for a future discussion).

In any case, until you have done something with the problem, you won't be able to SAVE any more programs on the disk, so you should be satisfied that fixing *this* problem is the first order of business. But how, you say? Why not just SCRATCH this program and be done with it? Well, for two reasons:

1. You don't know what SCRATCH is!
2. You don't want to cause *even more* problems with that diskette.

While the first reason is somewhat frivolous (and will be rectified later in this session), the latter *is* important. There is only one proper way to eliminate an unclosed file from a diskette: the VALIDATE command.

```
OPEN 15, 8, 15     (all machines)
PRINT#15, "V0"     (assume Drive #0)
CLOSE 15

>V0 or             (with the WEDGE)
@V0

COLLECT D0         (BASIC 4.0 only)
```

The letter "V" as part of a DOS command is an acceptable abbreviation for us clumsy folks. Only in the first case do you need to explicitly OPEN and CLOSE the "command channel" (see **Part I** of the series) to perform a DOS command.

I'll pause a moment while you rifle through your disk collection, trying to find that bum disk to work with... With our example disk, performing the VALIDATE takes a couple of moments (on the 1541). When the red activity light goes out, the process is complete. Fetching the disk directory now will show that "MESSED UP" has disappeared. It is entirely possible that the "BLOCKS FREE" line will show a different (and larger) number. This depends on what caused the original error. Here, the number doesn't change.

Anyhow, what's going on?

The VALIDATE command serves to "clean up" the disk BAM (Block Availability Map). As you recall, the operating system needs to keep track of which sectors have been used, in order to do its job properly. An unclosed file causes some sectors (of the "incomplete" file) to get "allocated" improperly. What we want to do is free up all the sectors first. Then look at all the "properly"-used sectors (those that are actually *part* of full programs and data files). Mark those as being used, or "allocated". Then calculate the new total that is available, and keep that one. So, only the sectors that *are* used are shown that way, and our unclosed file quietly vanishes into the night.

VALIDATE works by skipping through the pointers that "link" the sectors of a program together. If you choose to SCRATCH that unclosed file instead, there is a distinct probability that you can mess up a totally unrelated program. (For the technically-inclined only: the pointers in the unclosed file could link up to a chain from another "active" program. This causes sectors of *that* program to be "freed" erroneously). You may not discover the mess until sometime later, when you try to LOAD the affected program, and the computer just crashes on you. Hence, the ticking bomb label.

Amidst all this wonderfulness, there is a caution with VALIDATE: There is a certain type of data file, rarely used, that does not show in the directory. These "direct-access" files were used in the early PET days, before "relative" files were included in the DOS. A VALIDATE command done on such a disk will "free" the sectors where this information lies. Then a "write" to the disk will likely destroy the data right away. If you don't use commercial business software, it's unlikely that you will bump into this class of files. To be on the safe side, count the total number of blocks (free and used). If they total 664, you're not affected. It's a minor factor, but I thought it should be mentioned.

If you remember nothing else from this lesson: don't SCRATCH those unclosed files from a disk; VALIDATE the disk!

Scratch

Now that you have been converted (baptism available at additional cost), it's time to introduce this very useful DOS command. You may have a disk with scads of unwanted programs. You may wish to dispose of an old version, in order to SAVE the current version of a program – using the identical filename of the "old" version. Suffice to say that there is a frequent need to get rid of a file from a disk. This is where the SCRATCH command comes in. I think that most everyone feels a little uneasy when the time comes; perhaps some practice will ease the anxiety.

A little digression to deal with disk compatibility is in order. The various Commodore DOS types allow you to *read* disks that were prepared on 1540, 1541, 2040, 2031 and 4040 drives interchangeably. Unless you know *for sure* on which disk drive that disk was "formatted", you ought not to perform any *write* operation on it. This includes: SAVE, VALIDATE, SCRATCH, COPY, and RENAME (plus writing data to files with "PRINT#" instructions, which we haven't covered yet). When in doubt, don't do it – use a copy utility program. COPY-ALL and UNICOPY, both by Jim Butterfield, are the programs I use for this job. Part I of this series has lots of commandments like this that you can adopt as your very own guidelines.

(Editor's Note: There is a rumour that some 1541's write disks slightly differently than do others. To be on the safe side, don't perform any "write" operation on a disk unless it was formatted on the actual drive which you are using).

Let's refer to our example disk, and say that we wish to eliminate the program named RATAFRATZ from the disk. Here's the various ways it can be done:

```
OPEN 15, 8, 15            (all machines)
PRINT#15, "S0:RATAFRATZ"
CLOSE 15
```

```
>S0:RATAFRATZ        (with the WEDGE)
@S0:RATAFRATZ
```

```
SCRATCH "RATAFRATZ", D0 (BASIC 4.0)
```

The "0" designation refers to Drive #0 of a dual drive system. You may leave it out with a single drive unit, but it doesn't hurt to include it. Note that Commodore BASIC 4.0 has the SCRATCH command built into the BASIC language interpreter. Once again, the WEDGE sequence is shorter to enter, and hence easier to use.

continued overleaf

You can use pattern matching, but do so with extreme care. For example: in a moment's weakness you divine that scratching all files starting with "RATA" will do the job. So you try:

>S0:RATA* (WEDGE only)

With a large disk directory, you may not have noticed that RATATATZ was also on the disk, and a check of the error channel, with:

@ or (WEDGE only)
>

Gives the following message (if you did it this latter way):

01, FILES SCRATCHED,02,00

That signals that two files have been removed. Try not to outsmart yourself — I've made this mistake a number of times. (The comfort of being able to look inside the system and "unscratch" them was not a consolation for the stupidity).

Since the pattern matching and wild card features can be useful, let's take some examples from our disk. We wish to scratch the first three entries from the current directory. Here is a trick that I use to check before I perform the scratch, when using these features. Display the disk directory first:

LOAD "$0:ACCOUNT.?", 8 (AS A PROGRAM)

>$0:ACCOUNT.? (with the WEDGE)
@$0:ACCOUNT.?

Note that there is no version of the BASIC 4.0 CATALOG or DIRECTORY command that you can use to display a "selected" version of a directory. This was an error or an oversight in the interpreter design by Commodore. The designers of the BusCard (Batteries Included) and the C-64 LINK (Richvale Telecommunications) both rectified this in their firmware. If you don't have these "extensions", then you'll have to resort to the above techniques to do this.

The question mark in the filename is the "wild card" symbol, and means that all filenames starting with "ACCOUNT.", and followed by exactly one character, will be included. Display of the directory will show only the three files we want. Now slide the cursor back up to the directory command line (WEDGE version) and change the "$" to an "S" (and hit "RETURN"). When the cursor returns, the three files will be gone. The directory now looks like this:

```
0  "DATA DISK FOUR"     D4 2A
11    "ADD.CUST"          PRG
11    "ADD.INDX"          PRG
42    "ACC.CUST"          PRG
23    "ACC.INDX"          PRG
8     "ADMIN.A1"          PRG
16    "ADMIN.B1"          PRG
37    "STATS.PRG"         PRG
14    "STATS.TXT"         PRG
20    "STATS.INS"         PRG
13    "ADDITION"          PRG
1     "FILE"              SEQ
468 BLOCKS FREE.
```

Now, I want to get rid of the two eight-character file names ending with "INDX". These are the various ways:

OPEN 15, 8, 15 (all machines)
PRINT#15, "S0:????INDX"
CLOSE 15

>S0:????INDX (with the WEDGE)
@S0:????INDX

SCRATCH "????INDX", D0 (BASIC 4.0)

In the next case, I wish to scratch the program ADDITION. The directory command (WEDGE only):

@$0:AD*

yields:

```
0  "DATA DISK FOUR"     D4 2A
11    "ADD.CUST"          PRG
8     "ADMIN.A1"          PRG
16    "ADMIN.B1"          PRG
13    "ADDITION"          PRG
502 BLOCKS FREE.
```

This is definitely more than intended. With a single file, it's often clumsier to use wild cards and pattern matching. Just type in the whole filename, and be done with it.

For the final example, let's eliminate all the "STATS" programs. The customary directory command:

>$0:STATS*

gives:

```
0  "DATA DISK FOUR"     D4 2A
37    "STATS.PRG"         PRG
14    "STATS.TXT"         PRG
20    "STATS.INS"         PRG
515 BLOCKS FREE.
```

This is what we want, so cursor up to the WEDGE command, change the "$" to "S" as before, and the three will be gone from the directory. Our final directory looks like this:

```
0  "DATA DISK FOUR"     D4 2A
11    "ADD.CUST"          PRG
42    "ACC.CUST"          PRG
8     "ADMIN.A1"          PRG
16    "ADMIN.B1"          PRG
1     "FILE"              SEQ
586 BLOCKS FREE.
```

I hope that the above exercises have been helpful. Please forward any correspondence through the Editor. Your comments and suggestions for future columns are welcome. While the letters received thus far have been mainly from "advanced" users, don't be shy about elementary topics. That's what this series is supposed to be about. *TPUG;*

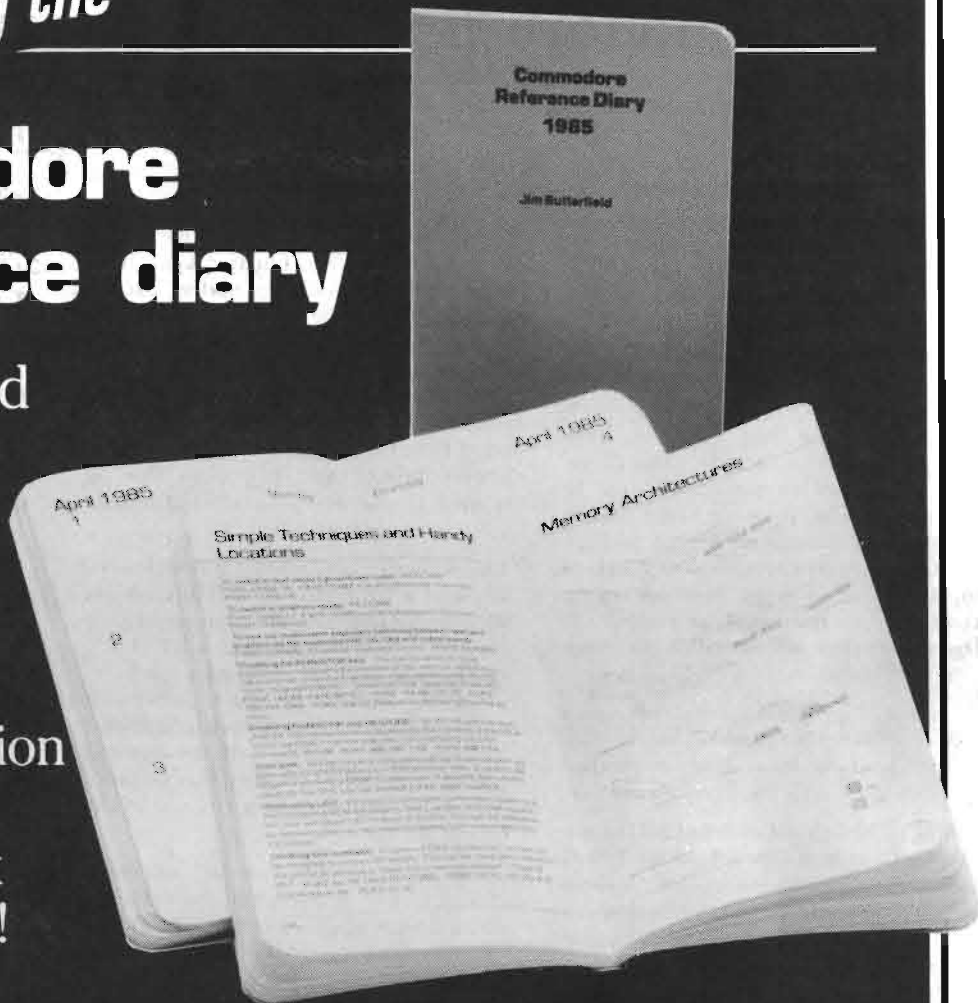
Note to Mr. T. Traub: your array example program worked perfectly for me, on both a PET and a C-64. I got no "FOR NEXT" errors, using the exact syntax you used. Sorry!

To Be Continued. . .

Introducing the

commodore reference diary

Jim Butterfield



...the answer for Commodore enthusiasts — a combination computer reference book and date book!

The **Commodore Reference Diary** is both a computer reference book and date book — the perfect combination for computer enthusiasts.

In a convenient, pocket-sized format, the **Commodore Reference Diary** features a whole week per page spread in a clear, uncluttered format allowing plenty of room for those important daily notations.

The unique computer reference section features a complete listing of major computer shows across North America and Europe for 1985, plus extensive computer programming material and terminology, written by computer expert Jim Butterfield, is included for the Commodore 64, Vic 20, PET/CBM Series, B-Series, and Commodore 264 and 364. Memory lay-outs, screen codes, useful short programs, the machine language instructional set, kernal subroutines, and much more!

141565 **$6.95**

To order, fill out this handy coupon and return to: TPUG Magazine, 1912A Avenue Road, Suite 1, Toronto, Ontario, M5M 4A1, **(416) 782-1861**

Please send me ___ copy/copies of the *Commodore Reference Diary* 1985, 141565, at $6.95 each.
Add 5% for shipping and handling per order.

I enclose________Cheque________Money Order (No COD's or cash),

Charge________VISA,________Mastercard.

Card No. Bank Expiry Date

Signature

Name

Address

Province Postal Code

Price is subject to change without notice.

Copp Clark Pitman Ltd., 495 Wellington Street West, Toronto Ontario M5V 1E9 (416) 593-9911

The Commodore B-128 Computer

Jim Butterfield
Toronto, ON

Commodore's B series computers have been coming for a long time. Now they have arrived – but not in Canada – and promise to have a brief but glorious lifetime. In the United States, they are being sold as part of a very attractively-priced package by Protecto Enterprises (Box 550, Barrington, IL 60010). In Europe, there are a number of them on hand (they are called 700 series there); their sales situation isn't clear to me at this end.

The B briefly emerged about a year ago; a number of machines leaked out to consumers. There was also a C machine, recalled by Commodore, which will not be available.

Because of the unavailability of B-128 computers in Canada, it may be hard for TPUG to act as an information centre on these machines. Programs and information can be circulated as available, but no facilities will exist to check these in Toronto.

B-128 Hardware

The B-128 is low-slung and stylish; it requires an external monitor. The screen is 80 columns by 25 lines, similar to the 8032, 8096, 8296 and SuperPET. No color or high resolution video is available. The style is still that of a Commodore machine, with screen editing taking place in the same way.

Connection to disk and printer is via an IEEE-488 bus identical to that of PET/CBM. Standard Commodore IEEE peripherals attach normally. Cassette tape software is not in place, making the B-128 a disk-oriented system. There's an RS-232 interface, using the standard RS-232 connector. With an ACIA chip built in, good communications can be achieved.

The 6509 processor is a close cousin of the 6502. It can access "banks" of 64K memory via special registers mapped at addresses 0 and 1, allowing 128K of memory (or in principle, even more) to be accessed.

The Keyboard

There's a big keyboard, complete with a numeric keypad and function keys. It's a change from previous Commodore layouts, and very close to the standard ASCII form. There are lots of keys. Separate CTRL (Control), ESC (Escape) and "Commodore" keys make special functions easier to implement. Cursor movement keys are separated from the main keyboard, and there are now four of them, so that no "shift" is needed. Display and program control keys are also neatly segregated.

The numeric pad includes a double-zero key and a CE (Clear Entry) key, which wipes out an entire number you might have entered. There's also an extra RETURN key marked ENTER. There are ten function keys, which can be used together with the SHIFT key to generate twenty functions. The first ten are predefined; the moment you turn the power on, they may be used. Any of the twenty can be re-defined by the user at any time.

There are twenty-six "escape" sequences (ESC, followed by a letter of the alphabet) which do useful screen configuration jobs such as setting up windows.

Big BASIC

BASIC is upwardly compatible with previous Commodore BASIC implementations. All previous BASIC commands and functions – including the 4.0 disk commands such as CATALOG, DLOAD and SCRATCH – are in place. Many new language features have been added.

But BASIC programs saved from a B-128 may be difficult to load to another Commodore machine, due to "zero page chaining". The relocating loader on the C-64 thinks the first line is the end of the program. This difficulty may be resolved wiith a simple utility program; for example, **MERGER** from the TPUG library or from *COMPUTE!* October 1983, page 144, will do the trick.

PRINT USING (and the associated PRINT#n USING) allows information to be printed neatly. For example, PRINT USING "#,###,###.##";9876.789 will cause 9,876.79 to appear neatly aligned within the space provided.

Memory is arranged in "banks" of 64K each. You may pick a bank with the BANK command; this will affect commands such as PEEK, POKE, SYS, LOAD or SAVE; but you'll need to know what you're doing to find your way around in this vast memory. Machine language "boot" programs will use BLOAD and BSAVE commands, together with complex "transfer sequences", to allow the various banks to intercommunicate.

Error trapping is implemented to allow the programmer to catch problems as the program runs. Strings are 4.0 type – in other words, there's no problem with garbage collection delays.

BASIC is generally noticeably faster. Between the faster clock speed of the 6509 chip, the IEEE bus and the elimination of the garbage collection, users will find this a speedy machine.

There's a reset switch at the back of the machine which does a "warm start"; in case of emergency, you'll be able to recover your program.

The 6509

The 6509 chip can address more memory – in theory, up to over a million bytes. But it works much like a 6502, which has addresses that can reach only 64K of memory. How do we get all of memory? By "bank selection" – a process by means of which the computer can select one of several 64K slices, or "banks".

Bank selection is a machine language feature, and comes in two flavors. The easy part is data selection; we can change how the indirect load and store commands work. The two commands LDA (xx),Y and STA (xx),Y have been modified slightly so that they load data from or store data to a selected bank of memory. The bank is selected by the contents of address 1.

It works this way. If, in machine language, I store a value into address 1, the LDA, indirect indexed, and STA, indirect indexed, will now draw data from whatever bank I have specified. For example:

```
LDX #$02
STX $01
LDA (. . . ),Y
```

will cause data to be loaded into the A register from bank 2, since we have stored a value of 2 into address 1.

To make a transition to a new bank, using address 0, we need very careful coding that is "synchronized" between the new bank and the old one. This coding is available — it's called "transfer sequences" — it handles all the details so that you don't need to know all the technical trivia that makes it all work.

Software Availability

Because the B-128 is likely to have a much smaller user community, programs written specifically for this machine will be in limited supply. But many BASIC programs will load directly from other Commodore machines without any need to be adapted. Programs with machine language content, however, are unlikely to run without significant change. Many utilities will be initially unavailable; it seems likely that special interest groups of users will have to develop them for themselves.

There is some custom software available; what I've seen of it looks good. I'd advise users who think they might need the software to buy it now, since it might become hard to get.

The B-128 contains a machine language monitor with extra features designed for the new architecture. Adding extra monitor features, such as a tiny assembler, is possible; but it's not a simple adaptation of existing monitor extenders such as SUPERMON, since the MLM has a different style.

Information Sources

Commodore is rumored to be preparing a reference guide for the B-128; if so, that will be a valuable source of data.

I have published memory maps of RAM and ROM for the B-128 in *The Transactor* magazine. The Great Reference Issue is long since out of print, but it will be published again soon. (*The Transactor,* 500 Steeles Avenue, Milton, Ontario, Canada L9T 9Z9).

There's a brief description of the B-128, plus an appendix giving a RAM memory map, in my book *Machine Language for the Commodore 64 and Other Computers* (Brady Communications, Bowie, MD 20715, $12.95). The B-128 is something of an exception in the Commodore line; in the book, I suggest that it's not the best machine for beginners to use when learning machine language. Even so, the map and description is there; and in the disk which optionally accompanies the book, I give an example of a B-128 program complete with boot and transfer sequences.

My "Commodore Diary 1985" (Copp Clark Pitman Ltd., 495 Wellington Street West, Toronto, Ontario, Canada M5V 1E9) is space-limited; but I did jam in an "architectural sketch" and an abridged memory map of the B-128.

B-128 information is a problem area. It seems likely that the total population of this computer will be less than ten thousand (as compared to millions of VIC 20's or Commodore 64's). As a result, there's limited profitability potential for books or software written specifically for this machine. Users will need to take care of themselves more.

Yet it's part of the Commodore family, with many features in common with other machines. And I feel sympathetic towards users and potential buyers who want more data.

Bottom Line

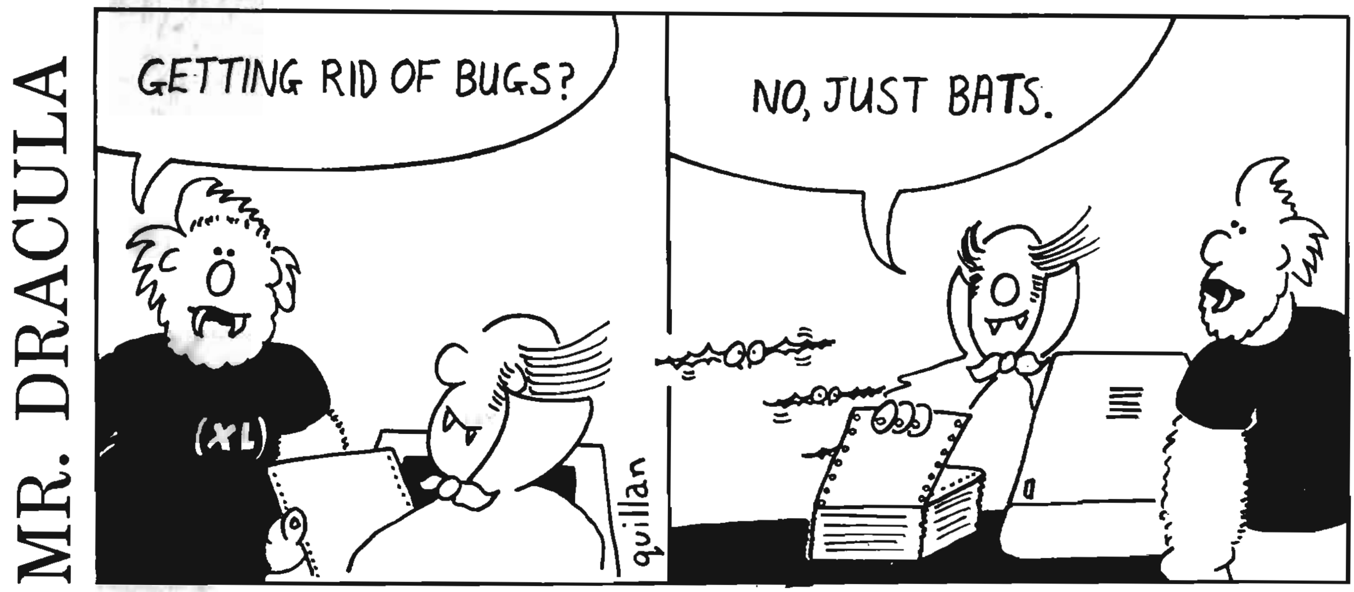
If you rate the things you like about a computer, the B-128 comes high on the list. It looks good. It has a fine keyboard. It's fast. It has lots of memory. Its BASIC language is very good. The B-128 has many convenience features, from screen controls to the reset button.

But there won't be many of them. A B-128 user will be something of an orphan — left to his or her own resources for many things. There won't be much software out there; there won't be many books.

Many BASIC programs will adapt nicely. Machine language programs will mostly need some re-chopping . . . at least, if you want to take advantage of the extra memory space.

The B-128 is late. When it was first shown, IBM wasn't seriously entrenched in the personal computer market. Now we have different concepts of what a "big" machine is, and the B-128 has lost some of its novelty and glamor.

It's a good machine at an astonishingly attractive price. But you are on your own a little more than you would be with other Commodore machines. Is this the machine for you? Could be. . . if you're a little more independent than the average user. *TPUG*



C-64 M/L Development System

J. Allan Farquharson
Paris, ON

MICOL SYSTEMS is offering the user a chance to develop machine language on the Commodore 64 computer. Included you will find an editor, assembler and a monitor. An editor is usually a simple word processor in essence. The assembler is the program which puts together the machine code made from the mnemonic code which most programmers use to write longer programs. That sounds involved, but if you write in BASIC or another language you know that subroutines and other code are put together to produce the program which collectively will do the job that it is written to do. Since machine code is rather lengthy, but very rapid, many subroutines are usually kept to do all the little jobs you wish to do over and over again. The monitor is a machine language program which allows one to look at various parts of memory in the computer and allows one to manipulate the code found there. Older Commodore machines had a monitor built-in, but the C-64 comes free of any such handy device.

Before I am flogged to death, I hasten to mention that VIC 20 owners too may use this system. Most systems do not have a manual which is a tutorial on how to write computer machine code. The manual provided is no exception. It is thorough, but not a complete explanation for those who have never written any machine code before. Many books abound on the subject. You may wish to use both another text of your choice and this system, if you are a beginner. Seventy-three pages long, the manual is reasonably complete. I must admit to a bias which says that manuals should be standard 8½" X 11" size and in pica-size print. This one is smaller, but the print is readable.

Both cassette and disk owners may use this assembler. I suspect most will turn to disk before long because it is more convenient, especially when storing various routines for later retrieval.

The disk may be backed up. I recommend you do so, and use the original for backup and a copy for daily use. Never begin without some backup. Having no useable copy is no fun! By the way, the system assumes you use a single disk drive. (Some people, like myself, use dual drives through an interface to the C-64).

Once loaded into the computer, you may work without loading other programs into the computer separately. A menu takes over this chore for you.

This system offers you some excellent facilities to write code. Generally a person writing any lengthy material will want to keep on file his/her "source file" which will provide comments on what each routine is doing and by which the code may be changed or modified at a later date. The editor will permit one to load the code and change it at will. "Object Code" is awkward to work with as it must be disassembled to make sense. Changes in code produce an immediate crash, most times.

Among the handy dandy features are: conversion from binary to hexadecimal, hexadecimal to decimal, and so on. The text editor allows one to add lines, delete lines and copy lines. A textfile may be loaded from and saved to both disk or tape. One may examine the disk directory, find a string in a file, replace a string with another string and list part or all of a file.

A MEM command tells one how many bytes are used in a file, number of lines defined and the file name, if any, on disk. Disk commands may be sent to the disk from the editor. This is convenient. Listing the code to the printer is easy.

Strings may be replaced by other strings. This allows many occurrences of a single item to be replaced throughout the whole source code.

The usual format is observed for assembly. A line begins with a label, if used, followed by the op code, address field and then comments, if any. Pseudo-codes are available. Where a program is long, the computer may get files from disk by a CHN (chain) command. BYT allows the programmer to insert numbers in decimal, hex, octal, or binary. This simplifies entry without looking up the binary values.

Conditional code may be written using an IF. . . ELSE construct. You might want different code for, say, a 22, 40, or 80 column machine. Another idea is EXP, which allows you to expand the parameters for a pseudo-op which may vary in different places. Equates are available to set fixed memory locations.

One can list to the screen or printer during assembly; or neither, for faster operation. One may nest macros. This is a nice touch. Assembly may go to memory or disk. One need not worry if the same address is specified for the code in memory as is occupied by the assembler. It writes somewhere else during assembly. I remember one system I once used. It wrote all over itself. Needless to say, this didn't work for long!

Portions of memory may be reserved for later inclusion of a number of bytes by a command. One may wish to insert a special table somewhere at a later date. This function then would be useful.

ASCII data is generated from the ASCII letters directly, when between quotes in one command. This is terrific! Now you may write some clever words to the screen and not have to figure out the hex code for each number or letter. Let me see. . . P is $50 and R is ??? You just write it: 'PRINT YOUR NAME' and out comes the code! Very nice. Good thinking went into this assembler. I could get to like it very much.

For dummies like me a handy command, WOR, allows me to place a two-byte hex code in the machine and it writes it out, low byte, high byte. $FFD2 then becomes D2 FF in the order which the 6502 needs it. Not essential, but a nice touch. Incidentally, a comment may be added to a line by leaving a space after the code. No need to write a semi-colon. Good for lazy people.

I am glad that I never make errors. My code is always perfect. Perfectly awful, that is! Errors are printed for everyone to see at the end of the assembly. MICOL SYSTEMS uses a two-pass assembler. This is usual for most assemblers I have used. The first pass writes 1's on the screen, pass two writes 2's. A symbol table may print at the end, if desired together with the related addresses. These are very useful. All I know is that they save a lot of sweat.

Source files are stored on disk, or tape, then called in when the assembler begins its job. After assembly you are informed of the total number of errors and the number of lines processed.

A feature of this monitor, which is lacking in some, is the inclusion of a disassembler. This is neat to look at segments of code to see what is going on. I know some smarties read the hex code directly. But then I never claimed to be Frank N. Stein. As with the PET, you can load and save from this monitor. TRACE and SINGLE STEP are both included. Code may be moved to another location as well. This includes a relocation facility. Most moves will crash without reworking them on relocation.

Very handy is the next idea: To debug code, you can set break points. This stops the program where you wish so that you can try segments of code. In addition, you will see all registers, flags and counters when you "execute" this. Sometimes the flags are set such that some code will not run properly. Seeing the flags is an aid to sanity.

I don't like the colour choice on the screen: this is the usual default value when the C-64 is turned on. I am sure this could be changed. I like a bit more contrast, myself.

If you want to find text in some code, an inspect command allows you to look through the hex code and locate text; it is displayed as the character instead of the hex code! Sometimes looking for ASCII code can be a pill. This makes it easy to spot.

Code may be run from the monitor. Make sure you have a file copy somewhere in case it crashes! I know someone that forgot this once. You will too. So save the code in a file first.

My reaction to this assembler is positive: I like it. I understand the manual. A first-time user may flounder a bit, but the manual is clearly written in a forward manner. I once read a manual for an assembler. On page 38 it said: "now to begin." I begin on page one, myself.

What I see as lacking in an otherwise excellent system is a short tutorial to get one started. A few practical examples of a typical assembly would enhance the manual greatly, especially for beginners. The concept of a macro is hard for some to grasp. Macros are useful; one needs to get the hang of them. I think the remedy is to include a few print-outs of typical assembly code from a source file. This was not done. I hope future issues will include several examples. They could be included as files on the disk. The manual could mention them and tell how to get to them and execute the assembly.

Again I highly recommend this product. It is sold by: *Micol Systems, 100 Graydon Hall Drive, Suite 2301, Don Mills, Ontario, Canada M3A 3A9.*

Ask for their **MACHINE LANGUAGE DEVELOPMENT SYSTEM.** For VIC 20 and C-64 – tape and disk. *TPUG*

MICOL REPLIES

We wish to thank Mr. Farquharson for his review of the Micol Systems' Machine Language Development System, Commodore 64 version.

We, at Micol Systems, listen with great interest to feedback from our customers and critics. It is for this reason we have just released Version 2.0 of the above-mentioned package. In keeping with Mr. Farquharson's suggestions, we now include on the disk an example program which is lavishly commented. The user is invited to study, assemble and run this program to give him/her an even greater insight into the development of assembly language on the Commodore 64.

In addition, it is now possible to change the colour combinations of the screen to any of fifteen colours the user might desire.

These are only two of the many improvements we have included in version 2.0 for the Commodore 64 disk **only**. *An excellent product is now even better. Any purchaser of the original version may obtain Version 2.0 by sending us $10.00 U.S. ($13.00 CAN) together with a copy of his receipt to our Canadian address.*

Menu Handling - Part II

John Easton
Toronto, ON

To illustrate the concepts involved in menu creation, I am going to take a walk through a simple menu-driven program or two. For example, let's decide that we're planning to provide a variety of programs to a mixed public audience. Furthermore, our director wants some statistics, to back up the effectiveness of this demonstration. For instance, he wants to know what age groups are interested in what programs, where these persons may be from and perhaps the background of their interest — all the stuff that statistical reports are made of. The following program excerpts are from various programs demonstrated at the Annual TPUG Conference in illustration of the use of menus and statistical tracking of such use. I believe that programs similar to these are available on a **Speakers' Disk** — only the line numbers have been changed to protect the innocent.

BOOT

Let's start with a basic BOOT program. Regardless of where you might like to go in the demo stuff, should you be required to keep statistics, it will be necessary to clean up an area in memory to store such statistical stuff — thus the BOOT program. This program (traditionally the first one on the disk) checks to see what machine we're currently running (basic to any BOOT program) and then in this case prepares a Statistical Notepad in a convenient cassette buffer. Finally, it asks for the statistical background material requested by our statistics department.

On completion of its tasks it then conveniently loads the Main Menu for us, and things proceed merrily from there till (hopefully) our satiated user says he's had enough — at which time our Main Menu program saves these important statistics to disk and we arrive back at the BOOT to wipe the slate clean and start again. I say *hopefully* because it is our experience that most casual passers-by will hit a key or two, maybe select something from the menu — but invariably, when they've had enough, will merely walk away, leaving a rather cryptic message for the next user — PLEASE SELECT YOUR CHOICE ARNOLD! A timing loop to intercept a pre-determined time of null response (indicating that Arnold has gone for coffee) I leave to you. Such a loop should be capable of closing everything down in the Statistics department, as far as Arnold is concerned, and returning to the BOOT to await the next happy customer.

Rather than reproduce a tedious listing of typical menus, why don't I just point out various conventions and aids that we've come across over the years to make this chore a little less formidable? For actual examples of programs, I would suggest you refer to the **Speakers' Disk** for the May Conference — I believe most of the example menu formats are reproduced there for your use and adaptation.

So let's just pick up some of the program lines as we go — and I'll explain them as we walk through a typical program:

10 CL$ = CHR$(147):

It's usually a good idea to clear the screen before inflicting strange Upper/Lower Case And/Or Graphics Modes upon your unsuspecting user. I prefer to define it in this manner because it's more obvious to some stranger who might be attempting to translate your CBM BASIC to run in his TRS or Apple. CL$, for instance, looks very similar to the command CLS in TRS BASIC — and will now operate in a similar manner. The equivalent in Applese is CALL -936: but then, we all have to put up with *some* inconvenience.

15 POKE 50003,64:C = PEEK(50003):IF C=64 then 80:

Establish machine version by POKEing to an area that is ROM in anything but a C-64. If the POKE 'takes' it must be a C-64. Incidentally, this particular PEEK will also tell you which ROM version on the GREEN MACHINE you may be working with:

IF C=0 THEN ROM=1, IF C=1 THEN ROM=2, IF C=160 THEN ROM=4
35 IF PEEK(213) = 39 THEN 55:
Check for 80-column machine (an 80-column screen will return a value of 79)
40 PRINT CL$;"(8 cursor-down)", "(reverse) Please re-load after I have changed screens "
45 LOAD "CBM *",8
Load Chuan Chee's **Screen Scrunch** program to force a 40-column screen, after which the simplest thing is to re-start the program, and since we now have 40 columns, the program will skip this portion of the program.

GREEN MACHINE SETUP ——————————
55 PRINT CL$:POKE 59468,14
Clear screen and set upper/lower case
60 POKE 42,0:POKE 43,41:CLR:
Set END-OF-BASIC pointers higher than any of the following programs. This, in effect, will allow program chaining and parameter passing since, from this point onward, your machine will think it is running the same program (the largest one on the menu)

61 C = PEEK(50003):
Re-set 'C' after clear
65 IF C = 160 THEN KEY=144:EN=85:DI=85+3:POKE KEY,DI:
Disable stop-key for ROM 4.0
66 EX = 64721:IF C = 160 THEN EX = 64790 :
Breakout and machine re-set (ROM 2.0 & 4.0) — note this is just a way out so that, when necessary, you *can* break out of the program, even though you've bulletproofed it thoroughly.
67 GOTO 110 (skip the C-64 stuff)

C-64 SETUP ——————————————
80 PRINT CL$:PRINT CHR$(14)
Clear screen and set upper/lower case
81 PRINT CHR$(5):POKE 53280,6
Set white characters on blue background — the simplest C-64 text format in my rather limited C-64 experience.

85 POKE 45,0:POKE 46,45:CLR:
Set END-OF-BASIC pointers high — you will note that this POKE (to location 46) is 4, or a value of 1024 (4*256) higher than the equivalent POKE in a GREEN machine — and that's the difference in the location of your BASIC programs.

90 C = PEEK(50003):
Re-set 'C' after clear
95 KEY = 788:EN = 49:DI = EN+3:POKE KEY,DI:PRINT CHR$(8):EX = 64738:
Disable stop and reset keys and establish breakout location.

110 FOR I = 834 TO 907:POKE I,0:NEXT
If you're attempting to keep statistics, here's where you clear a bunch of "pigeonholes" sufficient in number to track all the data in which you might be interested for the next pass through the programs. This space is in a cassette buffer and since we're not planning to use cassette, it's a handy spot for our "memo pad". *Make sure* that any such location doesn't interfere with any fancy stuff that might be included in following programs. *(Editor's Note: In PET/CBM's, using this part of the second cassette buffer is incompatible with use of BASIC 4.0 disk commands).* The safest buffer for CBM owners would be in the 'First Cassette Buffer', with starting address of 634. Maximum bytes in this casette buffer is 192.

Now we set up a few housekeeping-type variables to allow us simpler code routines in the subsequent programs. You will note that those defined variables we wish to preserve are concatenated with a null ("") to force them into upper memory, away from the destructive influence of program overlays.

120 BL$ = " (39 spaces) " +""
125 LN$ = " (39 half—graphic key) " +""
CHR$(249) or similar
130 SB$ = "(rvs) Press Space-Bar to Continue " +""
135 RK$ = "(rvs) Press 'RETURN' Key when finished " +""

All these 'prompt-type' messages are set to 39 characters to bypass any possible line-wrap problems.
140 IF C = 160 THEN BEL$ = CHR$(7)
Ring bell on CBM machines used as a prompt (usually to signal incorrect input). I'll leave the equivalent C-64 routine to you C-64 experts.

And finally, we'll set up vertical tabs in a manner similar to that found on the Apple—greatly simplifies screen formatting and prompt locations.

145 VT$ = "(home+23 cursor-down)"
150 DIM VT$(24)
155 FOR I = 1 TO 24 : VT$(I) = LEFT$(VT$,I) : NEXT
Your VT$() variables will automatically build into upper memory.

OK—now that we've set up things, we might as well stick our simple subroutines right near the beginning of each program. NOTE : don't forget to bypass them in the main body of your program with a line like:

160 GOTO 1000

continued on next page

We always need a simple GET or two -

```
************* Yes/No **************
200 GET A$:IF A$ > "" THEN 200
210 GET A$:IF A$ = "Y" THEN RETURN
220 IF A$ <> "N" THEN PRINT BEL$:FOR I=1 TO 100:NEXT:GOTO 210
230 RETURN

*********** Space-bar *************
300 PRINT VT$(24);SB$;
310 GET A$:IF A$ > "" THEN 300
320 GET A$:IF A$ = CHR$(216) THEN SYS EX
*** Note - a simple way to break out with a SHIFT/X character
330 IF A$ <> CHR$(32) THEN PRINT BEL$:FOR I=1 TO 100:NEXT:GOTO 320
340 PRINT VT$(24);BL$;
350 RETURN

** Double-Get (any number to 99) **
400 GET A$:IF A$ > "" THEN 400
410 GET A$:IF A$ = "" THEN 410
420 IF A$ = "0" THEN RETURN
*** note 420 allows for the convention of '0' to exit
430 A = VAL(A$):IF A < 1 OR A > 9 THEN PRINT BEL$:GOTO 410
*** note if input is to be limited to a maximum of say 15
then add the following optional intercept line.
to acknowledge only the possibility of one 'ten' value.
435 IF A > 1 THEN RETURN
440 FOR I = 1 TO 600 : NEXT
450 GET A$:IF A$ = "" THEN RETURN
460 A = (A*10) + VAL(A$)
470 RETURN

****** Alpha STRING input *******
Note prior to accessing this subroutine, it is presumed that you have printed a
prompt at the bottom of the screen such as PRINT VT$(24);RK$;  - and further to
that, since this routine echoes your input by printing to the screen, you will
have returned the cursor to your current position on the screen by means of the
VT$( ) and TAB( ) functions.

500 A$=""
510 GET AA$:IF AA$ > "" THEN 510
520 GET AA$:IF AA$ = "" THEN 520
530 IF AA$ = CHR$(13) THEN PRINT VT$(24);BL$;:RETURN
540 IF AA$ = CHR$(20) AND A$ = "" THEN PRINT BEL$:GOTO 520
550 IF AA$ = CHR$(20) THEN PRINT "[cursor-left blank cursor-left]";:
A$ = LEFT$(A$ , LEN(A$)-1):GOTO 520
560 IF ASC(AA$) < 65 OR ASC(AA$) > 90 THEN PRINT BEL$:GOTO 520
570 PRINT AA$;
580 A$ = A$ + AA$:GOTO 520
```

Note prior to accessing this subroutine, it is presumed that you have printed a prompt at the bottom of the screen such as PRINT VT$(24);RK$; – and further to that, since this routine echoes your input by printing to the screen, you will have returned the cursor to your current position on the screen by means of the VT$( ) and TAB( ) functions.

A FEW THOUGHTS IN PASSING:

The previous small subroutines are by no means to be taken as the final word in programming techniques. On the contrary, as everyone has his own style, merely build on this and compile your own library of simple routines. Each of the above routines could borrow from others – for instance, you may or may not prefer to print your input to the screen as it comes in: you may or may not prefer to ring the bell on incorrect input (a flashing prompt may be more effective in your estimation, or *both*). Some common ideas, though, that you should be aware of include the 'keyboard buffer clear' as exemplified, for instance, in line 510. Such a procedure assures us that the input expected at least should somewhat resemble what we get – otherwise, there is no telling how many times the keyboard may have been inadvertently leaned upon during the previous course of your fantastic program.

Also, any prompts printing to the bottom line are best held there with a terminating semi-colon, else the screen may tend to scroll up one line – and remember, any subsequent returns to the screen must allow for cursor re-positioning.

There are two common ways to check for valid input – the VAL( ) or the ASC( ) functions. VAL(A$) on line 430 will return the numeric value *only if the input was numeric* – the VALue of other characters is 0. An optional numeric input check might read as follows : A = ASC(A$)-48 : IF A < 0 OR A > 9 THEN ..(error). Similar ASCII value 'filters' may be employed to check for any specific character, or range of characters. *TPUG*

To Be Continued. . .

TPUG COMAL COURSE - Part III

Borge Christensen
Tonder, Denmark

In Parts I and II of this series (TPUG Magazine, July and August September 1984) Mr. Christensen discussed COMAL procedures, variable-passing, recursion and conditional looping. In this article, he explores methods of branching.

5. **HOW TO FIGURE OUT A STRING.**

Clear the workspace and the screen and enter this function:

```
0010 FUNC VALUE(T$)
0020  NUM:=0
0030  FOR C:=1 TO 8 DO
0040    IF T$(C:C)="1" THEN
0050      BIT:=1
0060    ELSE
0070      BIT:=0
0080    ENDIF
0090    NUM:+NUM+BIT
0100  ENDFOR C
0110  RETURN NUM
0120 ENDFUNC VALUE
0130 //
```

Type RUN and enter this command:

```
PRINT VALUE("00001000")
```

Note the number returned by the function. Then try these commands one by one:

```
;VALUE("00001111")
;VALUE("10000000")
;VALUE("11111111")
```

The character ";" may replace the keyword PRINT. I think you have figured out what the function is doing. In any case this is how it operates:

PRINT VALUE("00001111")	The **function is called**; i.e. you ask for a number to be returned and printed out.
FUNC VALUE(T$)	The string "00001111" is **passed as a value** to the formal parameter T$.
NUM:=0	A value of 0 is assigned to NUM
FOR C:=1 TO 8 DO	The string held by T$ is scanned character by character, and

IF T$(C:C)="1" THEN
BIT:=1
ELSE
BIT:=0
ENDIF
	if the C'th character is a "1" BIT is assigned a value of 1, but if it is anything **else** BIT is assigned a value of 0.

NUM:+NUM+BIT ENDFOR C RETURN NUM	The present value of NUM is doubled and BIT is added to it, thus the binary equivalent of T$ is accumulated in NUM whose value is **returned** to be used in the calling statement in the place of function.
ENDFUNC VALUE	End of definition of VALUE.

continued overleaf

One of the statements in the FOR loop may need a little more attention:

IF T$(C:C)="1" THEN

The expression T$(C:C) returns a **one character substring** of T$, i.e. simply the C'th character of T$. If this character is equal to "1", the Boolean expression in the IF statement comes out TRUE; otherwise it comes out FALSE. We shall have more to say about substrings later in this article.

In COMAL the **truth value** TRUE is represented as a numerical 1, and the truth value FALSE is represented as a numerical 0. This fact can be utilized to simplify the function VALUE.

Remove the lines 40-80 by using the command:

DEL 40-80

and insert this one:

40 BIT:=(T$(C:C)="1")

Type the command RENUM (don't forget to press <RETURN>) and then LIST to see the following on the screen:

```
0010 FUNC VALUE(T$)
0020  NUM:=0
0030  FOR C:=1 TO 8 DO
0040   BIT:=(T$(C:C)="1")
0050   NUM:+NUM+BIT
0060  ENDFOR C
0070  RETURN NUM
0080 ENDFUNC VALUE
0090 //
```

Note how the lines have been **renumbered** to become orderly sequenced. Type RUN and then use some calling commands like the ones given above to test the function's working.

Since it is only tested whether an individual character is a "1" or not, other patterns than those applied before may be used. Try commands like this:

;VALUE("ABCD1111")

or simply

;VALUE("   1111") **There are four blanks in the string**

and note the results. Then change line 40 to become:

BIT:=(T$(C:C)="X")

Type RUN and try this command:

PRINT VALUE("   XXXX") **There are four leading blanks.**

Then try some of your own. What is the printout from this one:

PRINT VALUE("TXPXUXGX")

It is an important fact that patterns can be evaluated in different ways. We are going to used this extensively in the following.

Important note. In revision 0.14 if an error occurs during a direct mode call of a function or a procedure, the prepass is injured. Therefore in such case you have to restore the internal states by applying the RUN command before you can use the function or procedure again. In revision 2.00 the prepass state recovers automatically after a runtime error.

As you may know already, a number written in **eight bits** — binary digits — is called a **byte**. So VALUE simply converts a string that represents a byte into a decimal number.

Enter the command

AUTO 100

and add this procedure to the one already in workspace:

```
0100 PROC INTERPRET(LINE$)
0110  FOR I:=1 TO 3 DO
0120   BYTE:=VALUE(LINE$(I*8-7:I*8))
0130   PRINT BYTE;
0140  ENDFOR I
0150  PRINT
0160 ENDPROC INTERPRET
0170 //
```

Type RUN and enter these commands (don't forget <RETURN> after each:

```
INTERPRET("XXXX     XXXXXXXX     XXXX")          24 chars. in all
INTERPRET("  XXXX  XXXXXXXXXX  XXXX  ")
INTERPRET("  XXXXXXXXXXXXXXXXXXXX  ")
```

Each string may be looked upon as a representation of three bytes; each "X" being a "1" and each blank being a "0". And each call returns 3 decimal numbers, one for each byte.

Here is how INTERPRET works:

INTERPRET("XXXX XXXXXXXX XXXX")	The **string constant** is
PROC INTERPRET(LINE$)	passed as a value to the parameter LINE$, and the value of LINE$ is then
FOR I:=1 TO 3 DO	divided in three **substrings** each of which is passed on
BYTE:=VALUE(LINE$(I*8-7:I*8))	to VALUE. The numerical value returned is assigned to BYTE
PRINT BYTE;	and this BYTE is displayed.
ENDFOR I	
PRINT	A linefeed follows,
ENDPROC INTERPRET	and all is done.

Let us take a closer look at this line:

BYTE:=VALUE(LINE$(I*8-7:I*9)

If I is equal to 1, the **substring** LINE$(1:8) is pointed out, which means the first eight characters of the string that has been assigned to LINE$. This eight-character string is taken to VALUE for evaluation, and the result is assigned to BYTE. Next the substring LINE$(9:16) is picked up, and finally the substring LINE$(17:24) for similar processing.

What do you get out of this command:

INTERPRET("EXTRA FOR YOUR NEXT XMAS")

Try to make an answer without using the computer; then compare with an actual run.

Change line 130-150 of INTERPRET and add some lines to the program to get:

```
0010 FUNC VALUE(T$)
0020  NUM:=0
0030  FOR C:=1 TO 8 DO'
0040   BIT:=(T$(C:C)="X")
0050   NUM:+NUM+BIT
0060  ENDFOR C
0070  RETURN NUM
0080 ENDFUNC VALUE
0090 //
```

```
0100 PROC INTERPRET(LINE$)
0110  FOR I:=1 TO 3 DO
0120   BYTE:=VALUE(LINE$(I*8-7:I*8))
0130   PRINT USING "  ###": BYTE
0140   VALUE'OF'TPUG:+BYTE
0150  ENDFOR I
0160 ENDPROC INTERPRET
0170 //
```

continued overleaf

```
0180 DIM THISLINE$ OF 24
0190 //
0200 VALUE'OF'TPUG:=0
0210 REPEAT
0220  READ THISLINE$
0230  INTERPRET(THISLINE$)
0240 UNTIL EOD
0250 PRINT "------"
0260 PRINT USING "#####": VALUE'OF'TPUG
0270 //
0280 DATA "XXXXX XXXXX X   X  XXX  "
0290 DATA "  X   X   X X   X X   X "
0300 DATA "  X   X   X X   X X     "
0310 DATA "  X   XXXXX X   X X  XX "
0320 DATA "  X   X     X   X X   X "
0330 DATA "  X   X      XXX   XXX  "
```

It is up to you to find out what currency is used, but I can inform you that EOD means "End Of Data". It is a **Boolean function** that returns a value of TRUE when the last item in a data queue has been read. The statement

DIM THISLINE$ OF 24

declares THISLINE$ such that it can have a string assigned to it of 24 characters at most. Strings must always be declared in revision 0.14. In revision 2.00 simple strings need no declaration. *TPUG*

To Be Continued. . .

Integral Adventure Game

Brad Bjorndahl
Bramalea, ON

Editor's Note: This article describes a use for a computer as a tool for intellectual exploration. The "game" to which it refers is of a very academic nature.

The numerical adventure that I will describe here is one which has no ending, can take any direction which your imagination can find and has infinite and very real rewards for the resolute.

Each pathway is well-defined and can only be followed with an open mind and clear-headed reason. A familiarity with previous terrain speeds adventurers forward and will help them to recognize invalid routes and rewards which disappear on close examination. Often more than one path leads to the same reward and one path can diverge at the end to multiple rewards. This adventure becomes richer the farther it is travelled and so becomes correspondingly more difficult.

This article will provide a starting point, some references of those who have written of their own adventures and suggestions on how to use your computer for exploration and experimentation.

This adventure is integral in the sense that all versions of the game are parts of a whole and each version is simply a different starting point. This adventure is also integral in the sense that it uses integers. Professional integral adventure players call their game Number Theory but we amateurs intend to enjoy ourselves and will call it an Adventure, which is a far more accurate designation.

The starting point described here is relatively new, discovered sometime within the last half century. A fine introduction to this version of the Integral Adventure is found in reference 3. Briefly, the adventurer chooses any positive integer. If it is even, divide by 2; otherwise multiply by 3 and add 1. So simple! Now you have a new number, either greater or smaller than the first. If it happens to be equal to 1, then stop; otherwise repeat the even/odd test. Continue this way to get a sequence of numbers which may or may not end (at 1). Let's try 5, which is odd. The next number is 5*3+1 = 16. This is even so 16/2 = 8 is next. Again even so 8/2 = 4 is next. 4/2 = 2 and 2/2 = 1 to end the sequence. Try a few numbers for yourself but start small. If you experiment with 27 you will find out why reference 3 calls these hailstone numbers.

Before we start exploring we should, as good adventurers, do a little thinking. A simple observation shows that any power of 2 will be continually divided by 2 until 1 remains. Also, you may have noticed that all your experiments have ended on a 4,2,1 sequence. You may have also observed that the 4,2,1 sequence is a loop since the sequence formula applied to the number 1 yields 4. Which starting numbers end up in the loop? At least as many as you have tried and all powers of 2. Are there any more loops, that is, any numbers which lead to themselves? Why are some sequences long, such as the one that starts with 27 which has length 111. Where is the pattern in the sequences? Surely all integer sequences have patterns. There are many more questions to ask, but exploring a little now may shed some light. Try writing a program which will accept a starting value (e.g. 27) and apply the formula until the sequence hits 1 and will at the same time count the length of the sequence. This is not difficult, and you can check your results with reference 3. This is just an exercise, though, because all numbers up to 2 to the exponent 40 have been checked (in Japan) and they all end up 4,2,1. Does this mean all sequences end? Perhaps (2 to exponent 40)+1 ends in a different loop or does not end at all: i.e. continues climbing to infinity. A more difficult program to write will search for and record new loops. This program is useful for exploring sequences which are generated by adding a value other than 1 to odd numbers, that is, 3*N+C where C must be an odd number (why?). For C=3 you will immediately find the loop 12,6,3 and for C=5 two loops: 20,10,5 and 4,2,1 again.

There are more loops for C=5 but they are difficult to find without a computer. I have found 6 different loops after looking at starting values up to 500. Two of the loops are 44 numbers in length so your program which searches for loops must be generous in storing long numeric vectors. APL on the SuperPET allows arrays to be dynamically dimensioned to avoid this problem of overflow. APL is an ideal tool with which to play these numbers games but it is not fast. I expect that BASIC with a simple machine language routine to check for parity (i.e. the even/odd condition) and to do the arithmetic would be much faster but of course not as flexible as APL.

There are a few more small points before you continue on your way. First, use the number C to prove that there is always at least one loop. Second, be aware that this version of the Integral Adventure is relatively unexplored. Since there is little help outside your own skull it is both more exciting and more difficult. Lastly, for reasons described in reference 3, I call this version the Hailstone Adventure. For those interested in other versions, I can suggest the Prime Number Adventure and, my favorite, the Fibonacci Adventure. Enjoy the game. *TPUG*

References:

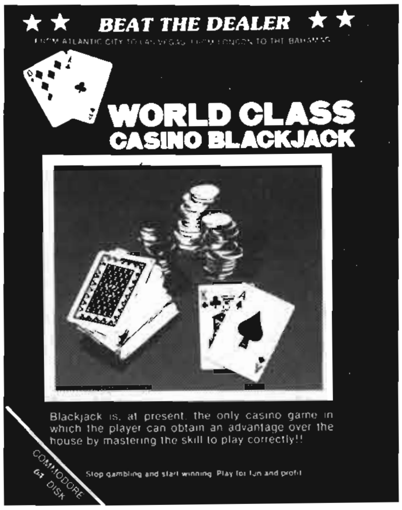
Beiler, Albert H. Recreations in the Theory of Numbers, 2nd ed. New York: Dover Publications, 1966.

Grosswald, Emil. Topics from the Theory of Numbers. New York: The Macmillan Company, 1966.

Hayes, Brian. "Computer Recreations", Scientific American, January 1984, pp. 10-16.

"... AT BLACKJACK THE PLAYER, IF HE PLAYS RIGHT, IS EXPECTED TO WIN IN THE LONG RUN!! ..."

NEW **FOR THE COMMODORE 64**



The Ultimate Blackjack game

COMPLETE WITH SUPER GRAPHICS AND SOUND EFFECTS

WORLD CLASS CASINO BLACKJACK

$49.95

Master the skill and self-control needed to play winning blackjack. Learn the most effective yet the simplest "card counting" system, which is used by the world's most successful international blackjack players. You can simulate the rules of the best casinos around the world where you ought to be playing and winning. Play one hand against the dealer and learn the basic strategy using the "card count" system, or play up to seven hands and see how well you can play under real casino conditions.

Playing one hand: the computer works the "card count", displays the basic strategy, all cards left in the shoe, each card's chance of being drawn, the percentage chance of going bust, and keeps track of all bets. By choosing different dealing speeds, you can progressively improve your "card count" until you can keep up with the fastest dealers. You can also get a summary of your game at any time during play.

BONUS — Included is a separate 37K M/L program that lets up to 7 different players play blackjack against the Casino at the same time making the game as close as possible to the real thing.

World Class Casino Blackjack was actually written by an international blackjack player who has won thousands of dollars in casinos all over the world. It is the most powerful tool to the player who wants to master the skill of playing winning blackjack.

World Class Casino Blackjack is the only computer game at which, if you are good enough to beat the dealer, you are good enough to win a million dollars.

AVAILABLE AT

WORLDS BIGGEST BOOKSTORE
20 Edward St.
Toronto, Ont.

COLES
the book people!
EATON CENTRE
Toronto, Ont.

COLES
the book people!
726 Yonge St.
Toronto, Ont.

COLES
the book people!
COMMERCE COURT
Toronto, Ont.

BONUS
Purchase World Class Casino Black at any of the above locations and receive absolutely FREE a copy of the book...
"BEAT THE DEALER"
By Edward O. Thorpe
"World Class Casino Blackjack" is published by NOVA COMPUTER SYSTEMS.

SuperPET 6809 Assembler - Part IV

Brad Bjorndahl
Bramalea, ON

This is the last of four articles on the assembler project PITS. A little more detail about the 6809 instruction set is necessary at this time. The 6809 processor is a very-much-enhanced version of the 6800, which in turn is similar to the 6502.

The enhancements include an extra stack pointer and index register, a direct page register and access to the 8 bit A and B accumulators as one 16 bit D accumulator. Also available are a set of extremely powerful addressing modes which can use any of the index registers and stack pointers. A programmer can apply a constant or variable offset to a 'base' register using the A, B or D accumulators for the variable offset. That is, the constant or accumulator is added to the register to obtain an address. Also the programmer may cause the register containing an address to automatically increment or decrement by 1 or 2. This facilitates table handling. On top of these address options, the programmer can also apply indirect addressing. This means that the contents of the address location is treated as an address which in turn points to the data of interest. This feature is useful for using tables which contain addresses for subroutines, for example. Lastly, the constant offset option can be used with the program counter as the base register. This allows position-independent code to be written.

Other enhancements in the 6809 are a number of 16 bit instructions for addition, comparisons, etc. There is an instruction which will multiply the 8 bit A and B accumulators and place the result into the 16 bit D accumulator. 8 and 16 bit signed addition may be performed in the 16 bit registers.

Among several hardware enhancements are the addition of three software interrupts (see 'swi' in the last article) and improvements in the bit representation of some instructions to increase efficiency.

The remainder of this article will describe selected parts of the program PITS which use some of the features described above.

Listing 1 is part of the routine which creates the island. The instructions at locations B and E load the index registers X and Y with addresses which correspond to the top and bottom left of the screen in preparation for displaying the upper and lower island boundaries. The # symbol signifies immediate addressing which means that the registers are loaded with the two bytes immediately following the load instructions. The instruction at 12 loads B with the number of asterisks to display and at 14 loads A with the contents at location ASTER which happens to be the ASCII representation of '*'. The loop is very simple; it decrements B and branches back until B is equal to 0. The interesting parts are the 'sta' instructions which store accumulator A (an '*'). The storage addresses are contained in index registers X and Y. The preceding comma means that there is no offset and the following plus sign causes the register to be incremented by 1. Two plus signs would mean an increment of 2.

Listing 2 is an example of a variable offset. First B is loaded with the contents of location Z PTR, which is a pointer to a zombie. Then X is loaded with the address of Z MOVE CTR, which is an array containing the number of moves left for each zombie. Then the contents of the array, offset by the zombie pointer, are decremented. This code is executed whenever a zombie has moved. Elsewhere, the same array position is tested for zero, using the same offset indexing method.

Listing 3 is a complete subroutine which is called during island building when a 'z' command is encountered. The subroutine checks if the zombie position array is full, and if not, then it stores the new zombie position and displays the zombie on the screen. First, the zombie counter (Z CTR) is incremented and compared to MAX Z. This is done with a 'cmp' instruction, which is basically a subtract that does not save the result but does set condition codes accordingly. The IF-ENDIF structure has not been previously described, but it is quite simple in this case. The 'if lt' only allows execution if either the negative or overflow condition codes are set, but not both. Thus 'lt' is translated as 'less than zero'. Within the IF, first the zombie counter is stored. Then Y is loaded with the contents of NEXT Z POS which was previously given the address of the array holding the zombie position. D is then loaded with the current row/column island position and stored in the position array. Register Y is automatically incremented by 2 in order to point to the next available place in the position array and is saved. When the current row/column position was determined, the corresponding memory address was calculated and saved in RCADDR. Therefore, in order to display a zombie, it is necessary to store a 'Z' at the memory address pointed to by RCADDR. That is exactly what 'stb (rcaddr)' does. The '( )' symbols mean indirect addressing as described above. This routine is more difficult to describe in English than it is to write. It is included to illustrate the power of some of the 6809 machine instructions.

Lastly is the routine in Listing 4 which converts a row/column position into a memory address. It was used to provide RCADDR in the last example and will illustrate some 16 bit operations. The routine assumes that row and column are in accumulators A and B respectively. Since people (that is, I) like to start counting at 1 but machines like to start at 0, it is necessary to decrement the row and column values. This is done with 'dec' and the column is stored in COL. I have used a trick here to be able to give one address two names that are meaningful, depending on whether 1 or 2 bytes are to be accessed. I have reserved 0 memory bytes for COLUMN, so that its address is the same as ZERO. I clear ZERO and end up with a 16 bit value for COLUMN. Next, B is set to $50 and is multiplied by A (the row) using the 'mul' instruction. The product is in D to which is added first the 16 bit column value and then an offset of $8000 for the screen address. 16 bit arithmetic makes address handling quite easy.

This completes my development of the PITS project. I have not touched some essential topics such as stack handling and methods of passing parameters to subroutines. These are probably best handled in a dedicated article. I must make two more comments before I close the subject. Those who examine the code in detail will find weak points in the usage of data arrays because they were omitted from the project design entirely. Let that be a lesson. Second, to prove to myself that the game structure is sound, I wrote the game in Waterloo microBasic with the identical structure as described. It ran correctly, but was a trifle (more than 20 times) slower. I hope that the last four SuperPET articles have provided more motivation for assembly language programming than a mere decrease in execution time. *TPUG*

continued overleaf

LISTING 1

```
000B  ; display top and bottom row boundaries
000B          ldx #$8000
000E          ldy #$86e0
0012          ldb #$50
0014          lda aster
0017          loop
0017            sta ,x+        ; store an asterisk from $8000 to $804f
0019            sta ,y+        ;  and from $86e0 to $871f
001B            decb
001C          until eq
```

LISTING 2

```
01FE          ldb z_ptr      ; decrement the number of
0201          ldx #z_move_ctr ; moves left to the zombie
0204          dec b,x
```

LISTING 3

```
0131  add_z equ *
0131          ldb z_ctr
0134          incb              ; increment the zombie counter
0135          cmpb max_z
0138          if lt             ; if zombie counter is less than max
013A            stb z_ctr       ; save the zombie counter
013D            ldy next_z_pos
0141            ldd rc_pos      ; and save the row/column position
0144            std ,y++
0146            sty next_z_pos
014A            ldb #'Z'        ; and display a zombie
014C            stb [rcaddr]
0150          endif
0150          rts
```

LISTING 4

```
0000  calc_rcaddr equ *       ; on entry, row is in A and column in B
0000          deca            ; this is to count rows and columns from
0001          decb            ; 0 instead of 1 so the calculation is correct
0002          stb col
0005          clr zero
0008          ldb #$50        ; the row number is still in reg A
000A          mul             ; multiply the row number by 80
000B          addd column     ; add on the column number
000E          addd #$8000     ; add the screen address offset
0011          rts
0012  column  rmb 0           ; 16 bit column value
0012  zero   rmb $01
0013  col    rmb $01          ; 8 bit column value
```

Using The User Port - Part 1

David Williams
Toronto, ON

(This article is based on a presentation which was given by the author at the 1984 TPUG Conference).

All Commodore computers are designed to interact with a wide variety of external equipment. For example, they can all be used with printers, tape-cassette decks, disk drives and modems. Many of them also come with sockets for plugging in other standard peripherals such as joysticks, external video monitors and sound amplifiers. In short, there are lots of off-the-shelf items which Commodore users can buy and connect to their computers.

The range of peripherals is not confined to those which can be bought from computer stores. Commodore has incorporated into all PET/CBMs, VIC 20's and C-64's a "user port". This is intended to allow users to make their own peripherals and connect them to their computers. The port allows the computer to receive information from the outside world, and also to output signals which can be used to control external equipment. These articles will describe the main features of the user port, and outline some of the ways in which it can be used.

Figure 1 is a diagram of the port, as seen looking into the computer from behind, showing its important connections. Physically, the port consists of a projecting part of the computer's main circuit board which is accessible through a large aperture in its case. On both the top and the bottom surface of the circuit board are twelve metal contacts, so there are twenty-four contacts in all. These contacts are connected within the computer to various points in its circuitry, so that connections to these points can be easily made by the user. Some of the connections are different on the various kinds of Commodore computer. However, the important contacts which are labelled along the bottom side of the circuit board in **Figure 1** are the same on all PET/CBMs, VIC 20's and C-64's. These are the contacts with which this article will be mainly concerned.

Port Connection Diagram

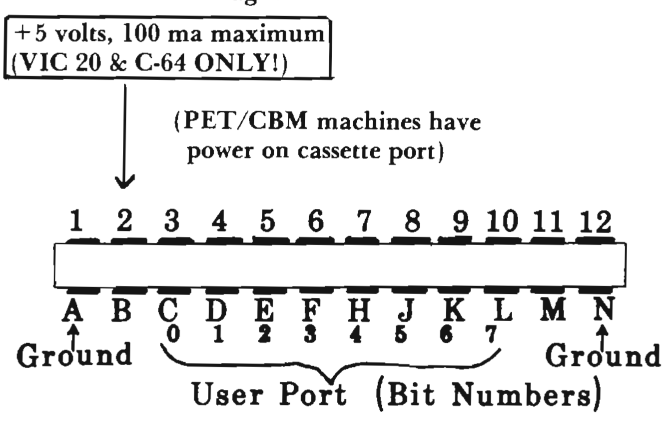


fig. 1

Getting Connected

It is important to realize from the outset that the contacts along the bottom of the port are quite separate from those along the top. Many a computer has spent a while in a repair shop because its user unthinkingly used an alligator clip, or something of the sort, to make a connection to the user port, thereby connecting a bottom contact to the one above it on the top of the board. I would strongly recommend anyone who plans to use the port to buy a "proper" connector for it. This connector consists of a strip of plastic containing a slot which fits snugly over the projecting part of the circuit board in the port. Twenty-four metal contacts within the slot make connection with the contacts on the circuit board, and are accessible through twenty-four separate solder-lugs on the outside of the connector. The spacing of the contacts is 3.96 millimetres (0.156 inches). The total length of the slot is 52 millimetres (2.04 inches). A suitable, commercially-made connector is the Edac type 308-024 400-202, a photograph of which is shown in **Figure 2**. I buy mine, in the Toronto area, at Electro-Sonic, on Gordon Baker Road, near Victoria Park and Steeles.

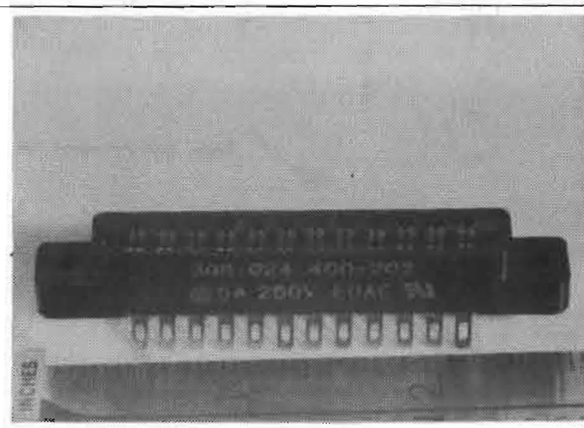


Photo by Roberto Portolese

fig. 2

Unfortunately, you may find these connectors hard to obtain. (In fact, in order to get a hundred of them for sale at the 1984 Conference, TPUG had to have them especially made! They were all sold long before the Conference was over). The twenty-four contact connectors are less common than other sizes. This has led many users to try desperate measures such as buying unsuitable connectors and cutting them to fit. If you try this kind of thing, I suggest that you take a ruler to the store so that you can check that the contact spacing is correct. If you buy a connector which is too long, then cut it to fit, make sure that you close the open end of the slot with some material such as epoxy glue so that the connector cannot slip sideways when installed. Slipped connectors can cause disasters by bridging together adjacent contacts on the circuit board. Another desperate measure is to cover the top contacts on the port with adhesive plastic tape (preferably a transparent kind, so that the contacts can still be seen), then to use alligator clips to make connections to the bottom contacts. This works as an emergency procedure, but I would not recommend it for routine use. The clips are liable to abrade through the tape before long!

The Eight-Lane Highway

As is shown in **Figure 1**, the twelve contacts along the bottom of the port are normally described by labelling them with letters, A through N. (G and I are omitted to avoid confusion with other letters and digits.) The outermost pins, A and N, are both connected to "system ground". This is the reference point, relative to which everything else is measured. (Incidentally, the outermost pins on the top of the port are also grounded). Pins B and M are used for purposes which are beyond the scope of this article. The remaining eight pins, C to L, can all be used to send information into or out from the computer. This is done by switching the

continued overleaf

voltages on the pins between two levels, which can represent the digits zero and one. In a sense, the eight pins can be considered as being like a highway with eight separate lanes, each carrying its own independent flow of traffic. However, unlike a highway, the eight pins on the user port are not permanently divided into "inbound" and "outbound" channels. The user can choose whether any given pin should carry information into the computer or out from it. Any combination is possible. All the pins can be used as inputs, or all as outputs, or some of each.

This choice of directions is made by POKEing an address called the "direction register". Like every other address in the computer, the direction register consists of eight bits, each capable of holding a binary digit of information – a zero or a one. The contents of each of the eight bits in the direction register determines whether one of the user port pins functions as an input or as an output. The least significant bit (Bit 0) determines the function of Pin C, and the sequence follows in simple numerical order, so that Bit 7 determines whether Pin L is an input or an output pin. If a bit contains a zero, the corresponding pin is an input. A bit containing a one corresponds to an output pin.

Renaming the Pins

Here, I intend to make a change in nomenclature which should simplify the rest of this article. Instead of referring to the eight pins by their letters, C to L, I will instead use their bit-numbers, 0 to 7. Thus, Pin C will be called "Pin 0", and so on. This should make things easier to undertand. However it is important to realize that these numbers have nothing to do with the numbers which are used to describe the pins on the top of the port. Don't get confused between them!

The number which must be POKEd into the direction register must therefore have a bit-pattern which corresponds to the desired pattern of input and output pins. For example, if you want pins 2, 4 and 5 to be outputs, and the rest to be inputs, you must POKE the direction register with the number in which bits 2, 4 and 5 are ones, and the rest zeroes. To do this, first look in Table 1 to find the address of the direction register in your computer (I'll call this address "DR"), then, either in direct mode or as a line of a program, execute the command:

```
POKE DR,(2↑2) OR (2↑4) OR (2↑5)
```

(Oops! Maybe some of you are not familiar with the "Boolean operators" such as "OR". If you know all about OR, AND and NOT, just skip to the end of this paragraph. Otherwise read on... The OR operator performs an operation on two numbers to generate a third. Here is an example:

```
10010111
01000110
--------
11010111
```

The numbers are expressed in binary notation, and the resulting number has a one in every bit-position where one OR the other OR both of the starting numbers has a one. Similarly, ORing the numbers 2↑2, 2↑4 and 2↑5 (as in the previous paragraph) has the following effect:

```
00000100
00010000
--------
00010100
00100000
--------
00110100
```

The final number has the desired bit-pattern. The AND operator is similar, except that both of the starting numbers – one of them AND the other – must have a one in any position in order for the resulting number to have a one there:

```
10010110
01011011
--------
00010010
```

The final Boolean operator is NOT, which has only one starting number and simply reverses all its bits:

```
01100011
--------
10011100
```

There! Now you know all about Boolean operators in BASIC! I suggest that you experiment with them in order to become totally familiar with them).

The first programming operation which must be done whenever the user port is to be used must be to POKE the direction register. A few people sometimes omit this POKE if they want to use the port only to input information into the computer. They rely on the fact that, when the computer is first switched on, the direction register usually contains a zero, making all the port pins inputs. However, my own experience is that this is unreliable. If the direction register has previously held a non-zero number, it may not be restored to zero if the computer is briefly switched off. I suggest, to be safe, that you should POKE the direction register even if you want all the pins to be inputs.

Reading Input Pins

Let's try a little experiment. To do this, you will need a user port connector and a piece of wire, preferably with an alligator clip on each end. (Wires with alligator clips on both ends are called "clip leads", and are sold in stores such as Radio Shack. I recommend any hardware experimenter to have lots of them). Connect one end of the wire to one of the grounded pins on the port and leave the other end unconnected for the time being. Switch the computer on and POKE the direction register to zero, so all the pins are inputs.

continued on next page

TABLE I

Addresses	PET/CBM (2.0 AND 4.0)	VIC 20	C-64
Direction Register (DR)	59459 ($E843)	37138 ($9112)	56579 ($DD03)
Port (P)	59471 ($E84F)	37136 ($9110)	56577 ($DD01)

Now PEEK the actual port address, which I call "P", and which is shown for all Commodore machines in Table 1. You should find that it contains the number 255 (decimal), which, in binary notation, is 11111111 — an eight-bit number in which every bit is a one. Now connect the free end of your grounded wire to one of the eight port pins C to L, which are now renamed 0 to 7. Let's suppose that you ground Pin 3. PEEK the port address again. You should find that you now get 247 (decimal), which has the bit-pattern 11110111. Bit 3, corresponding to Pin 3, has become a zero. (Remember that the rightmost bit in a binary number is called Bit 0, since it represents 2↑0. The leftmost bit, Bit 7, represents 2↑7.) Do some more experiments, connecting various of the pins 0 to 7 to ground and PEEKing the port address. More than one pin can be grounded at one time, if you wish. You should find that the port address always contains a number in which the bits corresponding to the grounded pins are zeroes and the rest are ones.

If you have a voltmeter available (all serious hardware hackers own voltmeters), try measuring the voltages on the ungrounded pins. You should get approximately +5 volts, relative to ground. The grounded pins, of course, are at zero volts. These two voltages, +5 and 0, are the standard TTL (that's short for "Transistor-Transistor Logic") voltage levels to represent the digits one and zero, respectively. If any input pin on the port is at roughly +5 volts (say between +4 and +5), the corresponding bit in the number found by PEEKing the port address is a one. If the pin is close to zero volts, the bit is a zero. Thus information can be fed into the computer from the outside world by connecting something to the port which drives a pin to either +5 or 0 volts, and running a program which PEEKs the port address and looks at the appropriate bit.

The Boolean operator "AND" can be used to look at just one bit of a number. For example, if you want to look at bit 5, simply AND the number with 2↑5. Thus if you want to write a program which will do something if Pin 5 of the user port is at zero volts simply use the condition:

IF (PEEK(P) AND (2↑5))=0 THEN . . .

This condition will be satisfied whenever bit 5 in PEEK(P) is a zero, irrespective of all the other bits. Incidentally, the outermost brackets are very important. Without them, the condition will never be satisfied since 2↑5 never equals zero! The "equals" operator is performed before "AND", unless brackets are used to force the opposite order.

I hope you will now have no problem in seeing that the following condition is the one to use if you want the computer to do something if Pin 2 is at +5 volts:

IF (PEEK(P) AND (2↑2))>0 THEN . . .

More Observations

The results of our little experiment of looking at voltages on input pins suggest some other important facts.

First: The voltage on any pin of the user port should NEVER be driven outside the TTL range of zero to +5 volts, relative to ground. There are some people who cheerfully put voltages well outside this range onto the port and get away with it. Similar people plug air-conditioners into lighting circuits in the expectation that the circuit designers will have put large enough safety margins into their calculations to make such overloads possible. Fire marshals know that people who do this kind of thing sometimes get away with it for a while, but rarely for ever.

Second: A voltage of +5 volts represents a logical one, and zero volts represents a logical zero. If the voltage is wrong by half a volt or so, the logical digit will not be changed. However, there is a "hazy" region around +2 or +3 volts where the system will not reliably determine which digit is indicated. Putting an intermediate voltage such as these onto a user port pin will not cause any physical damage, but it will tend to make your logical system work unreliably.

Third: An input pin which is not connected to anything outside the computer is held at +5 volts, and a logical one, by the internal circuitry of the computer. To make it read a logical zero, the pin must be grounded. In practical circuits, this grounding is often done through a resistor, so that the pin can be put back to +5 volts by driving a current through the resistor. I have found that, in practice, the resistor must have a value of no more than about 1K (1000 ohms) in order for the pin reliably to appear at a logical zero. Usually, I use 470 ohms or thereabouts.

Programming Output Pins

Let's make the user port pins into outputs, and try another experiment. To do this, you'll need a voltmeter. If you don't have one, I guess you'll just have to read along and take my word for it.

Make all the pins into outputs by POKEing the direction register with 255 — the number in which all eight bits are ones. POKE the port address with zero, and measure the voltages of all the eight pins, relative to a ground pin. You should find that they are all close to zero volts. Now POKE the port address with 255 and measure the voltages again. All the pins should be close to +5 volts.

POKE the port address with the number 245, which has the bit-pattern 11110101. Measure the voltages. Pins 1 and 3 should be at zero volts, the rest at +5 volts. This corresponds to the fact that bits 1 and 3 in the binary equivalent of the number 245 are zeroes and the rest ones. Try POKEing the port address with other numbers and measuring the voltages on the pins. You should always find that the pattern of +5 and 0 voltages on the pins matches the bit-pattern of the number you have POKEd into the port. Each bit in the port address controls one output pin.

Try POKEing the direction register with a number between zero and 255, then repeat the experiment of POKEing various numbers into the port and measuring voltages. You should find that your POKEs to the port address can influence only the pins which you have programmed to be outputs with the direction register. For example, if bit 5 in the number you have put into the direction register is a one, pin 5 can be switched between zero and +5 volts by POKEing the port address. However, if bit 5 in the direction register contains a zero, this pin is an input and is unaffected by POKEs to the port. Like all unconnected input pins, it remains at +5 volts.

continued overleaf

Why a Voltmeter?

Those of you who don't have voltmeters may be tempted to try the above experiments with some other means of detecting voltages, such as flashlight bulbs. Surely, if a bulb is connected between ground and a user port pin which is at +5 volts, it should light. If the pin is at zero volts, the bulb will not light. Try this if you like, but you're inviting disappointment. The bulb will never light, no matter what numbers you POKE into the direction register and the port address.

The problem is that the output pins on the user port are each capable of supplying a current of only about one milliamp (one thousandth of an ampere) at +5 volts. A voltmeter takes far less current than this, so it responds properly in our experiments. However, a flashlight bulb requires much more current to light — typically around a hundred milliamps. If an output pin on the user port is at +5 volts and a flashlight bulb is then connected between it and ground, it cannot supply enough current to light the bulb. Instead, the voltage on the pin is dragged down close to zero volts, almost as if the pin were grounded with a thick piece of wire.

Incidentally, the internal circuitry of the computer is set up so that grounding output pins is not supposed to cause any damage. However, to be on the safe side, I would not recommend doing it for long periods.

POKEing One Bit at a Time:

When a number is POKEd into the port address, its bit-pattern determines the voltages on all the output pins. In practical situations, if there are several devices connected to different output pins, a programmer is likely to be interested in writing code which will control them one at a time, not all at once. What is needed is a technique, in BASIC, of changing just one bit in a number, leaving the others unaltered.

Several operations are needed. First, the port address is PEEKed to find what number is already in it. Boolean operations are then done to change just one bit in this number, and the resulting number is POKEd back into the port address. In practice, these operations can usually be written into one line of code. For example, if you want to POKE the port address so that it will contain a one in bit 4, simply execute the instruction:

POKE P,PEEK(P) OR (2↑4)

Whatever number was originally in the port address is ORed with 2↑4. If you recall what the OR operator does, you should recognize that this forces a one into bit 4 and leaves all the other bits unchanged. The resulting number is then POKEd into the port.

To force a zero into a bit-position in the port address is only slightly more complicated. Here is an instruction which will make bit 6 contain a zero, and leave all the other bits unchanged:

POKE P,PEEK(P) AND NOT (2↑6)

NOT (2↑6) is a number which has a zero in bit 6 and a one in every other position. ANDing this number with PEEK(P) produces a new number which has a one in every position where PEEK(P) has a one, EXCEPT bit 6, which is bound to contain a zero.

A Slight Snag

The logic of the last few paragraphs depends on an assumption which seems so obvious that it is virtually never questioned. It assumes that PEEKing the port address will produce a number which represents the "true state" of the port, so that every output pin which is supposed to be at +5 volts will be represented by a one in the corresponding bit-position.

This assumption can occasionally (fortunately avoidably) be wrong. PEEKing the port effectively measures the actual voltages on the pins. If an output pin is connected to an external device which draws more than about one milliamp from it, its actual voltage may be well below +5 volts even though it is programmed to be at +5 volts. PEEKing the port address may then produce a number which has a zero, instead of a one, in the corresponding bit-position. (You can easily confirm this experimentally, if you want, by grounding an output pin and PEEKing the port.) When this number, modified by Boolean operations on another bit, is POKEd back into the port, the result is to reprogram the low-voltage output pin to be at zero volts, so it ceases to supply any current to the external device to which it is connected. This can mystifyingly cause external devices to be affected by instructions which are intended to affect only other pins.

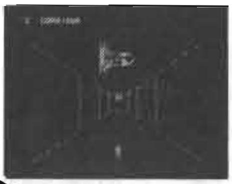
This problem never occurs unless an output pin is held at a low voltage by an external connection which draws too much current. For this reason, I make a practice of making all connections to output pins through resistors of at least 4.7 kilohms. These resistors limit the currents taken from the pins to acceptable levels. All I then have to do is to design my external circuitry in such a way that these small currents are enough to perform the functions I want — which is an entirely different problem! *TPUG*

To Be Continued. . .

Super Action Software!

PROGRESSIVE PERIPHERALS & SOFTWARE
DENVER, COLORADO

CYBERWORLD $39.95



This five-screen arcade adventure packs the computer with intense graphics and sound! You are a special Cyberleague agent in a universe full of hostile aliens and vicious robots. Joystick and keyboard transport you through 3-D rooms, space barriers, fleets of invaders, and warship-ridden quadrants of space. Over 100 sprites, 8 new character sets, and dozens of mind-boggling sound effects make up this multi-layered adventure. Animation, action, and strategy all combined into a game so extensive that two disk sides are jammed with game programs and data! Reach the ultimate rank of admiral and you may carve a niche in the permanent high-score list. A full-size book quality manual with full-color covers is included to guide you through your most exciting game experience.

NEW

NEW

Perplexian Challenger $29.95

The incredibly responsive three-axis joystick control of a space fighter is in your hands. Split screen graphics provide a continuous display of your ship's instrumentation, as well as a three-dimensional, animated view of space.

You, as a pilot, must utilize lightning fast reflexes to destroy invading ships, and avoid their return fire. Simultaneously, you must maneuver your ship to capture space debris that remains from the explosions.

Outstanding graphics features include smooth 3-D rotations, split screens, and the most incredible high-resolution hyperspace sequence ever produced.

Programmed entirely in machine language, this action-strategy game is guaranteed to blow you away.

All the professional features you expect are included: automatic self-demo, high score retention, pause, and provisions for 1 to 4 players. Add to this, features you don't expect like easy-loading, and music during the load. Perplexian Challenger is a game that brings the arcade experience to your home.

wizard $39.95

Jump from ropes to ladders, dodge plummeting boulders and duck under deadly arrows in your quest for sparkling diamonds, gleaming bars of gold, and glistening pearls. With joystick in hand you must explore forty dazzling screens, each a new and exciting adventure. Take the key to unlock the doorway to your next spine-tingling level. Each key restores your magical powers, allowing you to cast over ten different spells. With these magic spells you have the power to overcome vicious creatures, terrifying traps, and perilous plunges.

Your Wizard is realistically animated in every possible direction. Dozens of movements are possible — jump over burning fires, shimmy up or down ropes and ladders, even slip down treacherous sliding staircases! Magic portals move your Wizard through midair and protect you from a myriad of fully-animated fiendish monsters. Catch an elevator to the top of the screen and dart through sliding gates in your quest for magic and treasure.

Wizard's fascinating variety of screens are sure to please and entertain, and of course you can build an unlimited number of your own levels using the construction set provided with your game.

Gothmog's Lair $39.95

Real-time adventure excitement at its best. Solve countless puzzles and slay over a dozen monsters by using the huge vocabulary of over 200 words. Two challenging difficulty levels await you with over 80 areas, each fully described in Old English script.

Menacing monsters, knaving villains, tattered code books and treacherous terrain are just a few of the situations you must overcome in your quest for the thirteen priceless treasures. More than seventy objects are invaluable to you in your search for glory and wealth!

A full-size, thoroughly illustrated manual is included. Featuring color front and back, book quality, and a fold-out map, this "extra" further extends the professionalism of this game. The following are quotes from unsolicited testimonials sent to us by adventurers in Gothmog's Lair.

"I have extremely enjoyed Gothmog's Lair, and plan to buy more adventure games."

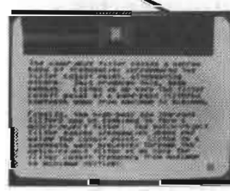
Scott Tulman
Memphis, TN

"Gothmog's Lair is the best adventure I've ever played."

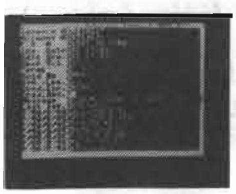
Dennis Manochio, Jr.
Saratoga, CA

THE PROFESSOR $34.95

An in-depth self-tutorial for the Commodore 64 on a two-sided disk. This menu-driven tutorial covers every aspect of your C-64—BASIC, keyboard, sound, music, simple and advanced graphics. Quizzes test your comprehension at the end of many lessons! On-screen illustrations, sound effects and full-color interactive graphics make learning easy and interesting. The PROFESSOR is your ONLY choice for an all-in-one, thorough tutorial about the Commodore 64!



Supershipper 64



A complete multi-printer shipping system which operates on the inexpensive Commodore 64. Offers all the features of more expensive, more cumbersome business software at a low, low price. Full-screen, full-cursor editing on all data entry. Prints invoices, C.O.D. tags, mailing and shipping labels. Sorts your customer list alphabetically, by city/state or by salesperson. Keeps complete records of all invoices and accounts on disk — up to 800 accounts per disk! Automatic backups, product charts and many other features are also included.

Supershipper Accounting

The accounting supplement to the Supershipper 64. Prints statements, bank deposits, past due accounts and daily or monthly sales reports. Breaks down sales commissions and prints all customer's past transactions. Also provides inventory control for up to 200 different products. The only way to fully computerize your business!

Supershipper 64	$99.95
Supershipper Accounting	$79.95

Progressive Peripherals & Software
2186 South Holly, Suite #2, Denver, Colorado 80222

VISA **Call for more information or a dealer near you.** MasterCard

TELEX: 888837 (303) 759-5713 (303) 757-0830 TWX: 9109971314

Forecasting With The 8032 -- Part IV

John Shepherd
Islington, ON

The TPUG February PET disk, (P)T6, contains four programs which can be used to project historical data into the future, through a process called time series analysis. The first three parts of this series of articles (*TPUG Magazine*, May, June and July 1984) defined some of the forecasting terminology and concepts, and showed how two of these programs are used.

Moving Averages

The third program on the disk, "forcst ma.8", uses moving averages of historical data to generate forecasts. An "N-period simple-moving-average" is merely an arithmetic average of the most recent N observations of historical data. The forecast is equal to the calculated value of this moving average.

The apparant simplicity of this forecasting procedure is deceptive. There are many applications in business where there is so much variability in the historical data that the simple-moving-average forecasting model performs just as well as much more sophisticated techniques.

Recent developments in the theory of time series analysis were consolidated by G.E.P. Box and G.M. Jenkins in 1970, in their classic text *Time Series Analysis, Forecasting, and Control.* These forecasting techniques, called Box-Jenkins models, are complex but generally produce superior results. However, the procedures are time-consuming to implement and preferably require about 100 observations of historical data. I have had experience with monthly business sales forecasts, where a six-month simple moving average produced just as accurate forecasts as the optimum Box-Jenkins model. Box-Jenkins models are beyond the scope of this series of articles and are not included among the programs on the TPUG disk. I suspect, however, that for many applications these TPUG programs can provide close-to-optimum forecasts.

The properties of the simple-moving-average method depend upon N, the number of historical observations to be averaged. If N is large, the moving average will react slowly to changes or trends in the data. If N is small, the moving average will react more quickly and will be more influenced by random fluctuations in the data. For most data there is often an optimum N which produces a minimum forecast error.

The Naive Model

If N=1, then the forecast is equal to the most recent recorded data. This is called the "naive" model, and is the forecasting process used intuitively by most of us in our daily lives, in situations where there is no obvious trend or seasonality and no obvious degree of randomness in the data.

In business the naive model is useful in perhaps 30% of all short-term forecasting situations. The program "forcst ma.8" uses the naive model as an option, so that its results can be compared against other moving average models.

Moving Averages for a Trend Process

A simple-moving-average will always lag a trend in the data. The larger the value for N, the larger will be the lag. For this reason, simple-moving-averages are not used for forecasting where the data shows a pronounced trend. A double-moving-average method is often used in such situations.

A "double-moving-average" is merely a moving average of the simple-moving-averages. The following formula is used to calculate the forecast, which is corrected for a linear trend:

F = (2*M1 – M2) + 2*LT*(M1-M2)/(N-1)
where:
F is the forecast for LT periods ahead
LT is the forecast lead time
M1 is an N-period simple-moving-average
M2 is an N-period double-moving-average
N is the period for both moving averages

Program "forcst ma.8"

Load the program and run it. The program first asks for the name of a data file and calls it into memory. Load the sample data file "forcst data.8". The data can be displayed graphically, if desired, to see whether it appears to have any seasonality or trend. This data seems to have both.

Next, choose one of three forecasting models; the naive, the simple-moving-average, or the double-moving-average. As a general principle, always choose the simplest forecasting model that might reasonably work. It's often informative to compare the performance of the naive model with one of the other two. For our data, let's try the double-moving-average model. Enter next the desired forecast lead time, let's say 2 periods or 2 months.

If one of the moving average models has been chosen, the program asks whether you wish to input the moving average period, N, or whether you wish to have the computer calculate the forecast error for a range of periods and determine the optimum period. For our data, let's find the optimum, for a range of 2 to 12 months.

For any of the three models chosen, the program next asks whether we wish to make any multiplicative corrections for seasonality (see **part II**, *TPUG Magazine*, June issue) and if so, asks for the period of seasonality. If there is any doubt whether seasonality is significant, it is wise to try this option. Even the naive model becomes extremely useful when corrected in this manner. For our data (monthly beer sales), we obviously wish to correct for seasonality.

The program then goes to work and calculates forecast error over the entire time series (for our data, over the prior 60 entries). It displays the mean square error (M.S.E) and the mean absolute deviation (M.A.D.) of these forecast errors for each moving average in the range chosen.

For our data file the optimum moving average period is 9 months. The M.S.E. is 1.7 units and the forecast for period 62 is 21.2 units. (A unit is one-thousand cases of beer). This means that we can expect our forecast to be accurate to within plus or minus 1.7 units two-thirds of the time and to within plus or minus 3.4 units 95% of the time. If we display the forecast error versus time we can see that the error looks random and that the forecast can probably not be improved very much by other models.

In the next article we'll look at the program "forcst exp.8". This program also produces short-term forecasts but uses a special kind of moving average, using a technique called "exponential smoothing". *TPUG*

To Be Continued. . .

Accurate C-64 Timekeeping

Glen C. Bodie
Toronto, ON

Commodore computers use a version of BASIC which has two built-in system variables for use in keeping track of the time: TI and TI$. Information on these can be found in *The Commodore 64 User's Guide* on page 113 or in *The Commodore 64 Programmer's Reference Guide*, page 89.

These variables are both based on the system "jiffy" clock. This is an interval timer which is updated every 1/60th of a second by the system. The TI variable returns the interval timer value numerically. The TI$ variable is updated by the interval timer, but is read and written as a character string which contains "time" in a 24-hour HHMMSS format. Either of these (especially TI$) can be very useful when you are writing a program which wants to keep track of the time. Once you set a value in TI$, the system will keep the time updated for you.

A simple program which allows you to enter a time, and then displays the time back to you like a clock is shown in Program 'A'. Note that in order to prevent getting a BASIC error message, there should really be some checking on the values entered for the time.

The Commodore 64 User's Guide does not mention it (*The Commodore 64 Programmer's Reference Guide* does), but the values of these system variables are not accurate following tape I/O. This is because the 1/60th second system interrupt is disabled during part of the I/O process. What both Guides fail to mention is that these system variables are also not accurate following disk I/O.

If you are trying to write a program which keeps track of the time and also does some I/O (disk or tape), this can be a real headache. There seem to be only a few alternatives:

1) Use machine language to modify the interrupt code so that the 1/60th second interrupt is not disabled during I/O!

2) Derive some sort of algorithm which tries to "account" for the time which is lost: ie. add 7.33 seconds to the time, because the disk write was for 23 blocks and had to do 2 track seeks!

3) Forget about trying to keep track of the time or forget about doing any I/O!

4) Find another way to keep accurate time even when doing I/O.

It is this fourth alternative that turns out to be the best. Off-hand, that sounds like a formidable task, but the Commodore 64 has an easy way to do this. I am referring to one of the many wonderful abilities of the CIA chip (Complex Interface Adapter). The C-64 has two of these which are described in great detail in *The Commodore 64 Programmer's Reference Guide* in Appendix M starting on page 419.

The way the C-64 is put together, CIA #1 is used primarily to interface to the keyboard and joysticks, and CIA #2 is used primarily to interface to the serial bus and the user port. None of these functions interferes with one of the other functions of the CIA chip—maintaining a TOD (Time-of-Day) clock. It is this TOD clock which we will use to keep the time.

Using this TOD clock is certainly more complex than using the TI$ variable, but at least it doesn't lose any time! It is more complex because the TOD registers are designed for use in driving a digital display and thus present their data in BCD (Binary Coded Decimal). BCD is an encoding technique which keeps decimal numbers in a format which is easy to understand, but difficult to calculate with. The examples below show some numbers stored in BCD format and in ordinary binary (internal computer format).

NUMBER	in BCD	in Binary
01	0000 0001	0000 0001
09	0000 1001	0000 1001
10	0001 0000	0000 1010
25	0010 0101	0001 1001
99	1001 1001	0110 0011

In the BCD format, the largest number which can be held in one byte (8 bits) is 99, since each 4 bit "nybble" is treated separately, and can only hold the decimal digits between 0 and 9 inclusive. In binary, the maximum is 255 since the two nybbles are joined together and each can hold a number up to 15. That is enough for BCD, now let's look at the registers in the CIA chip which are used when dealing with the TOD clock.

In total, there are 16 registers presented externally which are part of each CIA chip. The only ones which we need to concern ourselves with when using the TOD clock are registers 8, 9, 10, 11, 14 and 15. These are mapped into the C-64 memory at locations in the range 56320-56335 (DC00-DC0F) for CIA #1 and 56576-56591 (DD00-DD0F) for CIA #2. The address for a particular register is obtained by adding the register number to the starting address for the desired CIA chip. The functions of the various registers are explained below.

8) TOD 10th of seconds:
This register contains the tenths of seconds data for the TOD clock. It is encoded in BCD, but since the maximum value is 9, this is not of any concern.

9) TOD seconds:
This register contains the seconds data for the TOD clock. It is encoded in BCD as well.

10) TOD minutes:
This register contains the minutes data for the TOD clock. It is encoded in BCD as well.

11) TOD hours:
This register contains the hours data for the TOD clock in a 12-hour format. The high order bit (bit 7) indicates whether the time is AM or PM. If the high order bit is masked out, the rest of the register data is just hours encoded in BCD.

14) Control Register A:
The high order bit (bit 7) of this register is used to determine what frequency base is to be used for timing. We do not really need to do anything to this bit, since it is set to zero when the system is powered-up and this means 60 Hz. The construction of a C-64 (in North America, at least) has a 60 Hz signal applied to the frequency input pin on the CIA chip.

continued overleaf

15) Control Register B:
The high order bit (bit 7) of this register is used to interpret any data written into the TOD registers (8-11). If this bit is zero (as it is when the system is powered up), it means write to the current time. If this bit is one, it means write to the alarm time.

You may have noticed that bit 7 of register 15 allows you to write to the alarm time. The CIA chip will keep an alarm time and generate an interrupt when that time is reached. Handling this interrupt is beyond the scope of this article and requires the use of register 13, the Interrupt Control Register in order to allow the interrupt to occur and to handle it when it does occur.

In order to use this TOD clock in your programs, you need to be able to take a time value entered via the keyboard, put that value into the TOD clock, and then read and display that TOD clock value whenever you need it. Program 'B' is a subroutine which can be called to get the time data from the keyboard and put it into the TOD clock. Program 'C' is a subroutine which can be called to read the time data from the TOD clock and write it to the screen. Both of these can be incorporated into your programs without too much modification.

There are a few tricks to using the TOD clock. In order to set the time, the following sequence of operations must be performed in exactly this order:

1) Write to the hours register (11)
2) Write to the minutes register(10)
3) Write to the seconds register (9)
4) Write to the 10ths of seconds register (8)

The write to the hours register stops the CIA chip from updating the TOD clock, and the write to the 10ths of seconds register starts the CIA chip updating the TOD clock again. This guarantees that the seconds or minutes don't roll over and change your hour data after you entered it and thus the time starts exactly where you want it to.

In order to read the time, the following sequence of operations must be performed in exactly this order:

1) Read the hours register (11)
2) Read the minutes register (10)
3) Read the seconds register (9)
4) Read the 10ths of seconds register (8)

The read of the hours register freezes the TOD clock, and the read of the 10ths of seconds register un-freezes it. During the freeze, the CIA chip is still updating the values, but the registers presented to the outside world (that's us) are not allowed to change. This guarantees that you get a consistent set of time data, not some data before the hour rolled over and some afterwards. It is possible to read any register except the hours by itself, but if you read the hours, you must then read the 10ths of seconds in order to un-freeze the registers.

Hopefully, this description and the accompanying programs will prove useful to you when trying to keep accurate time with your C-64 while doing I/O. If nothing else, this should help make you aware of the many features hidden away inside your computer just waiting to be discovered!

continued on next page

```
100 REM-------- PROGRAM 'A' -----------
110 REM TITLE AND INPUT INSTRUCTIONS
120 PRINT "⊐"TAB(13)"KEEPING TIME": PRINT
130 PRINT TAB(13)"PROGRAM 'A'": PRINT: PRINT
140 PRINT "ENTER THE CURRENT TIME IN 'HHMMSS'"
150 PRINT "FORMAT WHERE 'HH' IS HOURS   FROM 00-23"
160 PRINT TAB(13)"'MM' IS MINUTES FROM 00-59"
170 PRINT TAB(9)"AND 'SS' IS SECONDS FROM 00-59"
180 PRINT: INPUT "ENTER THE TIME:";K$
190 REM SAVE THE INPUT DATA IN TI$
200 TI$=K$
210 REM DISPLAY THE CURRENT TIME
220 PRINT "⊐": PRINT: PRINT: PRINT
230 PRINT "CURRENT TIME IS"
240 PRINT: PRINT: PRINT "PRESS ANY KEY TO STOP THE PROGRAM.....□□□"
250 K$=TI$
260 PRINT TAB(16)"⊐"LEFT$(K$,2)":"MID$(K$,3,2)":"RIGHT$(K$,2)
270 GET K$: IF K$="" THEN GOTO 250
```

```
100 REM-------- PROGRAM 'B' -----------
110 REM TITLE AND INPUT INSTRUCTIONS
120 PRINT "⊐"TAB(13)"KEEPING TIME": PRINT
130 PRINT TAB(13)"PROGRAM 'B'": PRINT: PRINT
140 PRINT "ENTER THE CURRENT TIME IN 'HHMMSS'"
150 PRINT "FORMAT WHERE 'HH' IS HOURS   FROM 00-23"
160 PRINT TAB(13)"'MM' IS MINUTES FROM 00-59"
170 PRINT TAB(9)"AND 'SS' IS SECONDS FROM 00-59"
180 PRINT: INPUT "ENTER THE TIME:";K$
190 IF LEN(K$)<>6 THEN GOTO 310
200 IF INT(VAL(LEFT$(K$,2)))>23 OR INT(VAL(LEFT$(K$,2)))<0 THEN GOTO 310.
210 IF INT(VAL(MID$(K$,3,2)))>59 OR INT(VAL(MID$(K$,3,2)))<0 THEN GOTO 310
220 IF INT(VAL(RIGHT$(K$,2)))>59 OR INT(VAL(RIGHT$(K$,2)))<0 THEN GOTO 310
230 FOR I=1 TO 6
240 IF ASC(MID$(K$,I,1))<48 OR ASC(MID$(K$,I,1))>57 THEN GOTO 310
250 NEXT: CIA=56320
260 I=VAL(LEFT$(K$,2)): Z=0: IF I>11 THEN Z=1: I=I-12
270 POKE CIA+11,Z*128+INT(I/10)*16+I-INT(I/10)*10
280 I=VAL(MID$(K$,3,2)): POKE CIA+10,INT(I/10)*16+I-INT(I/10)*10
290 I=VAL(RIGHT$(K$,2)): POKE CIA+9,INT(I/10)*16+I-INT(I/10)*10
300 POKE CIA+8,0: GOTO 330
310 PRINT "▨INVALID DATA ENTERED!⊐"
320 FOR I=1 TO 2000: NEXT: GOTO 120
330 PRINT: PRINT: PRINT "TIME STORED IN THE CIA TOD REGISTERS"
340 END
```

```
100 REM-------- PROGRAM 'C' -----------
110 REM DISPLAY THE CURRENT TIME
120 PRINT "⊐": PRINT: PRINT: PRINT
130 PRINT "CURRENT TIME IS"
140 PRINT: PRINT: PRINT "PRESS ANY KEY TO STOP THE PROGRAM.....□□□"
150 PRINT "⊐"TAB(16);
160 CIA=56320: I=PEEK(CIA+11): Z=(1+((I AND 128)=0))*12: I=I AND 127
170 I=Z+INT(I/16)*10+I-INT(I/16)*16: K$=RIGHT$("0"+RIGHT$(STR$(I),1-(I>9)),2)
180 I=PEEK(CIA+10): I=INT(I/16)*10+I-INT(I/16)*16
190 K$=K$+RIGHT$("0"+RIGHT$(STR$(I),1-(I>9)),2)
200 I=PEEK(CIA+9): I=INT(I/16)*10+I-INT(I/16)*16
210 K$=K$+RIGHT$("0"+RIGHT$(STR$(I),1-(I>9)),2): I=PEEK(CIA+8)
220 PRINT LEFT$(K$,2)":"MID$(K$,3,2)":"RIGHT$(K$,2)"."RIGHT$(STR$(I),1)
230 GET K$: IF K$="" THEN GOTO 150
```

C-64 THERMOSTAT

This article describes a hardware project for those of you who are already past the beginner's stage in hardware building.

John Vanderkooy
Waterloo, ON

These are the times to save energy and, to promote this end, my son and I (he's the real programmer) decided to use our C-64 computer to act as a high-level controller for the furnace. (This summer we'll work on the air conditioner). The beauty of a computer is its programmability, allowing temperature variation to give comfort in the home with the best energy savings. For families with a fixed schedule, the temperature can be programmed well in advance, and to aid the non-programmers (who vastly outnumber us) an idiot-proof menu can be designed, using the keyboard. All the foregoing, of course, is simply an attempt to justify the existence of our home computer!!

The Commodore 64 (or VIC 20) lends itself very well to the above purpose. The temperature can be measured by using an appropriate thermistor on the game port paddle input; in fact, four temperatures could be so dealt with. It might be useful to have the outside as well as the inside temperature, for example, since one could then integrate the temperature difference to determine the number of degree-days of heat load for the furnace. If you have a printer, you can print out all of that and other data to impress your neighbours. Think of how much money you could save if you printed out the temperature every few minutes, so that you'd go through a box of Z-fold paper every fortnight. Seriously, though, I feel this project is worthwhile. It will save energy, teach some elementary interfacing techniques and tends to bring out the best in programming, because the end result really matters.

More on the method of connecting up the computer. I used a 6 kilo-ohm nominal thermistor, mounted just below the normal thermostat. The unit connects via thin wires and a jack to pins 7 and 9 of control port 1 (for safety, you might put a 220-ohm resistor in series with the connection to pin 7, which is the +5 volt lifeblood supply of the computer). If you make no other modifications, the value of POTAX found by peeking memory location 54297 will be very low, quite unsuitable for accurate temperature determination. It is desirable to have a PEEK(54297) value of about 200. That allows variation up to 255 (the maximum which occurs for low resistance; thermistor warm) and downward variations (which represent a cooler thermistor). The C-64 and VIC 20 have 1000 pf timing capacitors on the game ports, so that 470 kilo-ohms represents (nominally) a full scale reading of 255 (see *C-64 Programmer's Reference Guide*, p. 472). To use my 6k thermistor, I paralleled the internal capacitor with a 43 nanofarad polystyrene unit, connected to the same jack to pins 8 and 9 of control port 1. **Figure** 1 shows the connections to the game port. Almost any type of capacitor will do (except an electrolytic), but it should not be sensitive to humidity variations, since this would falsify the temperature reading. For my computer, I chose not to use an interrupt routine to read POTAX (*C-64 Programmer's Reference Guide*, p. 346), since this might compromise the real-time clock accuracy. Those with a C-64 link will have to place it at $c000 for proper clock operation. The C-64 uses an analog switch (U28, a 4066) under control from the keyboard scanning CIA (U1, a 6526). For a game, this would be a serious limitation, but for thermostatic control, the keyboard sees very occasional action, unless you make it work under interrupt control while you play your games! The VIC 20 doesn't have this problem; maybe some unused VIC 20's will end up as thermostats!

To prevent any contention, I wired a shorting switch across the 4066, but it doesn't really make any significant difference. In reading the thermistor "temperature", there is a slight variation in PEEK(54297), but by averaging 100 of these values this allows an increase in the relative accuracy with which the temperature can be determined. Our setup achieves a resolution better than 1/10 Celsius degree. Calibration is easily performed by using an accurate mercury-in-glass thermometer to measure the readings at, say, 15 and 20 degrees Celsius, and assuming a linear interpolation.

Now the other half of the problem; how to turn the furnace on and off? I contemplated this for some time and came up with a scheme that affords some protection from computer foibles or power failures. **Figure** 2 shows how a small 12 volt relay can be controlled from the user port. My relay pulls in nicely with anything over about 8 volts, so the 9 volt ac at the user port can be rectified and smoothed to provide an adequate supply. The relay can be driven by a single NPN transistor, which has base drive from the MSB bit of the user port. The program can easily set this bit to output via the DDR, and these steps are incorporated in the program which is given later. The base resistors are chosen so that the relay is normally off even when the user port is not active as an output. Almost any NPN transistor will do. The diode across the relay coil prevents the large spike which accompanies the turnoff of an inductive relay coil. Connections to the 24 volt ac control system of a normal thermostat should not be made with very thin wire; there is a substantial current required to properly operate the gas valve or relay on an oil furnace. This summer we got a central air conditioner, so I had an extra thermostat at my disposal. **Figure** 2 also shows how the two thermostats are used to advantage in a heating situation. Thermostat A sets an upper limit temperature which cannot be exceeded by any computer directive, since it shunts the control relay R. If the house temperature lies between the two setpoint limits, the computer can turn the furnace on and off with the relay. In my setup, A is set to about 21 Celsius and B to 14 Celsius. Our daytime computer controlling temperature is set to 20 degrees Celsius, and at night or when we're away, it is near 15 degrees Celsius. Any lower and the plants complain of numbness. This deployment of the two thermostats gives safety, flexibility and a reasonable default situation. If only one thermostat is available, I would use it like B (with A shorted), since this gives protection against freezing when the youngsters in the house forget about the computer's prime directive and turn to **JUMPMAN** instead. Purchase of a second thermostat would be nicer and certainly much cheaper than a clock thermostat. Electric heating thermostats cost about $15.00. Before the computer control, I had used the two thermostats as in **figure** 2, with a timer switching the relay for night

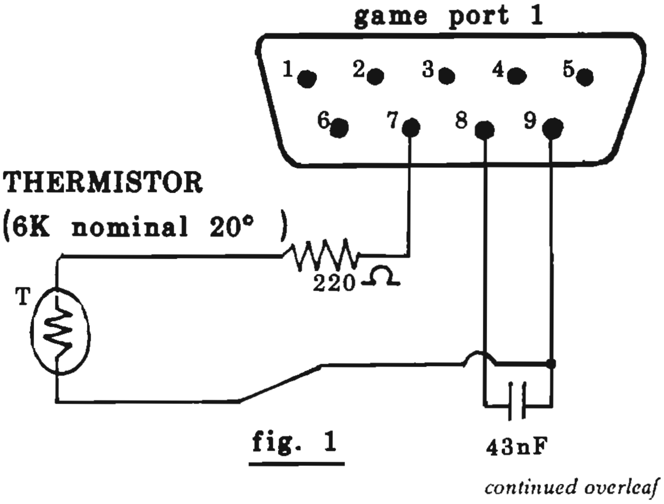


fig. 1

continued overleaf

setback, using the normally closed contacts, so that A can be set to the normal temperature during those times (or left there). The control program can readily be modified to use either relay sense or polarity.

An important point concerns the operation of the temperature-regulating algorithm. My program models the action of a normal thermostat. When it switches on or off, the motion of the mercury blobs causes a hysteresis so that switching it again requires a considerable change in temperature, perhaps as much as 1 or 2 degrees. This makes the action decisive but would create large temperature excursions if unchecked and result in a very low cycling rate, due to the large thermal inertia of the house. To combat this effect, a small heating coil is incorporated near the bimetallic temperature sensing spiral, which is activated by the current in the control line. This heater anticipates the rise in temperature and acts to decrease the cycling time. The combined effect of the two mechanisms produces a more constant cycling interval, in which the on-off duty cycle regulates the temperature. My programs incorporate hysteresis (about 1 degree) and anticipation, the latter by allowing the hysteresis to decay exponentially with a time constant of around four minutes. It's really nice to have a computer at this point; almost any variation in the algorithm can be easily tried. I have been told that 6 cycles per hour is ideal for a gas furnace, 3 for oil; that is easily arranged. In fact, it would be fairly simple to make the controller have a fixed cycle rate; one simply establishes a repetitive triangular waveform with an equivalent amplitude of, say, 1 degree, and uses a somewhat modified method to create modulation of the on-off duty cycle. The regulation tolerance and cycle time are related, but at least we have a number of interesting control possibilities. In our programming, we tend to make step changes in the temperature (for example, in the morning), but the internal computer "set-point" could be smoothly altered. Who knows what marvellous dreams might be conjured up by a once per hour sinusoidal temperature variation during the night!

The regulating program given below is liberally interspersed with REMarks. This short program gives just a regulating algorithm with no frills. A longer program is available from the author by sending him a self-addressed envelope with postage paid. In that program, the menu portion is one view of how programmability can be handled. BASIC is plenty fast enough for the control, and many embellishments are allowable. If the computer is turned off, or there is a power interruption, all control will be lost, of course. This is not serious if you have wired up two thermostats as described earlier. An ambitious possibility is to prepare an auto-start eprom at location $8000 which has a small routine to transfer the BASIC program (also in the eprom) to start at $0800, and enter the appropriate commands into the keyboard buffer to restart the control, bypassing the menu. Naturally, the time will be lost, so some nominal set-point must be chosen, for later updating by the disgruntled homeowner. I would be happy to hear about anyone's experiences with software "Thermostatgraphy", but I wish to emphasize that anyone's ill consequences encountered are not my responsibility: you are on your own!

Already I am giving thought to how I can control our air conditioner this summer, in such a way that humidity and the outside temperature are included in some "comfort algorithm". One could even include a cadmium-selenide photocell on the game port to incorporate action based on the solar heat input!

The programs tend to end up with a lot of variables. I recommend that you try a simple regulation version first, then adorn it with your own personality. We have a longer version to calculate the duty cycle and use the whole screen to display the recent furnace behaviour. One could even use a graphic display. Keep the wiring to the thermostat and furnace reliable; the rest of the family will not be amused if there's trouble while you're away. Perhaps a simple defeat switch would be advisable. *TPUG*

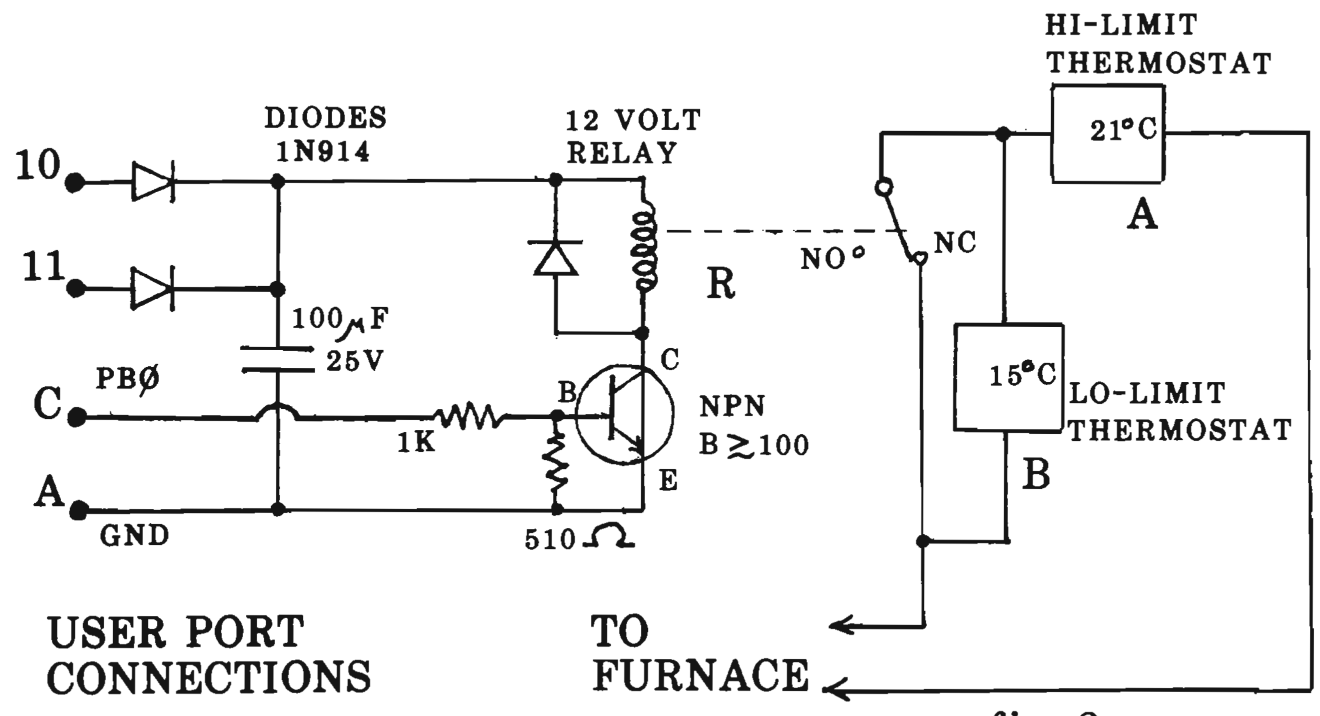


fig. 2

THE NEW SYS

Robert Rockefeller
Langton, ON

When the software engineers at Commodore were designing the C-64 computer, they did not program a new BASIC language for it. Instead, they used BASIC 2.0, which had been written for the second generation of the PET computers. This version of BASIC contained several "bugs", most of which have been corrected on the C-64.

Besides the corrections made, several improvements were made as well. One of the BASIC commands which was improved was the SYS command. It is now possible to pre-load the processor registers from BASIC for use by a machine language routine called by SYS. This increases the power of SYS immensely, and especially gives the BASIC programmer easy access to his computer's ROM (read-only memory) routines.

SYS used to work this way: first the number after SYS was converted to binary and stored as a two-byte integer on page zero, in location 20. This number of course is the start of your routine in memory. An indirect jump was then made to the location represented by this number and the machine language program was executed until the RTS (return from subroutine) instruction was encountered, at which time control of the 6502 CPU was returned to the BASIC interpreter. SYS still works in this manner with this exception; the Accumulator, the X-Index register, the Y-Index register and the Status register are loaded with the contents of four memory locations on page three before the called machine code routine is executed. Also, the contents of the registers are reproduced in these locations after RTS is executed in the called routine. The relevant memory locations are:

$030c	**780**	**—Storage for 6502 Accumulator**
$030d	**781**	**—Storage for X-Index Register**
$030e	**782**	**—Storage for Y-Index Register**
$030f	**783**	**—Storage for Processor Status**

To illustrate the ability of the "new and improved" SYS command to control ROM routines I have written a short BASIC program which uses four of the Kernal (VIC operating system) routines to save any portion of memory to disk or any portion of memory below 32K to cassette. The four routines used are SETLFS, SETNAM, SAVE and CLOSE. You should have no trouble learning to use your ROM routines if you study the program. Of course, it is necessary to understand the signifigance of each routine before it can be used. *The C-64 Programmer's Reference Guide* contains a section explaining most of the operating system routines. This is a good place to start.

To sum up then, the procedure for passing parameters to a ROM routine or any machine language routine is this; POKE the value you wish the processor registers to contain on entry to the machine code routine to the requisite memory locations listed above, then SYS to the routine. That's all there is to it. Happy SYS'ing. *TPUG*

```
10 PRINTCHR$(147)SPC(3)"HIT RETURN AFTER"SPC(7)"EACH ENTRY."
20 A=780:X=781:Y=782
30 REM
40 REM SET LOGICAL, FIRST & SECONDARY ADDRESSES
50 REM
60 PRINT:PRINT"DO YOU WISH TO SAVE TO":PRINT" CASSETTE, OR":PRINT" DISK":INPUT DEVIC$
70 IFLEFT$(DEVIC$,1)="C"THENDEVIC=1
80 IFLEFT$(DEVIC$,1)="D"THEN DEVIC=8
90 IFLEFT$(DEVIC$,1)<"C"ORLEFT$(DEVIC$,1)>"D"THEN60
100 PRINT:PRINT"WHAT TYPE SAVE":PRINT"0. RELOCATABLE":PRINT"1. NON-RELOCATABLE"
110 INPUTSAV$:SAV=INT(VAL(SAV$)):IFSAV<0ORSAV>1THEN100
120 POKEA,1:POKEX,DE:POKEY,SA:SYS65466:REM SETLFS
130 REM
140 REM SET PROGRAM NAME
150 REM
160 PRINT:PRINT"ENTER START,END,NAME"
170 INPUTSR$,EN$,NA$:SR=INT(VAL(SR$)):EN=INT(VAL(EN$))
180 FORJ=1TOLEN(NAME$):POKE511+J,ASC(MID$(NAME$,J,1)):NEXTJ
190 POKEA,LEN(NAME$):POKEX,0:POKEY,2:SYS65469:REM SETNAM
200 REM
210 REM SAVE PROGRAM
220 REM
230 SR=SR/256:POKE252,INT(SR):POKE251,(SR-INT(SR))*256:POKEA,251
240 EN=EN/256:POKEY,INT(EN):POKEX,(EN-INT(EN))*256::SYS65496:REM SAVE
250 END
```

Wizard

Malcolm O'Brien
Toronto, ON

WIZARD For The C-64
From: Progressive Peripherals, 2186 South Holly, Suite #2, Denver CO 80222 USA (303) 759-5713. $39.95

Go out and buy this game right now! This is my favourite! I've spent a zillion hours with it and still haven't seen it all! There are forty screens and if that's not enough, the **WIZARD Construction Set** allows you to create your own screens. You can put up to one hundred screens on a disk! And when that disk is full, start another!

Now that I've expressed my exuberance, I'll describe the game. **WIZARD** is a climbing game, basically, but this can scarcely convey the richness of the game. There is such a wide range of possibilities (including user-created ones), you are certain to be enchanted.

The magicians behind all this are named Steven Luedders and S.A. Moore. Both are High Adepts of the Ancient Mystical Order of Programmers and have infiltrated an organization called Progressive Peripherals and Software which markets their arcane creation.

Let me set the stage for you. You have to guide Wilfred the Wizard through these innumerable screens. You collect treasures and avoid the "bad guys". Most importantly, get the key! Once you have the key, take it to the keyhole. At this point, the wizard turns into a wisp of smoke and goes through the keyhole to the next level. The first screen is called "Playground" because nothing chases you. You can roam around at will and discover what you can do and what kind of things will happen. These are numerous, but I'll give you an idea.

You can climb ropes, ladders and stairs. But be careful — ropes can move or disappear or become invisible; ladders do the same. Stairs may smooth out to become slides and all manner of things can be waiting for you at the bottom! You can jump and duck — and when the Phantom Archer is shooting at you, you'll be glad you can jump and duck! You can ride the elevators up or down or be teleported by entering magic portals. And you can cast spells!

Here's a tip: the quicker you get the key and go through the keyhole, the higher your bonus points. This is easier said than done and becomes increasingly difficult as you move up the levels. I had to play the level called "Rock 'n Roll" many times before I figured out how to stay alive, let alone advance to the next level. It was quite a while before I figured out what to do on "Totally Tubular" and I rarely survive "Great Balls of Fire".

Whew! Sound tough? If you think so, create an easy version. The High Adepts have seen to it that *you* may use their enchantment as you will to create your own ethereal worlds. But beware — many a wizard has fallen. And I do mean fallen.

Whenever you lose a wizard he falls down to the bottom of the screen, complete with a circle of stars rotating above his head. But an interesting thing happens: his wizard's hat falls off first. It usually comes to rest on a higher level, but the wizard always falls all the way down. This is an amusing addition to what is already superb animation. The sound is equally superb. It's more like the soundtrack from a cartoon than the sound effects of a computer game.

Wilfred the Wizard is one of the most lovable and charming characters I've ever seen in a computer game. You're certain to feel very sympathetic when his hat falls off and he tumbles down to the bottom of the screen. He's part of the reason why this game has a compelling playability.

As if all this wasn't enough, there's more. Progressive Peripherals and Software intend to have the **WIZARD Expansion Set** available in the Fall of this year. The Expansion Set will feature a new Construction Set program with expanded advanced features, the top twenty levels sent in from current users of the program (the competition will end Sept. 30, 1984), and twenty other all-new levels!

We're talking serious gaming here! A few more points should be noted: one to six players, four difficulty levels plus Customized and Mystery (random screens) for a total of six game options, ten selectable play speeds (including too fast!), top scores are saved to disk, fast-running demo and 11 pages (8½ by 11) of excellent documentation.

There's only one thing to do. Go out and buy this game right now! And hold onto your hat! *TPUG*

Trivia

George Shirinian
Toronto, ON

TRIVIA. For Commodore VIC-20 and 64. Cymbal Software. VIC 20 disk $39.95, C-64 disk $49.95.

"Welcome to the wonderful world of **TRIVIA**, where facts from all walks of space and time converge in a medley of mind-boggling questions." It is surprising that someone has not come out with a computer version of the game that swept the country last Christmas season before now. **TRIVIA** is not a direct copy of the popular board game but is a highly entertaining quiz program in its own right.

The game contains 1600 questions in the C-64 version, 1000 for the VIC 20, and is available, in good Canadian tradition, in both English and French. It does not utilize the familiar game board of the "manual" version, and has a different logic to the play. When beginning a game you can select a "think-time" of 10, 15 or 20 seconds and a game length of 50, 100 or 150 points. Different questions have different points. There can be from 1 to 6 players.

The questions are presented on a colourful screen, but there is no particular graphic display. There are sound effects for correct and incorrect answers, but, again, these are unremarkable. The real excitement in this game is to come up with the right answers to the questions and type them in correctly, while watching the seconds fly by on the screen's timer.

One feature I liked is that questions are randomized, so that no two games play the same. This is essential in a program that will be played over and over. Secondly, I was quite impressed to discover the program gives partial points to a close answer, giving poor spellers or typists some encouragement. The questions are drawn from both American and Canadian experience and are interesting, challenging and fair. Topics range from TV and movies to sports, history, etc.

The first person to reach the game score wins. Statistics are displayed at the end of the game and reveal some very interesting facts. The person with the highest percentage of correct answers does not necessarily win the game. The variable points for the questions can help a weaker player win, if he or she gets enough of the higher ones. This makes the game more exciting and fun all around, as everyone has a real chance to win. *TPUG*

Creature Creator

Rich Westerman
St. Anne, IL

CREATURE CREATOR From Designware. $39.95.
System Requirements: Commodore 64, Disk Drive
Classification: Education (ages 4 to 8)

CREATURE CREATOR is a "gentle" introduction to programming—but the program's layout and operation would lead you to believe that it is a game. It has hi-res graphics, animation and sound effects, but these are used to disguise a tutorial on elementary programming structure.

CREATURE CREATOR is all menu-driven, and the introductory screens and activities are all controlled by only two keys, making it easily accessible to the new computer user.

Here's how it works: I'm sure you've heard of the "Identi-kits" used by police departments to reconstruct faces from witnesses' descriptions. **CREATURE CREATOR** is an identi-kit for creating monsters. Using only two keys, and prompted by simple questions and commands, you can create a composite cartoon-like character. Combining heads, bodies, arms and legs from seven selections in each body area, dozens of combinations are possible. When you have constructed a creepy critter on the screen, you can then make it "dance" by commanding it to wave, hop, stomp, grab, turn its head or even roar. The DANCE portion of **CREATURE CREATOR** even allows you to combine a series of those commands into a program.

Sooner or later, a child will press the key which says GAME and **CREATURE CREATOR** will begin doing its intended job of teaching the rudiments of programming logic. The GAME board shows two monsters side-by-side on the screen. At the bottom of the screen is a grid of boxes. The monster on the left is the computer's and the one on the right is yours. The computer will make its monster go through a set of "dancing" motions and it's your job to give your creature the correct commands which will make it duplicate the motions of the other creature. The grid at the bottom of the screen records your commands and are easily edited should you make errors. Multiple difficulty levels are available in the GAME portion of **CREATURE CREATOR**, allowing a measure of additional challenge.

CREATURE CREATOR is an engaging, colorful and well-designed educational program. It is perhaps overpriced in the wake of new arrivals of low-priced educational software for the Commodore 64.

CREATURE CREATOR also suffers from too much copy-protection. It is sometimes necessary to re-load the program two or three times before it will run correctly, and the wear and tear on the drive (caused by this protection method) is an annoying factor. *TPUG*

Kids On Keys

Rich Westerman
St. Anne, IL

KIDS ON KEYS From Spinnaker. $34.95. Disk For Commodore 64. Recommended for ages 3 – 7.

Perhaps the largest obstacle to overcome when convincing kids to move from joystick wiggling to keyboard usage is that, until you're familiar with the keyboard, you can't manipulate the computer with the speed and facility that you're used to with the joystick. It's hunt and peck – and, to most kids, that's just no fun.

There is a growing body of software appearing for the C-64 which attempts to tackle this problem. I'm referring to the "keyboard familiarity" programs, and **KIDS ON KEYS** is Spinnaker's entry into this market. **KIDS ON KEYS** is nothing less than a typing tutor for young children. It offers the child three separate arcade-style games, which are played using the keyboard. Game One requires the player to type the letter that slowly falls from the top of the screen. If the proper key is tapped before the letter reaches the bottom, the letter explodes and then appears at the top of the screen as a "trophy". This continues until fifteen letters have been "popped". You then get to play a bonus round. A child in a balloon slowly falls towards the bottom of the screen, festive music plays, the child in the balloon waves and the balloon again takes to the sky. Success in the bonus round will prompt the computer to ask the child if he wants to go to the next level of difficulty or go back to the main menu.

Games Two and Three require the player to identify different objects on the screen within a time limit, the duration of which is determined by the difficulty level.

It was in Games Two and Three that I found a serious problem in *my* copy of the software. **KIDS ON KEYS** contains files of objects that the main program loads. These files contain the graphic routines that create the objects the child has to identify. The authors of **KIDS ON KEYS** wisely realized that if the program is played often by the same children, the challenge of typing the names of the objects will disappear. Thus, an option to create and name new objects and then SAVE them to a separate disk was provided. The program gave me easy-to-understand prompts for the formatting of the data disk, and the joystick-driven drawing routine worked fine. When I tried to SAVE my new file of objects to the data disk, however, nothing happened. Try as I might, I couldn't get any new files SAVEd.

If the "MAKE PICTURES" option was never used in **KIDS ON KEYS**, the program would be quite impressive. The graphics are bright and colorful and the music and sound effects (while not as sophisticated as other offerings by Spinnaker) are adequate. The program can be run quite easily by kids of the recommended age group (what with all the prompts and excellent error-trapping, it seems to be crash-proof). Too bad my copy couldn't live up to its own manual's promises. *TPUG*

COMMODORE 64 COLOR GRAPHICS: A Beginner's Guide

John Moore
Windsor, ON

As a Commodore Computer owner and user for over three years now, I've always been curious and fascinated by the use of graphics. I played around with my 'old' PET character set, creating little pictures and then making them move around the screen in a variety of ways, and adding little graphic rewards in the educational programs with which I dabbled. One thing that encouraged me to trade in the PET for the new Commodore 64 was its unique graphic capabilities. High-res pictures and Sprites were a real novelty for me.

Adding colour and new twists to my old PET graphics has been fun, and playing with the **Koala Pad** and other high-res packages has been a real blast. But doing high-res pictures on my own has seemed to be more difficult than I could manage. At least, it was, until a friend introduced me to *COMMODORE 64 COLOR GRAPHICS: A Beginner's Guide.* This easy-to-follow, step-by-step guide shows you how to create your own detailed pictures while providing you with a "toolkit" of subroutines to accomplish this task. Ever bought a "beginners guide" that any idiot should understand, only to be blown away in the first chapter and left wondering how come you're so dumb? This will *not* happen with *COMMODORE 64 COLOR GRAPHICS: A Beginner's Guide.*

The Introduction in Chapter 1 assumes nothing. It starts by suggesting you turn your equipment "on" and tells you where to find the switch. It gives a few BASIC tips, a chart on Commodore Colors· on what keys to find them, and some tips about the program listings found in the book. It tells how to key them in, how to RUN them, SAVE and VERIFY, what to do if problems occur, and all those things that so many "beginners guides" assume you already know. So far, so good: even this 'idiot' could understand!

The next four chapters take you one small step at a time through the processes of turning on the high-res screen, setting colours, locating and plotting points, drawing lines, painting shapes and everything else necessary to create a detailed picture. The book provides the data needed to draw the picture that appears on the inside front page of the book. Every chapter also provides exercises that encourage the reader to experiment and create other designs and shapes. Every subroutine is explained in detail. It tells you what is happening, why it is happening and gives additional examples of what each routine is used for. All this in plain English!

The final chapter covers Sprites and Sprite movement. In the same plain English, and with the same attention to detail, the mysteries of Sprite creation, colour and movement are explained.

Again, several other examples are provided and the reader is encouraged to experiment with other designs.

Altogether I think *COMMODORE 64 COLOR GRAPHICS: A Beginner's Guide* does a very thorough job of explaining and teaching high-res graphics in a manner that any novice can really understand. But that's not the end of the story — or of the book either!

The Postscript provides very good ideas for additional uses for the program, along with more Sprites, including the data to make them. This opens up another whole new area of experimentation. Appendix A is a programmer's trouble-shooting guide.

It includes several preventive measures, as well as common cures for problems you might encounter. Appendix B provides a complete listing of all the subroutines in the 'toolkit'. Appendix C gives the listings of several additional 'tools', such as drawing and painting rectangles, polygons and circles, along with the usual complete explanation of what's happening and why.

The very important and useful Appendix D shows how you can speed up the subroutines with the use of machine language. The one thing that is most disturbing about high-res in BASIC is that it is so *slow.* Appendix D begins to solve this problem by replacing six of the most common routines with machine language equivalents, which are read in from BASIC. A brief explanation is followed by the six program listings. The little extra time spent keying in these extra listings is more than made up by the increased speed of the drawing routines.

The last few pages of the book include charts and graphs of screen locations, colour, Sprites and a toolkit reference guide. I'm sure I've more than got my $16.95 worth out of my tattered copy *TPUG*

COMMODORE 64 COLOR GRAPHICS: A Beginner's Guide
By: Shaffer & Shaffer Applied Research & Development
Publisher: The Book Company

Defending The Galaxy

Lorien Gabel
Toronto, ON

Defending The Galaxy is a book that covers almost all areas of the video gamer's world. It purports to be "the first complete guide to the entire world of The Games".

When you first flip through this book you are barraged by a number of assorted topics relating in some way to the video game world. Some of the topics presented are sections on how to tell the games apart, a list of different enemies that you will eventually be facing, tips for survival in such games as **Tron, Stargate, Turbo** etc., and a chapter on the evolution of the video games.

Included is a very interesting list of all the video games ever released and a short explanation about each (153 in all). There is also a guide to the most popular video game places in the U.S., state by state. Sadly, there is nothing in this section that includes Canada.

Also included is a list of the high scores for the most popular games, set at that time. Most of these have been broken by now but it is still helpful in that you can gauge your own scores by it.

The major deficiency (if you take *Defending The Galaxy* seriously as a guide to video gaming) is that it is outdated. It was released in late 1982 and doesn't have a thing on the new and very popular video disk games. Therefore, the hints about such games as **Turbo, Space Invaders and Zaxxon** are not very helpful. However, this doesn't pose a serious problem since much of this book refers to the video world in general.

I found that the book was not very satisfying in really teaching you anything about video games, but it is doubtful whether or not it is really trying to do so. Most of the book approaches video gaming from a humourous angle: e.g. Video Virgins, jamming, packing, and such aspects. Realising that his book will not teach you much in the way of serious gaming, the author has graciously added a list of books that *do* teach you how to play.

In conclusion, *Defending The Galaxy* is a *good* book if you are looking for something that is generally informative, entertaining and somewhat far from the serious side of video gaming. If you are looking for a book on how to master the games, you should look elsewhere. *TPUG*

DEFENDING THE GALAXY from Triad Publishing Co. by Michael Rubin. $4.95 (U. S.)

This is the third issue of *TPUG Magazine* which includes *Product Parade* — our regular feature presenting new, as well as not so new, software and hardware available on the market. As we depend on the information sent to TPUG by manufacturers and distributors, it is not always possible to strike the right balance between software and hardware products described in the feature. This month's *Product Parade*, for example, includes mostly software products. But we do have a lot of interesting new product announcements, both hardware and software, for the November issue. This is to inspire you with some Christmas gift ideas. Remember, however, that *Product Parade*, does not review products, but simply provides information.

Memory expander for the Commodore 64 from LETCO

LETCO (Leader Electronic Technology Company), the pioneer of the popular 64K memory expander for the VIC 20, has developed a 64K byte **memory expander** for the C-64. It plugs directly into the C-64 expansion port and provides eight separate 8K blocks of paged memory. Each page is selected by a single POKE instruction. Those users who already have the 64K memory expander for the VIC 20 need to purchase the **adapter** only to use their 64K **memory expander** on the Commodore 64.

The adapter (Model 64KVA) is priced at $29.95 U.S. The memory (Model 64KV) for use on the VIC 20, is priced at $109.95 U.S. **The combination (Model 64KC)** for use on the C-64, is priced at $139.95 U.S. All products are available directly from Leader Electronic Technology Company LETCO, 7310 Wells Rd., Plain City, OH 43064.

☆

Xref HELPER from (M)agreeable software, inc.

Xref HELPER prints a cross-reference listing to help the user find where a particular value came from in his Commodore 64 BASIC program.
Xref HELPER scans the program on diskette for all occurances of variables, constants, GOTOs, and GOSUBs. As it scans it inserts each name, and the number of the line on which it occurred, into a list. **Xref HELPER** orders this list alphabetically or numerically within each group. It also orders the variables by type; for example, integer or string. When **Xref HELPER** has completed the scan of the program, it sends the ordered list to the printer.
Written in KMMM Pascal, **Xref HELPER** scans small programs in a few seconds, medium programs in one to five minutes, and large program in five to ten minutes.

Xref HELPER is available from (M)agreeable software, inc., 5925 Magnolia Drive Plymouth MN 55442, (612) 559-1108 for $25.00 U.S. plus $1.25 U.S. shipping. Canadian residents may pay $32.40 Cdn. plus $1.60 Cdn shipping.

ZOOM Pascal 64 from Abacus Software

ZOOM Pascal 64 for C-64 owners offers the programming convenience of structured high-level language. Data types include floating point reals, integers, strings and scalars. The **ZOOM Pascal 64** allows also accessing conventional text files whether on disk or tape.

The **ZOOM Pascal 64** package consists of:
* EDITOR — to create, save and modify Pascal source language statements
* COMPILER — to translate the Pascal source language statements into intermediate code
* TRANSLATOR — to translate the intermediate code into actual 6502 machine language
* Comprehensive user's manual
* Sample programs to show how to get the most from ZOOM Pascal 64.

ZOOM Pascal 64 comes on diskette only for $39.95 U.S. Available from the local dealer or directly from ABACUS Software, P.O. Box 7211, Grand Rapids, MI 49510, (616) 241-5510

☆

TINY FORTH from Abacus Software

TINY FORTH is a high-level language to run on Commodore 64 or VIC 20 (with 8K expander). **TINY FORTH** comes with well over 200 words in its vocabulary. But it is extensible — the user can add his own words to this built-in vocabulary. **TINY FORTH** includes also a full-screen editor for creating new "screens" that can be saved to disk or tape.

The 45-page manual which comes with the **TINY FORTH** introduces the user to the FORTH language.

The price for **TINY FORTH** on cassette (for both C-64 and VIC 20) is $19.95 U.S. The version on diskette (for both C-64 and VIC 20) costs $22.95 U.S. Available from the local dealer or directly from ABACUS Software, P.O. Box 7211 Grand Rapids, MI 49510, (616)241-5510

Graphics Designer 64 from Abacus Software

Graphics Designer 64 for Commodore 64 allows designing hi-res or lo-res graphics. It can be used for architectural design, engineering graphics, graphic arts, artistic expression, and more.
The **Graphics Designer 64** Editor lets the user interactively create his picture. The Editor is completely menu driven. No programming is necessary. The user can edit a picture from the keyboard or with joystick or a suitable light pen. Finished pictures can be saved to diskette. The Slide Show feature makes it possible to combine ready pictures for later display. The user can also obtain hardcopy of his hi-res screen on the following printers: VC-1515, 1525E, Epson or Gemini printer.
Graphics Designer 64 (on diskette only) costs $34.95 U.S. Available from the local dealer or directly from ABACUS Software, P.O. Box 7211 Grand Rapids, MI 49510, (616) 241-5510

✰

GRAPH-TERM 64 from Bennett Software Company

GRAPH-TERM 64 for the Commodore 64 is a graphics terminal emulator program which prints text and high-resolution plots generated by a mainframe computer. It is thus of particular interest to scientists and engineers who use standard graphics programs generating plot files in Tektronix format.
Because it is written totally in machine language, **GRAPH-TERM** 64 not only can display high-resolution graphs as they are transmitted but can download the plot files and replay them up to 20 times faster. It can also produce hard copies of the plots on the Commodore 1520 plotter.
GRAPH-TERM 64 comes with a number of simple BASIC programs for saving and loading plot files and for illustrating the use of machine language subroutines in it.
In summary, **GRAPH-TERM** allows the user to:
* view Tektronix format plots generated by a mainframe computer
* download text or plot files
* generate plot files on the Commodore 64
* preview plots on the high resolution screen
* create hard copies of the plots on the Commodore 1520 Plotter.

GRAPH-TERM 64 may be ordered from Bennett Software Company, 3465 Yellowstone Drive, Ann Arbor, Michigan 48105. The price is $49.95 + 4.00 shipping and handling (U.S. funds).

✰

SPELL NOW from Access Software Incorporated

SPELL NOW an educational program for the C-64 by Todd Witzel, is designed to help children improve their spelling ability. The program, endorsed by educators, allows children to improve the spelling words that are giving them the most trouble. Up to fifty words can be entered onto a spelling list.
Kids choose from three game options: "See It, Spell It","Quiz Me" and "Hangman". Each game is uniquely designed to make spelling an enjoyable task.

SPELL NOW is available from Access Software, 925 East 900 South Salt Lake City, Utah 84105 for the price of $34.95 U.S.

DISKOVERY distributed by Micron Distributing

DISKOVERY programs promote learning through exciting and challenging game formats.
DISKOVERY has four programs available for children aged 3 to 8 — Alphabet Factory and Match-up (reading skills programs); and Adding Machine and Take Away Zoo (mathematical skills programs). For children aged 6 to 12, The Word Bird and Time Zone are helpful in developing vocabulary, word order, sentence meaning and reading skills.

All of the programs are available for the Commodore 64 and retail for $29.95. Look for the **DISKOVERY** educational programs at local computer stores or contact Micron Distributing, 409 Queen Street West, Toronto ON M5V 2A5 (416) 593-9862 (Toll Free Order Desk 1 (800) 268-9052).

✰

Melodian Keyboard, Melodian ConcertMaster, Melodian MelodyMaster and Melodian RhythmMaster from Melodian Inc.

Melodian Inc. introduced a musical keyboard for the Commodore 64 and three software packages to teach reading and performing music.

The 37-key, 3-octave keyboard is similar in appearance to a portable electric organ. It operates by plugging into a Commodore 64, using the microcomputer's synthesizer to reproduce 16 instrumental sounds.

The Melodian ConcertMaster, Melodian MelodyMaster and **Melodian RhythmMaster** feature entertaining and educational software which can be used either with the **Melodian keyboard** or separately.
The Melodian ConcertMaster combines the capabilities of a music synthesizer, recording studio and video display. Anyone can play the **Melodian ConcertMaster** immediately. No previous knowledge of music is necessary.
The Melodian MelodyMaster teaches music and pitch. After making a selections from a set of prerecorded or computer generated music, the player tries to reproduce the melody, note for note. As each note is played, it is displayed on the screen *in color*, indicating whether it is right or wrong. The computer shows the mistakes, guiding the user's performance until every note is correct.

The Melodian RhythmMaster is similar in concept to the **Melodian MelodyMaster.** It helps to develop perfect timing through the use of color coded video displayed notes.

The Melodian keyboard with the **Melodian ConcertMaster** retails for $200.00 U.S. The price for the Melodian software disks is $39.95 U.S. each.
For the nearest dealer contact: Melodian Inc., 115 Broadway — Suite 1202, New York, NY 10006, (212) 406-5163.

☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆

dealer's MARKETPLACE

Advertiser's Index

classified

This space is limited to TPUG member wanted or for sale items only.

Space cost is 25 cents per word. NO DEALER ADS ACCEPTED