COMMODORE MAGAZINE

VOL. 3
ISSUE 1

Registered by Australia Post
Publication No. NBP4288

REGISTERED FOR POSTING AS A PUBLICATION: CATEGORY B

The objective of this magazine is to disseminate information to all users of Commodore computer products. This magazine contains a variety of information collected from other Commodore publications, and generated locally. Contributions from all Commodore User Groups, and individual users are encouraged.

Advertising space is restricted and allocated on a first-come, first-served basis. Advertising rates can be supplied on request.

All copy and advertising should be addressed to:

The Editor
COMMODORE MAGAZINE
COPUTER REFERENCE GUIDE
284 VICTORIA AVENUE
CHATSWOOD 2067 NSW
AUSTRALIA

ISSUE No.	COPY/ADV DEADLINE	FOR PUBLICATION IN
1	February 18th	March 1983
2	APRIL 11	MAY
3	JUNE 3	JULY
4	JULY 29	AUGUST
5	SEPTEMBER 23	OCTOBER
6	OCTOBER 14	NOVEMBER

(Publication dates and advertising deadlines are currently under review.)

SUBSCRIPTIONS

	Annual Subscription	Single Copy
Postage paid within Australia	$A30.00	$A5.00
Overseas Postage Paid	$A38.00	$A6.00

Subscriptions to COMMODORE MAGAZINE can be obtained from your authorised Commodore Dealer, or from:

COMMODORE MAGAZINE,
COMPUTER REFERENCE GUIDE
284 VICTORIA AVENUE
CHATSWOOD 2066 NSW
AUSTRALIA

Vol 1 1981
Vol 2 1982
Vol 3 1983

Publisher:–
Computer Reference Guide
284 Victoria Avenue
Chatswood
2067 NSW
Phone (02) 419 3277

Production & Typesetting:–
Mervyn Beamish Graphics Pty. Ltd.
82 Alexander Street, Crows Nest
2065 NSW
Phone (02) 439 1827

Printer:–
M.A.P.S. Litho Pty Ltd
Cnr Gibbes and Smith Sts.,
Chatswood 2067 NSW
Phone (02) 406 6433

PLEASE NOTE: To provide a good information service to Commodore users, we will regularly mention equipment, software and services offered by companies and individuals not directly related to Commodore. In doing so, we are not making recommendations, and cannot be responsible for the validity and accuracy of any statements made.

COMMENT

Volume 3 of the COMMODORE MAGAZINE comes in a period of great excitement within the Commodore ranks. Never has there been so much activity – a virtual Commodore compatible software explosion is afoot.

The VIC-20 heralded the new era for the enthusiast and the Commodore 64 is cementing it. The overseas success of these machines is reflected in the software and associated items now coming onto the Australian market.

Software houses which previously seemed only interested in micro's bearing fruitier names are now concentrating much of their efforts on producing for the Commodore market.

Australian distributors and manufacturers are marketing Commodore compatible products with new vigour – catalogs, mail order and even a VIC computer club (with cartridge exchange).

There are many different uses Commodore computers can be put to, from accountancy to typesetting, from statistical analysis to space invaders and beyond.

We hope that this magazine will reflect this diversity in its contents. By its very nature its territory is wide and varied, covering the VIC-20 to the SuperPET and whatever else is in the pipe line.

Glad to have you with us. 1983 is really looking good.

Table of contents

"Why buy just a video game when you can get a full colour computer for the same price."

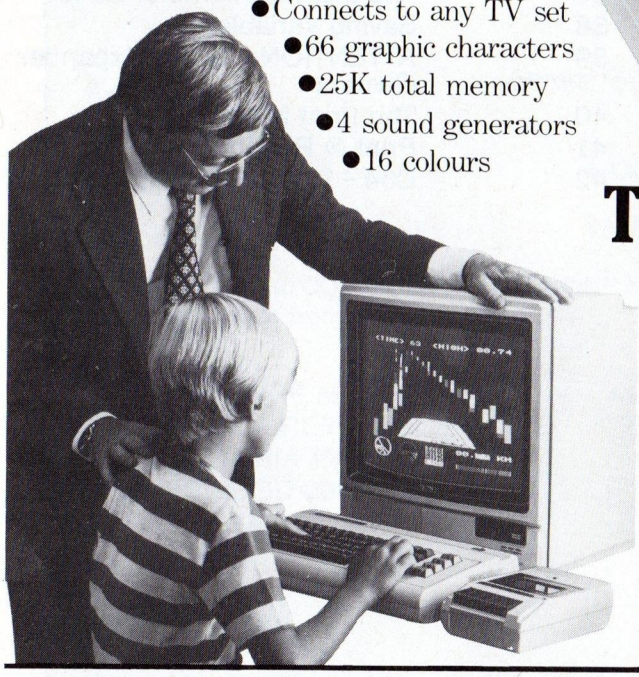
A computer like this would have been science fiction a few years ago. Now it's a reality. It's the Commodore VIC-20, a full-fledged, expandable colour computer that costs the same as a video game.

Everybody loves video games and the VIC-20 has some of the best. But the Commodore VIC-20 can also help the kids with their homework and mum with her budgeting. Dad can even take the light, portable VIC-20 to the office for financial and business applications.

And Commodore has many more applications on the way.

With Full Capability For:

- Education programs
- Recreational programs
- Personal computing
- Includes Microsoft, PET BASIC
- Full-size typewriter-style keyboard
- Easy to follow instruction manual
- Memory expansion to 32K RAM
- Connects to any TV set
- 66 graphic characters
- 25K total memory
- 4 sound generators
- 16 colours



VIC-20

The computer for everyone.

The VIC-20 is the friendliest way we know to learn computing. It has a full computer keyboard even a small child can operate.

It plays music, has exciting graphics and lets you create pictures. It even tells you when you've made a mistake and how to correct it. The VIC-20 can take your children from pre-school through post-graduate studies.

Why get just another game that could end up in the closet? Get an honest-to-goodness home computer for just $299. Get the Commodore VIC-20.

Learn more about Commodore, the home-computer you can depend on. Call or write for the name and location of your Commodore dealer nearest you.

The Commodore Information Centre,
3 Campbell St., Artarmon NSW 2064. Phone: 437 6296.

commodore COMPUTER

MLVL1655

Two Word Processors for the Commodore 64

reviewed by Neil Harris

The latest software for the COMMODORE 64! The Word Machine and Easy Script will be available soon, if not already, as you read this article.

Word processing is one of the single most useful functions for a computer. An editor once compared the difference between a typewriter and a CBM word processor to the difference between chiselling words in stone and molding them from clay. Personally, I'm a two-finger typist, and my production on a plain typewriter is only about 4 pages per hour. On a word processor, I can triple that without any of the aggravation that results from "stupid" typewriters.

What do you do with a word processor? Anything from letters to manuscripts to magazines. I once published an amateur magazine for a local science fiction club, typesetting it with a word processor. I could create neat, justified columns of text. It was a simple matter to take the columns and paste them up into a good-looking format.

EasyScript,

for the Commodore 64, gives you some extremely sophisticated capabilities for word processing. First, this is a 100% machine language program, offering flexibility and speed that BASIC programs just can't match. It lets you use letter-quality printers, like the NEC Spinwriter and Diablo printers, if you need the extra quality they offer for manuscripts and business letters. It also allows use with tape for transporting documents.

For those of you who like to see on the screen what you'll see on the page, EasyScript lets you adjust the screen width to any size from 20 to 240 characters. When using widths larger than the 64's 40 columns, the program scrolls sideways like the carriage of a typewriter. Personally, I just like to use 40 columns, because you can see all the text you're working with.

I am very impressed with the advanced capabilities of this program. I'm used to working with WordPro on the CBM 8032, and EasyScript can do anything WordPro can do, and then some. For example, WordPro allows you to create heading lines at the top of each page, which is essential in manuscripts. EasyScript lets you adjust the margins of the headings independently of the page margins, something I've often wanted to do but couldn't with WordPro.

EasyScript makes it simple to work with the disk while using text, another feature that WordPro doesn't have. You can read the disk directory in EasyScript without disturbing your text.

Some of EasyScript's features include moving and storing any section of the text (any characters, not just whole lines), moving the cursor by lines and pages (slowly or quickly), filling in blocks of text from another text file (for personalized form letters), deleting any section of text, hunting for words (you can choose to automatically replace them with other words), decimal mode for lining up columns of numbers, horizontal and vertical tabs that can be stored with the text, and more. Printer-oriented commands let you use bold face, underlining, super- and sub-script, programmable characters, and other features for special printers.

EasyScript is truly a professional word processing system, one that allows for any possible need. It works with almost any printer that can be hooked up to the Commodore 64, and can create an astonishing assortment of document appearances. Compared to word processors on much more expensive computers, it stands up well, and is a good investment.

The Word Machine

comes on a diskette for use with the Commodore 64 computer and 1541 disk drive. It is a BASIC program, which limits its capabilities, at least when compared with EasyScript. For a low price, including the mailing list software called **The Name Machine,** it is a bargain.

The program begins by allowing you to change the colors of the screen border, and characters – helpful for fuzzy home TV sets. I prefer to work with black letters on a white page, like a typewriter, but you can adjust for your own idiosyncrasies.

It then asks you if you're working on tape, disk, or both. Since the program comes on a diskette, I can't imagine too many uses for storing on tape. It is helpful if you're mailing a copy of your text to someone, since cassettes tend to travel with less loss of data than diskettes.

You also type in today's date, which is automatically tacked onto the beginning of the text on disk or tape. This lets you keep track of the latest revisions.

Next, the program displays the main menu. This menu lets you begin typing text, recall text from storage, edit an existing document, print out your text, store your text, display the text in printed form, search for words and replace them with other words, and process form letters (using the Name Machine's mailing list).

Creation of a document is very painless. Even though the program is

continued on page 15

EDUCATION

Ontario Schools Create Phenomenal Educational Software Package

Imagine. Six hundred-forty six (that's 646) genuinely workable educational programs on 50 disks, neatly packaged in two large volumes, clearly labeled and accompanied by a catalogue that lists programs by subject category, grade level and quality. Programs developed by teachers working in the classroom. Programs created to really teach, not just show off some flashy programming techniques. All of them in the public domain.

This may sound like some misty-eyed educator's hopelessly optimistic daydream. But, thanks to unprecedented cooperation among Commodore companies, the Ontario Department of Education and the government of Ontario, about 100 kids helped make this "daydream" a reality last summer.

The kids, from thirty school districts in the province of Ontario, were selected by their teachers to spend the summer revamping a veritable ocean of public domain programs, mainly for the PET, that had been collected in the Ontario schools, the Toronto PET Users Group, several colleges, and Commodore companies in Europe and North America.

The young programmers, aged 16-18, were paid mainly through a student summer work grant from Ontario's provincial government, along with a grant from the Ontario Department of Education. In addition, Commodore's Canadian office hired ten programmers and a full-time supervisor to oversee the project. The students worked every day from 8:30 to 4:30 standardizing the raw programs, then sent them to Commodore in Toronto, where the programs received a final cleanup, and were also converted to run on the Commodore 64.

Then, late in August—the frantic final week before school began—ten students ran forty dual disk drives eight hours a day to copy about 800 disks for distribution to the school boards in September. Astonishingly, they made the deadline.

One of the ten Commodore programmers works on conversions.

You may suspect it wasn't quite as easy as it sounds. Gaining a concensus among thirty school boards in a province the size of Texas is a phenomenon in itself, whatever the issue may be. You're right. It wasn't easy, but the enthusiasm of the participants carried the project through to completion.

The project actually began in 1981, when several educators affiliated with three large school boards in Ontario received government funding to hire a number of students for four-week periods during the summer. The intention was to begin cataloguing the programs, but, as these things go sometimes, the group found they first had to lay down some ground rules before they could plunge into work on the programs themselves. So they spent that summer learning where the problems were and putting down a foundation for the next summer's effort.

Meanwhile, Commodore's Toronto office, without knowing this specific project was already underway in the schools, decided it was time to start stepping up development in educational software. It wasn't too long, however, before Commodore's people got wind of what the schools were doing and decided to combine their efforts.

With the cooperation of the school boards, the project was extended to include conversions for the Commodore 64. Some pre-released prototypes were allocated by Commodore for this part of the project. A central clearing house was set up in Toronto, the students were hired, and by June, 1982, the all-out effort was underway.

Everything wasn't a bed of roses, however. The cataloguing committee, for instance, set a July cutoff date for completed programs, because they needed that much time to convert and catalogue the 600-odd programs that had been submitted in order to have them finished by Sep-

tember. However, the programmers in the schools went on working right through August, so by the time the school boards met, they received a total of 1031 fully catalogued, converted programs. Of these, the Commodore programmers had had time to convert 646 to run on the 64. (These 646 are the programs now available in the public domain, worldwide.)

Or, take the week of frantic disk copying, when the students ran out of disks in mid-week. And then ran out of labels the next day. As Commodore's Frank Winter, who became involved in the project early in 1982, put it, "It's the logistics that kill you."

"Nevertheless, this is a project that could be carried out by anyone anywhere in the world," Winter went on. "It's a great example of how cooperation among government, school boards and private industry can get a project finished that no one member could have accomplished individually."

Not only that, but, as Winter pointed out, Commodore and the schools are ready to do it again next summer — for another 600 programs.

As a result of the students' and teachers' efforts, each of the thirty school boards involved in the project received their complete set of disks at

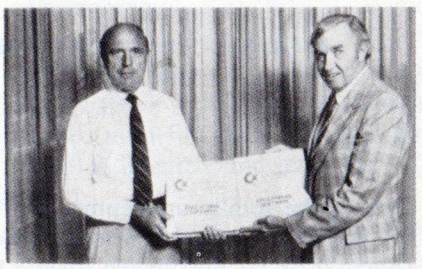
Finally, the disks are inserted into plastic jackets and assembled in binders.

a special presentation on September 13. According to Winter, more than one board member expressed surprise that the project had actually been completed.

"It's easy to start out on a project of this size," Winter commented, "but it's a lot harder to keep the ball rolling, especially in the summer, when many people go on holiday."

In addition to the disks received by the school boards, every authorized Commodore dealer in Canada also received a set of the 646 programs that had been converted to run on the 64. Each of these sets contains a total of 52 disks packaged in two volumes and a printed catalogue. Fifty disks contain the actual programs, and two contain demo programs, including character and sprite editors for the 64.

These 646 programs cover most subject categories, and also include administrative programs for grading, attendance and statistical analysis, a number of games and several utilities. They will run on any Commodore computer, including the 8032, PET (thin and fat 40) and Commodore 64, and will be available in the public domain all over the world.



Frank Winter (right) presents the first copies of the new public domain software to Ed Kellow, president of Commodore, Canada.

For those educators interested in exactly what subjects are covered in these programs, a brief breakdown: 24 business, 21 computer science, 108 English, 13 Francais, 10 history, 203 math, 101 science, 17 technology and 18 miscellaneous (includes things like "Man and Society", physical education and music).

Commodore's Public Domain Educational Policy

For some time Commodore companies around the world have been encouraging the development and exchange of public domain educational programs with schools. An example of this in the U.S. has been the Educational Marketing Resource Book, which contains 185 educational programs. The 646 programs referred to in this article will be released here in the U.S. and to the public domain throughout the world in conjunction with Commodore dealers, user clubs and Educational Resource Centers. Watch for details in future issues.

Ten students hired by Commodore ran 40 dual disk drives to copy 8,000 disks in one week.

TECHNICAL

PET Relative File Reader

by Elizabeth Deal

One of the totally undocumented features of the DOS 2+ is the syntax used in working relative files in Upgrade (Basic 2) PETs. Fortunately, over a year ago Jim Butterfield reported (perhaps discovered, too) the magic words in the DOS. And, fortunately, the relative files themselves are the high point of the Commodore disk books, even though the books are silent on the Upgrade syntax. Putting the two together is not difficult, and indeed, we can work the files with little more effort than BASIC 4.

Here is a list of several references on the subject:

1. Jim Butterfield, "Mixing and Matching Commodore Disk Systems," *COMPUTE* #9, Feb. 1981.
2. Jim Butterfield, "Relative File Mechanics," *COMPUTE* #11, Apr. 1981.
3. Raeto West, *Programming the PET/CBM,* Level Ltd and COMPUTE Books Publication.
4. Karl Hilden, "The Relative Record File System," reprinted from *The TRANSACTOR* in *Commodore Magazine,* July 1981.
5. Sieg Deleu, "Using CBM 8032 and DOS2.0," *The TRANSACTOR,* vol. 3, #3, also in *Commodore Magazine,* Dec. 1981.
6. Your disk book explains just about all there is to know about those files.

Relative files are a splendid invention. They can double as sequential files, are fast, convenient, never need a rewrite and, of course provide random access. You can write or read any part of a record without string concatenation. This means that garbage collection should rarely be a problem.

Relative files can be coded in Upgrade PETs so long as we use the DOS' native language, rather than words such as RECORD in our programs.

Since BASIC 4 disk commands are apparently translated to the simpler commands the disk understands I see no reason for the enclosed program not to work on 4-systems. I have not used the side-sector or super-side-sector feature in calculating maximum records, hence, this should also work with all DOS systems that support relative files.

SOME BASICS

To read a file: ST = 64 means the end of record. Error #50 tells us we overshot the file in the record number request of channel 15. We must NOT try to read, as the system will probably crash in the worst case, and return bad ST in the best. The presence of 255 in the first position tells us a record is empty, this being more important in writing than reading.

Usually files contain "normal" data in ASCII form (name, address type of thing), in which case the only trouble can be in proper positioning for reading and writing within a record. This is done with carriage returns. At other times, a file may contain coded data, put out by print #f,chr$(code), in which case we may run into reading problems, since it is difficult to echo some characters to the screen because they are control characters.

FILE READER

And this is one reason for the attached file reading routine. The ultimate file reader, in terms of accuracy, is, of course, a disk dump, such as Jim Butterfield's DISK VIEW or DISK MOD program. The latter permits effortless experimenting in destruction of files. In fact, little would get done without those two. Big thanks to Jim for writing and sharing the utilities.

I wrote this reader just for looking at what is in the file, (you cannot change the contents) and trying to put some

meaning into the values. In a way it is an interpretive program, but you will not see that part unless you do strange things or very wrong things. On normal, though perhaps, mismanaged records, you'll just see the normal-looking printout with several helpful indicators.

You may read the file in any order you wish. Numeric keypad keys are used to advance the record counter: 1 and 3 by plus/minus 1, 4 and 5 by 100, 7 and 9 by 1000. When a desired number is reached, push the RETURN key to read that record.

Alternately, to read sequentially, backwards or forwards push C key (continuous read) and hold on to the number key. To go back to random order push D (direct). Incidentally, in forward sequential reading we do not really need to do the RECORD-equivalent command in line 340, as the DOS takes care of positioning to the next record.

Every once in a while the counter gets stuck. Push any wrong key to fix that.

If you ask for a record with too high a number, PET will tell you about it. Eventually, if you hover around the area where you think the file ends, the PET will learn the number (line 710).

Q-key quits and closes the files. I recommend against using the STOP-key in any file handling programs.

Records get numbered, fields are on separate lines, all reading begins in byte one of the record. This organization is from PET's point of view. If the file is a mess, it may not reflect what you think is there or where it is, and this is also the reason for having an all-purpose reader. Please, bear in mind that the reader does not know about your logical file structure or any accessory index files.

CONVERSION OPTION

If you ask to see all characters, linefeed prints as reverse-j, carriage return prints as reverse-m and is acted on at the end of each field (chr$(13) and record (st = 64). Existing zeros in a record show up only if the end of record signal has not come in. Hence you may not see the trailing zeros. Squaring the record length with screen display will tell you what's happening. This is aided by a choice of making all spaces visible, as an apostrophe prints instead.

Other values may also show. They will be useful to people who in addition to writing normal ASCII files, put their information in coded numeric form by use of the CHR$(x) function. So long as x is a legal ASCII value, X will print normally. In the event of zero kids, thirteen bathrooms, and 147 sheep mowing the lawn, x needs to be handled differently, else the screen may clear on you in the best case.

Consequently, all such values print in the familiar PET manner: 147 comes out as reverse-shift-s, zero as reverse-@, etc. The advantage of the method is that the character count remains the same, but bear in mind that we are into interpretation. It is up to you to put the meaning into what you see. The point of the exercise is to make the record visible, and that it does. Once you know the record contains strange things, you can go after the exact contents with your own program.

MISCELLANEOUS NOTES and PROBLEMS

In DOS 2 and 2.5 relative files have side sectors (ss) containing 120 addresses. To arrive at the highest available record number (mx) take total blocks in file (bb) and record length (rl) from the directory and have the PET calculate records thusly:

(1) s1 = bb/120:s% = s1: ss = s% − (s1 < > s%), this tells us how many side sectors are in use. The rest are your records. So we compute:

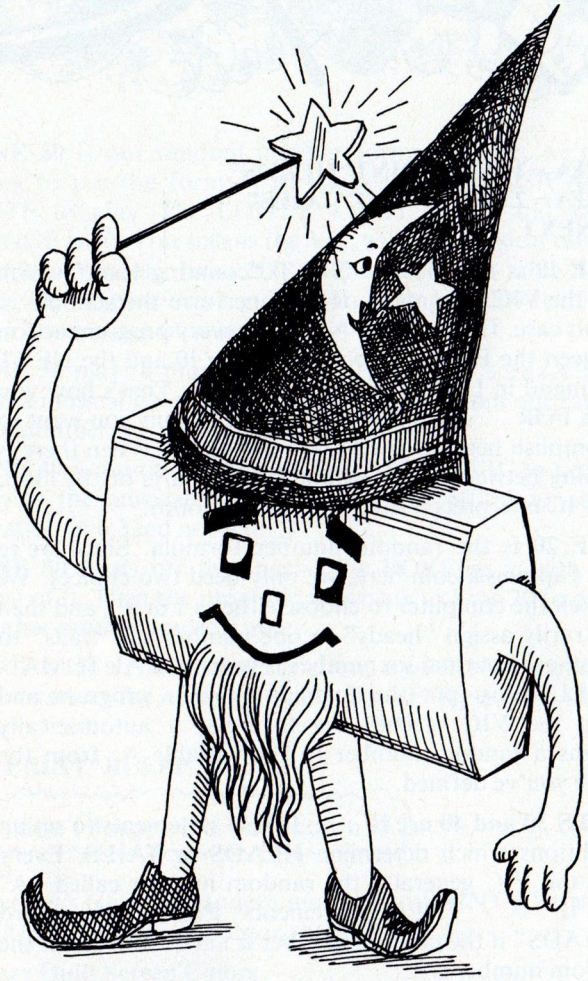
(2) m1 = 254*(bb − ss)/r1:m% = m1:mx = m% − (m1 < > m%) − 1, where mx is the magic answer. Record count may be larger than you think as complete blocks are assigned to the file.

The 4022 printer does not print five characters correctly, hence the need for reversal in line 490. A quote sent to the printer as chr$(98) does not set the quote mode, but the screen still needs this handled by a poke in line 400. This is important since we introduce reverse characters.

Apart from such conversions, this is a straightforward GET#1,I$ . . . PRINT#2,I$ type program though it may not look it on first sight.

The 4022 acts silly in some situations. I wanted to do two things: detect whether it is turned on (line 190) and to switch it to lower case mode (line 280). Following the instructions in the book I coded both lines one after another. This didn't get me anywhere. When the printer was originally off, then turned on, the desired switch to lower case mode did not take place until I separated the coding in both time and space. Can anyone explain this strange event? C:

```
100 REM-----------------------------------
110 REM RELATIVE FILE READER
120 REM ELIZABETH DEAL
130 REM-----------------------------------
140 FORJ=0TO9:READAV(J):NEXT
150 FORJ=2TO7:READOF(J):NEXT
160 DV=8:RN=1:PS=1:SA=2:SV=96+SA:QQ=256:NT=255:CR=13:Z$=CHR$(0)
170 PK=0:PJ=2:DR=1:MX=65535:P=4
180 INPUT"PRINTER    N■■■■";I$:PRINT:D=3:IFASC(I$)=89THEND=P
190 CLOSE3:OPEN3,D,10:PRINT#3:IFST=-128THENPRINT"■* TURN IT ON *":GOTO190
200 INPUT"DRIVE    0■■■■";DR$:IFDR$<>"0"ANDDR$<>"1"GOTO200
210 INPUT"RELATIVE FILE NAME    *■■■■";FL$:PRINT
220 CLOSE15:OPEN15,DV,15:PRINT#15,"I"DR$:GOSUB690:IFEGOTO200
230 INPUT"BEGIN AT RECORD#    1■■■■";I$
240 Q=INT(VAL(I$)):IFQ<1ORQ>MXGOTO230
250 INPUT"SHOW SPACES (')    Y■■■■";I$:F=39:IFASC(I$)=78THENF=0
260 INPUT"SHOW ALL CHARACTERS    N■■■■";I$:AL=0:IFASC(I$)=89THENAL=1
270 RN=Q:CLOSE5:OPEN5,DV,SA,DR$+":"+FL$+",REL":GOSUB690:IFEGOTO200
280 IFD=PTHENPOKE59468,12:CLOSE3:OPEN3,D,7:PRINT#3:REM LOWER CASE PRINTER
290 CLOSE3:POKE59468,14:OPEN3,D:PRINT
300 IFALTHENPRINT#3,"■ASC 0-31,128-159 --> REVERSE ASC+64"
310 PRINT#3
320 :
330 GOSUB560:IFEFGOTO440:REM QUITS
340 RH%=RN/QQ:PRINT#15,"P"CHR$(SV)CHR$(RN-QQ*RH%)CHR$(RH%)CHR$(PS)
350 GOSUB690:IFEGOTO440 :REM BAD
360 IFTBGOTO330 :REM TOO HIGH
370 IFD<>3THENPRINTRN
380 PRINT#3,"■*"RN
390 GET#5,I$:SS=ST:IV=ASC(I$+Z$):IF(64ORST)<>64GOTO440:REM IF POSSIBLE
400 GOSUB460:PRINT#3,I$;:IFIQTHENPOKE205,0
410 IFSS=0GOTO390:REM KEEP IN RECORD
420 IFD=3THENIFDRTHENPRINT#3
430 PRINT#3:GOTO330:REM RECORD
440 PRINT#3:CLOSE3:CLOSE5:CLOSE15:END
450 :
460 UV=IVAND127:IR=(IV=CR)OR(SS=64):IFFTHENIFUV=32THENI$=CHR$(F):REM BLANKS
470 IQ=(IVAND63)=34:IFIQTHENI$=CHR$(98-128*(D=P)):REM QUOTE
480 IFAL=0THENRETURN
490 IFD=PTHENIFUV>90ANDUV<96THENI$=CHR$(IV-128*SGN(IV-128)):REM [\]↑←
500 IFUV<32THENI$="■"+CHR$(IVOR64)+"■"+CHR$(-CR*IR):REM WEIRD & CR
510 RETURN
520 REM---USER INPUT REC#------------
530 DATA 81,13,49,52,55,51,54,57,68,67
540 REM  Q  CR  1  4  7  3  6  9  D  C
550 DATA -1,-100,-1000,1,100,1000
560 IFDRTHENPRINT"■                ■":PRINTRN:REM KEEPS COUNTER IN PLACE
570 KY=PEEK(151):IFKY=PKTHENIFPK>0THENJ=PJ:GOTO610:REM SAME OLD KEY
580 GETI$:AV=ASC(I$+Z$):REM NEW KEY
590 FORJ=0TO9:IFAV<>AV(J)THENNEXTJ:PK=-1:GOTO570
600 PJ=J:PK=KY :REMEMBER KEYS
610 IFJ=0THENEF=1:RETURN
620 IFJ=1THENPK=-1:GOTO670
630 IFJ<8GOTO650
640 DR=J-9:GOTO560 :REM TRUE ON D
650 RR=RN:RN=RN+OF(J):IFRN<1ORRN>MXTHENRN=RR
660 IFDRGOTO560
670 RETURN
680 REM---FLOPPY STATUS--------------
690 TB=0:INPUT#15,E,E$,ET,ES:IFE=0THENRETURN:REM OK
700 IFE<>50THENPRINT:PRINT"*"E;E$;ET;ES:RETURN:REM FATAL
710 TB=1:E=0:PRINT"■■■■■■■■■■■■■TOO HIGH":IFRN<=MXTHENMX=RN-1:REM EOF,NEW MAX
720 RETURN:----------------------------
```



The VIC Magician

Working With Random Numbers

by
Michael S. Tomczyk
Product Marketing Manager

Your VIC 20 has a built-in ability to generate RANDOM NUMBERS using a special command called RND. But what's a random number? The best way to show you how random numbers work is to give you an example that DOESN'T use a computer.

To see how random numbers work, try this: Take 10 pieces of paper and write a number from 1 to 10 on each piece. Next, put the 10 pieces of paper into a hat or other container where you can't see them. Now, cover your eyes and draw out one piece of paper. What number did you get? That number is a RANDOM number! Now put the number BACK IN THE HAT, mix up the papers and draw again.

Each time you draw a number, put the number back in the hat so there are always 10 pieces of paper to choose from. Keeping 10 numbers in the hat means you always have 10 RANDOM NUMBERS in the hat. When you take a number, you DON'T know which number is going to come next, but you DO know that the number will be between 1 and 10. This is the basis for RANDOM NUMBERS.

In programming, random numbers usually have a RANGE. This means there's an UPPER LIMIT and a LOWER LIMIT to the numbers you can draw. In our "hat" example, the range of numbers is 1 to 10. The lower limit is "1" and the upper limit is "10" . . . which means . . . ANY NUMBER FROM 1 TO 10 CAN COME UP "AT RANDOM" WHEN YOU DRAW.

Now let's see how a computer handles random numbers. Here's a program which generates 5 completely random numbers:

```
10 FORX = 1T05:PRINTRND(×):NEXT
```

This program tells the computer to pick 5 numbers "out of the air"—in other words, "random numbers"—and PRINT them. But these numbers all have decimal points (a number with "decimals" is 5.532 . . . a number without decimals is just 5 . . . the numbers to the RIGHT of the decimal point are called "decimals"). Most uses for random numbers require WHOLE NUMBERS without any decimal places.

You can make your random numbers always come out as whole numbers by using the INT function, which cuts off any decimal places. To see how INT works, type this: PRINTINT(5.236) and press the RETURN key. The computer will cut off the numbers on the right side of the decimal point and display only the "whole number" on the left side of the decimal point, which is 5.)

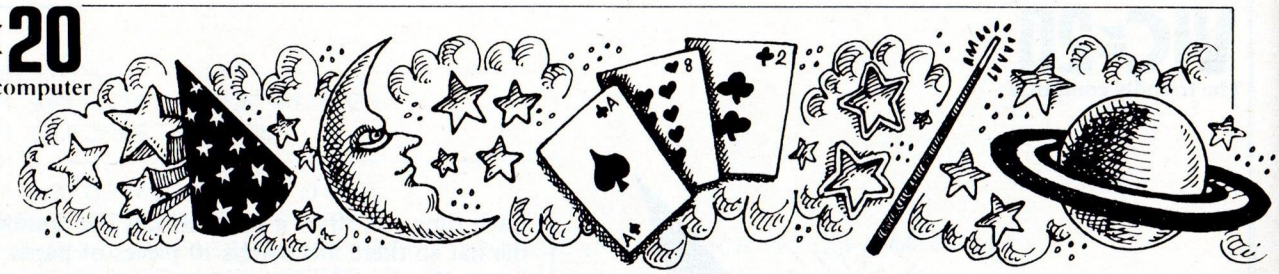
A Random Number Formula

Here's a "formula" which creates a RANDOM NUMBER within some RANGE of numbers that you define:

```
INT (RANGE * RND(1)) + LOWER LIMIT
```

The INT command tells the computer to cut off any decimal places and only give you whole numbers like 1, 45, or 320 instead of numbers like 1.223, 45.6677, or 320.59. Whole numbers are easier to work with when using random numbers.

LOWER LIMIT means the lowest number you want the computer to choose from, and **RANGE** means how many numbers are in the total group. For example, if you want to choose a random number from 1 to 5, the LOWER LIMIT will be 1 and the RANGE will be 5. If you want to choose a random number from 15 to 20, the LOWER LIMIT will be 15 and the upper limit will still be 5 because you are still choosing from a total group of 5 numbers. If you're choosing numbers from 2 to 100, the LOWER LIMIT will be 2 and the RANGE will be 98. See how it works?



PROGRAMMING TIP: To get a plus sign (+) just type the plus sign on the VIC keyboard . . . DON'T SHIFT THE PLUS SIGN. If you hold down the SHIFT key and type the PLUS key, you'll get a graphic symbol that LOOKS LIKE a plus sign, but really isn't. The plus sign is typed WITHOUT SHIFTING. The graphic criss-cross symbol is typed while holding down the shift key.

Let's use the formula to ask the VIC to choose one number at random between 1 and 5. This is a lot like shuffling 5 identical cards numbered 1 to 5 and laying them face down on a table. If you can only choose one card, which card will you get? The result will be random. Try this program (be sure to "balance" your parentheses by putting an equal number of left and right parentheses . . . there should be FOUR parentheses, two left and two right, as shown):

```
10 PRINTINT(5*RND(1))+1
```

RUN this program three or four times (keep typing RUN and pressing the RETURN key). The VIC20 keeps PRINTing a number from 1 to 5. The numbers are PRINTed randomly, in no special order. First you might get a 5, then a 3, but whatever you get, the number is ALWAYS one of these numbers: 1,2,3,4,5.

Multiple Random Numbers

Let's use a FOR . . . NEXT loop to tell the VIC20 to choose TEN random numbers from 1 to 100. We'll use the same formula, except now the RANGE is 100 and the LOWER LIMIT is 1. Type this line and RUN it:

```
10 FORX=1T010:PRINTINT(100*RND(1))+1:NEXT
```

The FOR . . . NEXT loop tells the VIC20 to display 10 numbers chosen at random from between 1 and 100. Yes, but what PRACTICAL use is this? Well, what if we chose a range from 0 to 15 . . . the numbers used to change the color of the screen?

Flipping a Coin

Here's another program that has the computer "flip" a coin 15 times . . . notice that in Line 20 we make the variable A equal to a random number represented by our entire formula which has 1 as the lower limit and 2 as the upper limit. In other words, the program "flips" between two numbers at random, just like a coin toss flips between 2 sides of a coin. The RANGE can be ANY TWO NUMBERS as long as they're consecutive (in order). For example, you could change 1 and 2 to 15 and 16, 200 and 201, etc. RUN this program several times to see how your imaginary "coin" randomly comes up heads or tails, with no real pattern:

```
10 FORN=1TO20
20 A=INT(2*RND(1))+1
30 IFA=1THENPRINT"HEADS"
40 IFA=2THENPRINT"TAILS"
50 NEXT
```

LINE 10 is a FOR . . . NEXT "counting loop" which tells the VIC how many times to perform the action . . . in this case, flip the coin. Note that every program action between the FOR . . . section in Line 10 and the NEXT command in Line 50 will be performed. That's how you use a FOR . . . NEXT loop. Put the actions you want to accomplish between the FOR and NEXT. (When there is nothing between the FOR and NEXT parts of the loop, the VIC interprets it as a "time delay" loop).

LINE 20 is the random number formula. Since we're only flipping a coin here, we only need two choices. We will tell the computer to choose either a 1 or a 2 and then arbitrarily assign "heads" to one number and "tails" to the other. The random number is generated AUTOMATICALLY. Just put the formula in your program and when the VIC reaches the formula, it automatically assigns a random number to the variable A, from the range you've defined.

LINES 30 and 40 use IF . . . THEN statements to set up conditions which determine HEADS or TAILS. Every time the VIC generates the random number called "A" the IF . . . THEN statements PRINT the word "HEADS" if the random number is 1 and "TAILS" if the random number is 2.

LINE 50 contains the NEXT command which matches the FOR . . . statement set up in Line 10. The result is that the ENTIRE PROGRAM occurs 20 times, including the random number being generated, and either a "HEADS" or "TAILS" message printed.

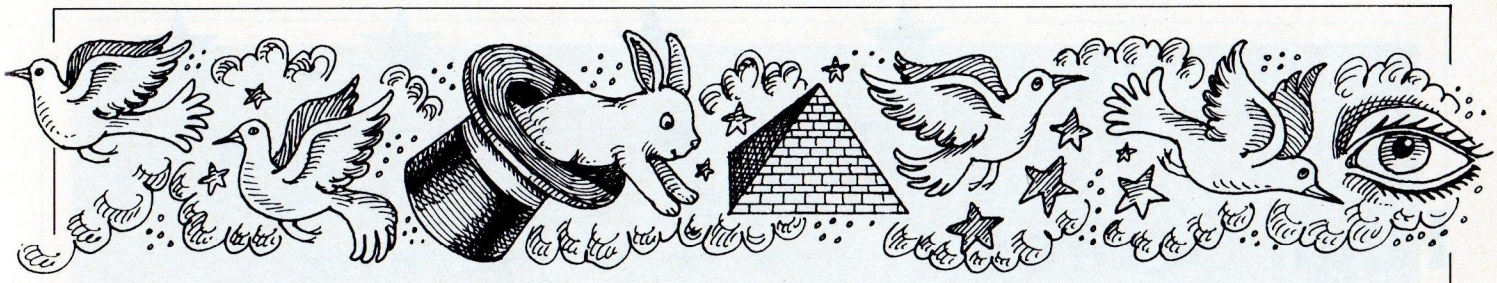
Random Music

Now let's use the VIC's musical capabilities to generate some "random music." Did you ever hear that strange computer music they sometimes use on TV shows about computers? You know . . . those "bleep . . . bloop . . . bleep" sounds? Well, here's some "bleep . . . bloop . . . bleep" music using random numbers:

```
10 V=36878:S1=36875:POKEV,15
20 N=INT(50*RND(1))+200
30 POKES1,N
40 FORT=1TO50:NEXT
50 POKES1,0:GOTO20
```

Hold down the RUN/STOP key and press the RESTORE key to stop the program.

LINE 10 defines V (for volume) and S1 for the speaker we want to use. Then we POKEV,15 to set the volume at its highest setting. See the SOUND AND MUSIC section of your user's manual for more information.



LINE 20 is our random number formula. Here, we're going to use the formula to choose which MUSICAL NOTE to play. The LOWER LIMIT is 200 and the RANGE is 50. This means the VIC will play musical note values at random from 200 to 250 (from the table on Page 135 of your user's guide). Try changing 200 to 150 for a lower range of notes.

LINE 30 plays a musical note by POKEing "speaker 1" with a musical note value (N) which comes from our random number formula in Line 20.

LINE 40 is a time delay loop which tells the VIC to keep playing the musical note for a count of 50 (about the duration of a 32nd note).

LINE 50 turns off the speaker by POKEing it with a value of 0. Then the program goes back to Line 20 to get another random musical note.

If you want an interesting variation, try PRINTing some characters as the notes are being played. Try adding this line:

35 PRINT"BLEEP. . .";

don't forget the semicolon (;)

Next, let's try using random numbers to PRINT a "crazy quilt" of colored symbols on the screen . . .

Crazy Quilt Screen Colors

The next example combines the RND command with the VIC's color capabilities . . . but first, let's write a program that DOESN'T use RND. This program fills the screen with the "#" symbol, in "crazy quilt" colors . . . but the colors appear IN ORDER, one at a time. Type the word NEW and press the RETURN key to erase your last program, then enter and RUN this program:

```
10 X=1
20 C$=" CTRL BLK CTRL WHT CTRL RED CTRL
   CYN CTRL PUR CTRL BLUE CTRL YEL"
30 PRINTMID$(C$,X,1)"#";
40 X=X+1
50 IFX=9THENGOTO10
60 GOTO30
```

Hold down the CTRL key and type the color keys shown

(Press RUN/STOP to STOP the program)

Like your crazy quilt? This crazy quilt isn't really crazy because it has a PATTERN, which means it isn't RANDOM. The program PRINTs a black symbol (0 = black) then a white symbol (1 = white) and so on up to 8 colors, then it REPEATS those colors in the same order, over and over again. Here's a quick explanation of how the program works:

LINE 10 defines the numeric variable X (it could be any letter) as the number 1.

LINE 20 defines the string variable C$ as the 8 "VIC color commands" which you typed in between the quotation marks (remember you can PRINT color and editing commands just like you PRINT letters, numbers and graphics. Notice that the VIC displays REVERSE GRAPHIC CHARACTERS on the screen when you hold down CTRL key and type a color key. This is a special VIC feature that shows you where color commands are in your program).

LINE 30 uses the MID$ function, a special BASIC command that lets the computer select any character from a "string" of characters in quotation marks, by COUNTING over the left side of the string. In our example, **PRINTMID$ (C$,X,1)** tells the VIC to PRINT a character from the "string" of color commands.

C$ identifies which string we are using (in case we have several strings in our program).

X means count X spaces over from the left and use that color command. At this point X equals 1 so the VIC takes the first character in the string, which is CTRL BLK, (the BLACK color command).

The number 1 in parentheses means do it one time.

The "#" symbol in Line 30 could be any symbol . . . try using a different graphic symbol, if you like.

LINE 40 changes X so that X is now 1+1, or 2. Now when the program goes back to Line 30, X will be 2 and the SECOND color command (WHITE) will be used.

LINE 50 only lets X go up to the number 8. If it reaches 9, the VIC resets X to 1 by going back to Line 10 (X = 1) and starting over. The effect is to PRINT the "#" mark in eight colors, then go back and do it over again.

Random Colors

Let's use the same program, but instead of displaying the "#" in the same 8 colors over and over, we'll display the symbol using RANDOM COLORS.In other words, the computer will mix up the colors without any special order, just like drawing the colors out of a hat.

Edit your program so it looks like this (if you're a beginning programmer, it might be easiest to type the word NEW, hit the RETURN key, then retype the program as shown):

```
10 X=INT(8*RND(1))+1
20 C$=" CTRL BLK CTRL WHT CTRL RED
```

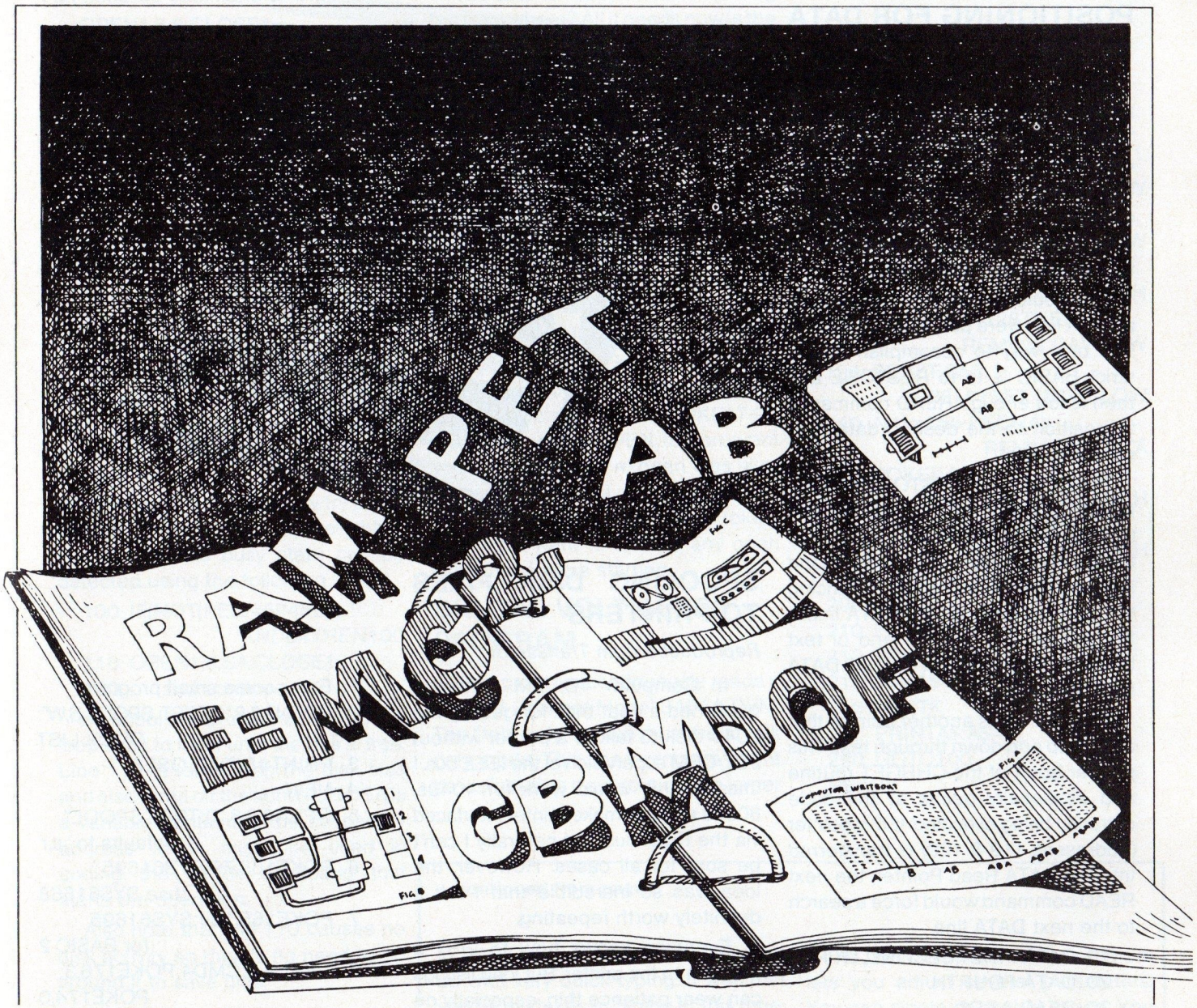
continued on page 48

High Scores

For serious game players who thrive on competition we'll be runnng the highest known scores for all Commodore games – cartridge and cassette. If you have a score that beats the existing record, send it in. But please remember you're on your honor, and phony scores will be on YOUR conscience, not ours.

VIC AVENGER
4,380
Debbie Hedin, Lincroft, NJ
JUPITER LANDER
101,600
Paul Dubrouillet, Holland PA
SUPER ALIEN
45,700
Robert Schaeffer, Brookline, MA
MIDNIGHT DRIVE
14 km
Amber Brandon, Santa Cruz, CA
RADAR RAT RACE
96,060
Candice Brey, Phoenix, AZ
SUPER SLOT
3,906
Amber Brandon, Santa Cruz, CA
MOLE ATTACK
296
Candice Brey, Phoenix, AZ
DRAW POKER
12,819
Angie Traina, Jonesboro, LA

CAR CHASE
34,965
Michael Dickenson, Nahant, MA
SLITHER
189
D.C. Murphy III, Ames, LA
OMEGA RACE
164,000
Brett Pickering, Five Dock, NSW
GORF
10,450
Garry Mason, Lane Cove, NSW
GALAXIAN
15,943
Roger Davis, Marsfield, NSW
ALIEN BLITZ
5,400
Keith Delamore, Karella, NSW



EXCERPTS FROM A TECHNICAL NOTEBOOK

Here is some additional information on accessing the serial port of the SuperPET which was not available at press time for the last issue. The RS-232 connector on the SuperPET is located on the middle circuit board near the right front corner. It is a standard 25-pin, female connector. The signals brought out to the connector are shown in the table below.

PIN NR.	SIGNAL DESCRIPTION	
1	Protective Ground	
2	Transmitted Data	(TxD)
3	Received Data	(RxD)
4	Request To Send	(RTS)
5	Clear To Send	(CTS)
6	Data Set Ready	(DSR)
7	Signal Ground	
8	Data Carrier Defect	(DCD)
20	Data Terminal Ready	(DTR)

SuperPET RS-232
Connector Pin-outs.

POSITIONING FOR DATA READS

Everyone knows how RESTORE, READ and DATA statements operate. The first READ gets the first DATA element, and so on. RESTORE sets the READ pointer back to the beginning of text. But there is no command that allows positioning to a particular DATA line.

This could be useful if, for example, a DATA line were part of a subroutine. The only way to accomplish this in strict BASIC is to to RESTORE and then issue enough READ commands to position to the desired data. This can be a pain!

RUN, CLEAR or RESTORE sets the DATA Read Pointer (address 62 and 63 decimal) back to $0400; the start of BASIC text. When a READ command is given, this pointer starts advancing through text looking for a 'DATA' line. If the pointer reaches the end of text before finding data, an ?OUT OF DATA ERROR occurs.

PET maintains another pointer that climbs up and down through text. This pointer is part of the CHRGET routine and essentially points at the code currently being executed. If this pointer (addresses 119 & 120) is transferred into the DATA Read Pointer, the next READ command would force a search to the next DATA line.

```
10 DATA FIRST,SECOND,THIRD
20 DATA FOURTH
30 READ A$,B$
40 POKE 62,PEEK(119):POKE63,
                    PEEK(120)
50 READ A,B
60 PRINT A$,B$,A,B
70 DATA 1,2,3,4
80 END
```

The READ command in line 30 gets "FIRST" into A$ and "SECOND" into B$, leaving the pointer pointing at "THIRD". Line 40 moves the pointer past line 10 and line 20 leaving it at some point in line 40. Since this is obviously not a DATA statement, the next READ causes an advance to line 70.

In summary, the POKEs of line 40 position to the next DATA statement in text.

SPOOLING DISK FILES TO PRINTERS

Reproduced from Transactor

In Computer #8, T.M. Peterson published a neat trick for getting the 2040 disk to talk to a printer without PET/CBM supervision of the IEEE bus. I imagine this would work for 4040s, 8050s and any make printer interfaced via the IEEE bus, but naturally I can't be sure for all cases. However, the idea was so incredible that I felt it definitely worth repeating.

Everyone knows how to LIST a program to the printer. But long listings can wear patience thin, especially on a slow printer! Not only thata, but while your printer is chugging along, the PET just sits there with everything disabled except RUN/STOP. Wouldn't it be nice if the disk fed the printer while you continue editing or play a quick round o space invaders or Microchess, that is if you can bear some of those arrogant printers. By the way, those wing nuts on the bottom of Commodore 202X printers..... take them out. They're only shipping screws that hold the mechansim tight. Once removed, the noise level is reduced considerably.

First a file must be created on disk. This could be ay SEQ file with the contents that are printer recognizable, but for now we'll create one of a program LISTing.

1. Enter some small program
2. OPEN8,8,8,"0:TEST SPOOL,S,W": CMD8:LIST
3. PRINT#8,"";:CLOSE8
4. NEW
5. OPEN8,8,8,"0:TEST SPOOL" ;defaults to ',s,r'
6. POKE165,72:SYS61695 ;use SYS61668
7. POKE165,104:SYS61695 for BASIC 2
8. OPEN4,4:CMD4:POKE176,3: POKE174,0

On hitting return the printer should fire up and continue at full speed to the end of the file. At this point your cursor might be ating funny (try hitting return on a blank line). To stop this, POKE14,0 for BASIC 2.0 POKE16,0 for BASIC 4.0. If you're lazy like me, invoking a couple of ?SYNTAX ERRORS (e.g. '=' and Return) will do the same thing. However, to restore normal cursor operation under program control (yes program control!), you would have to use the POKE. More on this in upcoming paragraphs.

Once the printer starts, don't try using the IEEE bus or the spool will abort. When it's all finished you can CLOSE the open disk file by sending

an Initialise command or with:

```
OPEN1,8,8:CLOSE1
```

A filename isn't necessary, but use the same secondary address as in step 5.

The NEW command at step 4 is only for clarity. Instead you might load another program or just leave the current one in for further editing. You can RUN the program in memory and even use the cassettes, but they're still as slow as before.

The example here uses all direct commands but they could just as easily be put in a program. Think of the applications! In a user oriented system, a report could be input to the disk and immediately spooled to the printer while the operator continues working on the next task. Of course the user might inadvertently try a bus operation which could kill everything. Fortunately this busy state can be detected using the following "trap":

```
100 IF(NOT(PEEK(59456)))
                AND64THEN100
110 OPEN1,8,SA:CLOSE1
130 ... and continue
```

If a spool is in progress, line 100 will loop back to itself until the bus is free. Line 110 is for closing the disk files and also turns off the active LED. SA is a variable containing the secondary address which might be used in coding the OPEN command that starts the operation.

Also note that line 110 causes no disk activity so there's no need to go around it to save time.

THEORY AND VARIATIONS

This example used device number 8 right through. However, you might have more than one disk on line which would mean a different device number. In step 6, address 165 (the IEEE output buffer) is POKEd with 72. This number (72) is derived from 64+8, where 8 is the device number. For versatility, this '8' might be replaced by a variable like DV.

The following SYS activates the ATN line on the bus telling all devices to 'pay attention'. The contents of 165 are then sent to the bus but only the device that has a matching 'talk' address responds, in this case device 8; the disk.

The disk is ready to start sending but from where? All it needs now is the secondary address of the OPENed file. Step 7 sets up the output buffer with the secondary address plus 96. Since 8 was chosen, the result is 104. This step might also be modified to read POKE165,96+SA.

Nothing happens yet because the ATN line isb't released by the PET. When CMD4 is executed (step 8), the printer becomes the output command device. PET releases ATN, the disk starts talking and the printer listens.

POKE176,3 tricks the PET into thinking the output command device is the screen and POKE174,0 simulates no files OPEN. In a program you would have to re-open files (e.g., command channel, modem, etc.) at spool completion. By the sme token, you might want to CLOSE any open write files before starting.

DIGITAL VOLTMETER PROGRAM

This program continuously reads a Hewlett Packard Model 3455A DVM. The "talk only" switch must be in talk only mode. This short program might be included as a subroutine in a larger data collection program.

```
 50 DIMA(14)
 90 A=59424
 91 B=59425
 95 F=59456
100 FORI=0TO14
105 POKEF,253:
        REM**SET NRFD LOW
107 POKEB,52:
        REM**SET NDAC LOW
130 POKEF,255:
        REM**SET NRFD HIGH
132 IFPEEK(F)>128THEN132:
        REM**IS DAV LOW
134 POKEF,253:
        REM**SET NRFD LOW
140 A(1)=255-PEEK(A):
        REM**GET DATA
160 POKEB,60:
        REM**SET NDAC HIGH
170 IFPEEK(F)<128THEN170:
        REM**IS DAV HIGH
180 IFA(I)=10THEN210
200 NEXTI
210 FORK=0TO1
212 PRINTA(K);
215 A$=A$+CHR$(A(K))
216 NEXTK
217 PRINTA$:A$=""
220 GOTO100
READY
```

continued from page 3

in BASIC, it has no problem keeping up with very quick typing. It won't let you fix mistakes in typing at this time; you must wait and go into edit mode later for this.

Once created, text should be stored – just in case. You wouldn't want to lose all your work due to a power failure.

Now you can go back and change your text using the Edit mode. This lets you select a spot in your text to change, or add more text in the middle. Here you must be careful to type slowly, for the program has quite a bit of work to do. The BASIC program won't keep up with any but the slowest typing rates when editing your text.

The printout section of the program lets you adjust several parameters. You can single or double space. You can set the number of characters per line, from 20 to 80 characters. The page size is adjustable, so customized forms can be utilized with this program.

The Word Machine is a simple word processor for the beginner who just wants to creat text and print it out on a printer like the 1525 or 1520. It requires patience to create, edit, and format a document to your liking – but on the other hand, its cost on disk, including The Name Machine, and is a good introduction for those with simple word processing needs.

Basic Program

The four function keys on the right hand side of the VIC are probably the most neglected part of the whole computer. Relegated to 'PRESS F1 TO START THE GAME', and dismissed with less than a page of text and a simple basic program by both the VIC REVEALED and the PROGRAMMERS REFERENCE GUIDE, the only way to use them as true definable function keys has been to spend £35 on one of Commodore's utility cartridges.

That is until now!

Using a simple 160 byte routine which sits at the top of BASIC memory, you can assign a separate function up to 8 characters long to each of the 8 keys.

This routine, which is loaded by a BASIC program, becomes part of the IRQ (Interrupt ReQuest) vector. This interrupt is the one which the processor calls 60 times a second to update the jiffy clock, scan the keyboard and check the RUN/STOP key.

Enter the program below and SAVE it before you RUN it! When the program has finished, it NEWs itself.

Now type RUN and press RETURN. All being well, the screen should clear and the messages "FUNCTION KEYS DEFINED" and "READY." should appear, along with the cursor. If you get the message "DATA ERROR", then you have made a mistake entering the data statements in lines 10-120. CHECK THEM CAREFULLY!

Assuming that you have entered the program correctly, pressing the function keys should give you the following functions:

F1	LIST + CHRS(13)
F2	GOSUB
F3	RUN + CHRS (13)
F4	PRINT
F5	GOTO
F6	CHRS)
F7	LOAD
F8	RETURN + CHRS(13)

If you wish to change any of these functions, simply alter the DATA statements in lines 300-310. To eiiminate the need to press carriage return, you can add one by simply entering' ' at the appropriate point(s).

For eaxmple: if you wish to LIST the program whenever a key is pressed, change the appropriate DATA statement to "LIST ".

The function keys can be cleared by pressing RUN/STOP and RESTORE together. To re-enable the keys, enter 'SYS 7520'.

1 — You can only have up to 8 characters maximum per key. To enter longer commands use the abbreviations (such as P SHIFT O for POKE) listed in the manual, or allot parts to individual keys.

2 — To change a function, the program must be reloaded.

Finally, to disable the RUN/STOP key while the function keys are in operation, change the last three numbers in line 120 to 194, 234, 170

USING THE FUNCTION KEYS

```
1 REM****************
2 REM*                *
3 REM* DEFINE  KEYS *
4 REM*                *
5 REM* BY DAVE TONG *
6 REM*                *
7 REM*  (C) 7/7/82  *
8 REM*                *
9 REM****************
10 DATA 120,169,128,141,20,3,169,29
20 DATA 141,21,3,88,133,56,169,96
30 DATA 133,55,96,160,64,169,0,153
40 DATA 191,29,136,208,250,96,234,234
50 DATA 72,138,72,152,72,165,197,197
60 DATA 251,240,44,133,251,41,39,201
70 DATA 39,208,36,24,165,251,42,41
80 DATA 240,172,141,2,240,3,24,105
90 DATA 8,105,128,133,252,169,29,133
100 DATA 253,160,0,177,252,153,119,2
110 DATA 200,192,8,208,246,132,198,104
120 DATA 168,104,170,104,76,191,234,170
199 REM      LOAD MACHINE CODE ROUTINES
200 POKE 55,96:POKE56,29:CLR:Z=0:FOR X=0 TO 95
210 READ Y:Z=Z+Y:POKE 7520+X,Y:NEXT X
220 IF Z<> 12270 THEN PRINT"DATA ERROR! RE-ENTER":STOP
230 SYS (7520):SYS (7539)
235 REM      SYS 7520 ACTIVATES THE KEYS
236 REM      SYS 7539 ERASES THE FUNCTIONS
240 FOR X=1 TO 8:READ N$
250 L=LEN(N$):IF L>8 THEN PRINTX;N$:PRINT"8 CHARACTERS MAXIMUM!":STOP
260 FOR Y=1 TO L:P=ASC(MID$(N$,Y,1)):IF P=95 THEN P=13
270 POKE 7607+Y+8*X,P:NEXT Y:NEXT X
280 PRINT"□FUNCTION KEYS DEFINED.":CLR:NEW
298 REM      PUT YOUR OWN FUNCTIONS HERE < MAXIMUM 8 CHARACTERS! >
299 REM      FOR CARRIAGE RETURN ENTER '←'
300 DATA "LIST←","GOSUB","RUN←","PRINT"
310 DATA "GOTO","CHR$(","LOAD","RETURN←"
```

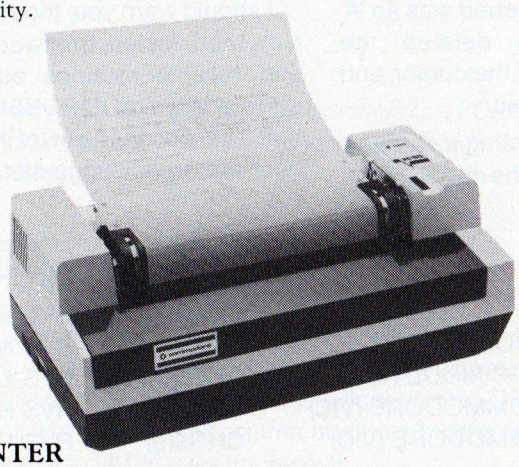
How to make the best home computer in the world even better.

Peripherals to turn a powerful computer into a super-computer for the professional.

With the Commodore VIC 20 Computer, you have the finest home computer money can buy. And the more you use it, the more you will ask it to do.

Pretty soon, you'll want to extend VIC's vast potential to the full; and there is a wide range of VIC peripherals to help you do it.

Disk drives, disk-based software, a printer, cassette unit, joysticks, paddles — with these, VIC computing becomes total computing: giving you true professional power and capability.



VIC PRINTER

The VIC Printer, like all VIC peripherals, offers a very high specification at a very competitive price.

It will print programs, letters, business data, graphic displays and so on.

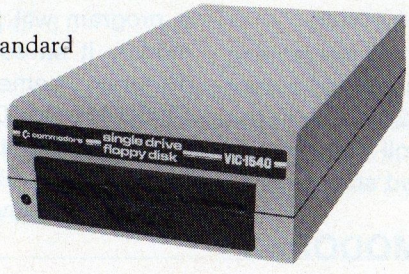
Its main features include: 80 characters per line • Tractor feed dot matrix • 30 characters per second print speed • Full alphanumerics and graphic printing • Double-size character capability.

VIC FLOPPY DISK UNIT

The VIC single-drive Disk Unit provides a fast, accurate and efficient means of storing and retrieving data and programs.

Together with the Printer, it transforms the VIC 20 into the ideal system for the small businessman or serious computer programmer.

Features include:
174,848 bytes capacity
• Uses soft-sectored standard 5¼" single density floppy disks
• Direct interface to VIC
• Direct compatibility with Printer
• Intelligent system independent of VIC.

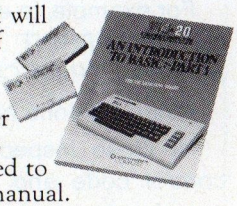


EXPANSION MEMORY CARTRIDGES

Special plug-in cartridges are available to expand VIC's memory. 3K, 8K and 16K RAM packs plug directly into the computer.

A Memory Expansion Board is also available to develop VIC's capabilities to the maximum.

1. Introduction to Basic



This package contains materials that will enable you to learn the fundamentals of programming in the BASIC computer language on your VIC 20 computer. It assumes no prior knowledge of computer programming and includes two cassettes containing 17 programs specially designed to accompany a comprehensive 152 page manual.

2. Progamming Aids

To aid programmers to write their own programs, Super Expander, Machine Code Monitor and Programmers Aid cartridges are available.

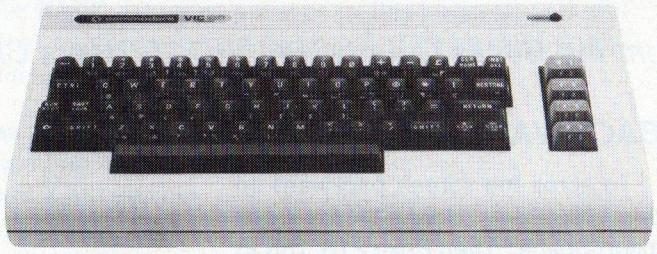
3. Games

The VIC 20 has the largest selection of cartridge and cassette-based games available on any personal computer. VIC 20 games offer a real challenge and take a great deal of skill, time and mental agility to win.

4. Joysticks and Paddles

These accessories make playing games on the VIC even more enjoyable. The joystick has even more practical uses and can be used with high resolution graphics to draw pictures or help with graph plotting.

For full details of VIC 20, its peripherals and software, and a list of your local dealers, contact:
The Commodore Information Centre,
P.O. Box 336, Artarmon, N.S.W. 2064
Tel: 437 6296



commodore

VIC 20

So much brain for so little. MLVL 1613

Control Codes

This applies to both VIC and PET machines.

During a program it may be necessary to print a 'non-printable' character. This is usually done by printing a 'control character' inside inverted commas. Quite a few of these control codes are available, for example, the Reverse Field Q to indicate "cursor down", RVS R for "print the following in reverse field" and RVS E, which on the VIC means "print the following in WHITE."

However, other codes require the form

PRINT CHR$(n)

For example, to execute a carriage RETURN, PRINT CHR$(13), or to perform a PRINT CHR$(20).

It can be done another way!

A close inspection of the RVS control code and the CHR$ value reveals some curious facts.

For instance, the value for CURSOR WHITE is CHR$(5). The code is RVS E. If you poke 5 to the screen, an 'E' appears. Coincidence?

CURSOR HOME <CHR$(19)> RVS S. Poke the screen with 19 and you get an 'S'. Hmmmmmmm.

Why don't we try a test. The DELETE character is CHR$(20). If we poke the screen with 20, a 'T' appears. Would a RVS T produce a DELETE?

PRINT "A (RVS T) B"

The (RVST) can be produced by typing the following (simple!) sequence:

[PRINT] ["] [A] ["] [DELETE] [RVS] [T] [RVS OFF] [B] ["] [RETURN]

It's not a little awkward, it's a lot awkward! It is necessary, though, to escape from the QUOTE MODE. We don't want a RVS R to appear when we press REVERSE. If you have a Programmers' Aid, you could CONTROL E to escape from QUOTE MODE.

Anyway, the result should have been a 'B' printed. (If not, are you sure you're using a Commodore Computer?)

What actually happened was an 'A' was printed, which deleted the character to the left of the cursor, and then the 'B' was printed.

If we want to try printing a carriage return <CHR$(13)> in the middle of our text, we should use RVS M. Now, before you go running off, I should warn you that it won't work. Everything on the line after RVS M is ignored.

Back to the drawing board? No, we'll just use a shifted return <CHR$>. I'm sure I couldn't tell the difference between printing a return and a shifted return.

An what's the code to use for a shifted return? A SHIFTED-M of course!

PRINT "HELLO [RVS SHIFTED-M] THERE"

HELLO
THERE

Bingo!

I should warn you, though, only use it on tried, tested, trustworthy, proven and reliable routines as it makes editing well nigh IMPOSSIBLE!!! Now that I've got that out of my system, here are some suggestions.

INSERT	CHR$(148)	RVS SHIFT-T
DELETE	CHR$(20)	RVS T
SHIFTED RETURN	CHR$(141)	RVS SHIFT-M
CHANGE OF TEXT MODE (VIC)	CHR$(14)	RVS N
CHANGE TO GRAPHIC MODE (VIC)	CHR$(142)	RVS SHIFT-N
LOCK-OUT SHIFT-COMMODORE (VIC)	CHR$(8)	RVS H
UNLOCK SHIFT-COMMODORE (VIC)	CHR$(9)	RVS I

BRAD FISHER

USER HINTS

From the Bucks County (PA) VIC 20 Users Club Newsletter

BACKWARD SCROLL

To scroll the screen backward on the VIC, Print "Home Down Back Insert Space." Then Poke 218, 158 to tell the VIC that the second row is not a continuation of the first row.

PRINTING IN THE BOTTOM RIGHT CORNER

To print in the bottom right corner of the screen a X, and to the left of the X a _____, for example, Print Tab (20)"X Back Insert _____(Up or Home)." The Up or Home is necessary so that the VIC will not scroll the screen and remove the X from the corner.

An alternate method for printing characters in the bottom right corner is to print the entire 23rd row in the 22nd row, and scroll the screen backward as described above.

KILLING COMMODORE/ SHIFT

To kill the Commodore/Shift key, Print Chr$(8). To lock the VIC onto upper/graphics or upper/lower Print Chr$(142) or Chr$(14) respectively. If a program will be in one of these modes, it saves memory to let N$ (Program name) = N$+Chr$(8)+ Chr$(142) or Chr$(14). Save N$. When the VIC tells you that it has Found N$, it will lock itself into the mode of your choice.

Bits and Pieces

Got some interesting bit of info you'd like to share?...a POKE, a screen dazzler, a bug, or some other anomaly? Send it in! We'll ge glad to print it! Ed.

Optical Illusion

This neat little machine language program was written by Dave Berezowski at Commodore Canada. It doesn't do very much except create a rather interesting looking screen. The program will work on 40 or 80 column machines but the 80 column seemed to be the most impressive.

```
033c    ldx   #$00
033e    inc   $8000, x    ;vic users must subst.
0341    inx                 screen address
0342    bne   $fa
0344    inc   $033f
0347    jmp   $e455       ;for BASIC 4.0 users
0347    jmp   $e62e       ;for BASIC 2.0 users
0347    jmp   $eabf       ;for VIC-20 users
```

As you can see, the routine is interrupt driven which means you'll need to POKE the interrupt vector to get it going.

```
poke 144, 60 : poke 145, 3
```

After servicing this code, the normal interrupt routines are executed which means you'll still see the cursor. You can even edit (and RUN) BASIC while this is running, just don't try to use the cassette buffer that it lives in or whammo! Try moving the cursor around the "affected area".

Notice that the program is self modifying, a practice that is OK for small programs but should be avoided like the plague in larger ones. Self-modifying software is the worst for debugging and finding out the hard way is not fun.

Vic users could also get this going without too much difficulty (maybe even with colour?). Just substitute the PET/CBM screen start address ($8000 in the second line) with the start address of the screen in your particular Vic, one of two possibilities, $1E00 normally or $1000 with some memory expansion units. To engage it. . .

```
poke 788, 60 : poke 789, 3
```

For BASIC 4.0 users, just type in this loader. Others will need to change just the last two DATA elements and the interrupt vector POKEs.

```
10  for j=828 to 841 : read x : poke j, x : next
20  data  162,   0, 254,   0, 128, 232, 208, 250
30  data  238,  63,   3,  76,  85, 228
```

One last note. . .don't try to include the interrupt vector POKEs in the above program. Chances are your machine will crash because before both POKEs get executed, an interrupt occurs somewhere in-between.

Selective Directory

Ever been searching through your diskettes for a program and found yourself sifting through SEQ and REL filenames that just seem to get in the way? Or how 'bout the opposite. . .when you're looking for an SEQ or REL filename that's lost in diskettes full of programs. Well..here's a quick way around it.

```
LOAD "$0:*=PRG", 8
```

When finished, LIST will display all PRG files from the directory. It would stand to reason that matching type filenames would appear for both directories if the drive number were omitted, but such is not the case. If you leave out the drive number the disk only returns filenames from the last drive used.

Mysteriously, DLOAD won't work the same way. You must use the LOAD command followed by ',8'. Any file type can be selected though. Merely substitute PRG for SEQ, REL or USR.

Another variation. . .subtitute the * for filename patterns. This has been discussed before, but now you can look for filenames that match a pattern and are also of a particular type. . .

```
LOAD "$1:B*=SEQ", 8
```

. . .would load a directory of all sequential files on drive 1 that start with 'B'.

A Most Welcome Error Message?

Never thought you'd see the day an error message would be pleasant, did you? Well today is the day! Just turn on your machine, hit HOME and RETURN. Too bad you can only get it when the machine is empty!

Quick File Reader

This three-liner will read just about any SEQ file. It's not very sophisticated but when you just want to "take a boo" at a file, it can be typed in quickly and isn't too hard to memorize. The RVS will help to spot any trailing spaces.

```
10 dopen#8, "some file"
20 input#8, a$ : ? "■" a$ : if st=64 then dclose : end
30 goto 20
```

For REL files, simply change the IF statement in line 20 to:

```
if st=64 and ds=50 then. . .
```

The Dreaded Illegal Quantity

Sometimes you want to read files one byte at a time. A routine much like the one above might be used, only the INPUT# would be replaced by a GET#. There's just one minor gotcha. It seems that when a byte value of zero is retrieved by GET#, the string variable slated to receive it is set to a null string, not CHR$(0).

The most common occurence of byte-by-byte reading is with PRG files from disk. Program files contain lots of these zeroes, at least one per line of BASIC (end-of-line markers). Usually a program to read the PRG file is set up like this:

```
10 open 8, 8, 8, "some prg file,p,r"
20 get#8, a$ : print a$, asc(a$) : if st=64 then close 8 : end
30 goto 20
```

The problem is that when a zero is read into A$, the ASC(function cannot cope will a null string and bombs out with ?ILLEGAL QUANTITY ERROR. The solution? You could add an extra IF statement after the GET#, for example:

```
if a$="" then a$ = chr$(0)
```

. . .but that would mean an extra line for the PRINT statement and the following IF. . .rather clumsy. Keep things tidy with:

```
print a$, asc (a$ + chr$(0))
```

The ASC(function returns the ASCII value of the first character of A$. If A$ starts with a valid character, then adding CHR$(0) will make no difference. If not, then CHR$(0) will be added to the null string and a "0" will be printed rather than the dreaded illegal quantity error.

The Mysterious Extra Records

Those of you familiar with the Relative Record system will know that the end of a relative file is flagged by the ?RECORD NOT PRESENT error, DS=50. However, the last record used for data is not necessarily the last record of the file.

As relative files get bigger, the DOS formats additional sectors by filling them with "empty records". An empty

record starts with a CHR$(255) followed by CHR$(0)'s to the end of the record which is determined by the record length. This formatting process occurs when data is written to a record that will require more disk space than has been allocated to the file so far.

Each 256 byte sector can contain 254 bytes of data (the other 2 are used by the DOS). Let's take an example record length of 127, thus 2 records fit exactly into 1 sector. Imagine that 2 complete records have already been written to the file. Upon writing a third record, the DOS must format another sector. Two empty records are written, but the first will be replaced by the data of our third record. Closing the file causes our third record and the one empty record to be stored on the diskette.

Re–opening the file is no problem, but how do we find the next available space for writing a new record? Athough our fourth record is empty, a RECORD#lf, 4 will NOT produce a ?RECORD NOT PRESENT error and the CHR$(255) could successfully be retrieved and mistaken for valid information. Therefore, we must test the first character of the record for CHR$(255). An INPUT# of this record will result in a string of length 1, so a combination of the two conditions might be appropriate. However, INPUT#ing live records of length greater than 80 will produce ?STRING TOO LONG error, so GET# must be used in combination with an ST test:

```
1000  rem *** find next available record ***
1010  record# (lf), (rn)          :rem rn = record number
1020  get#lf, a$                  :rem get 1st char
1030  if ds = 50 then return
1040  if a$ = chr$(255) and st = 64 then return
1050  rn = rn + 1 : goto 1010
```

This subroutine will search forward from wherever you set RN initially. It stops when either a ?RECORD NOT PRESENT occurs or when an empty record is found. For larger files, you might consider starting at the end of the file an work backwards, but you'll need to find the first live record and then move the record pointer one forward.

In summary, relying on RECORD NOT PRESENT is not good enough. Although it will insure an empty record every time, it will eventually leave you with wasted disk space. Often the first record of the file is used to store a "greatest record number used" variable which is updated on closing and read back on opening. Although this is probably the cleanest approach, it will only return new record numbers. Any records that have been deleted by writing a single CHR$(255) must be found with a subroutine like above. Possibly a combination of both these techniques will produce a more efficient filing system.

25 Columns of 30 Rows Screen Display for the VIC 20

by Colin Price

In trying to develop a software conversion to display 40 x 25 columns for the VIC 20, I have found that 750 characters instead of the standard 506 are easily displayed on the screen.

The following program is written for the standard VIC with no expansion passed $1FFF. Basically it lowers the limit of memory, sets up 750 locations for the video matrix above the memory limit, redimmensions and repositions the screen display, and changes the operating system and VIC chip screen pointers.

The screen may be now accessed in two ways:

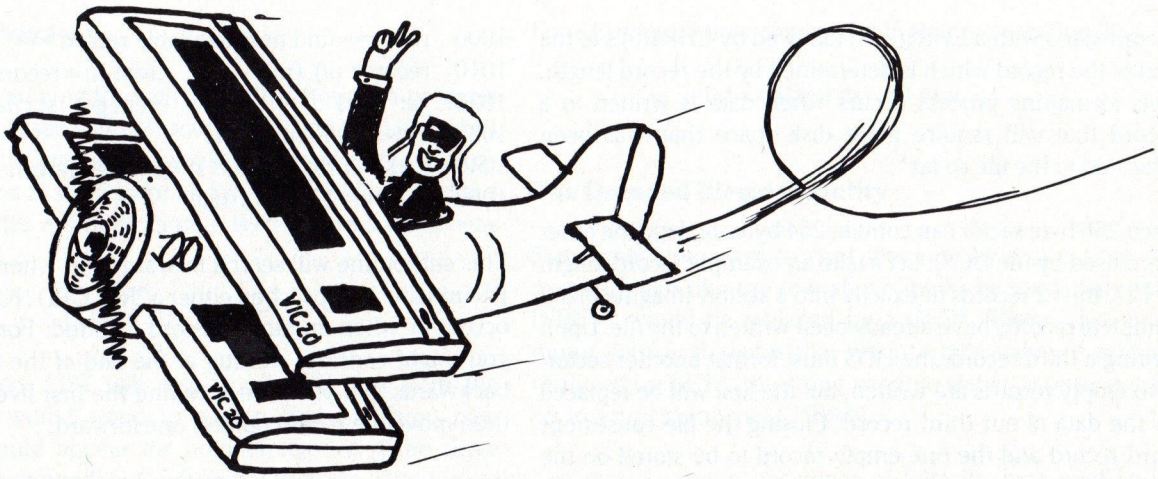
1) Normal print statements will access the first 506 locations the next 6 cannot be accessed. Poke 648,30 will access the last 238 screen locations. Poke 648,38 will now return you to the top section of the screen. (The middle section of the screen could be accessed by Poke 648,24 and normal print statements.)

2) Screen memory maps. The memory location of any position of the screen follows.

After running the program to set up the new display try this program.

```
1000 POKE RND(TI) * 750 + 7168, RND(TI) * 256 : POKE RND(TI) * 750 +37888, RND(TI) * 15 : GOTO 1000
```

continued on page 34



For Next Loops

The Vic 20 is probably the best computer on the market to learn about computers. It runs Commodore basic V2 which is more or less identical to the basic on all Commodore Business machines, whether they be for stock control or word processing.

I would like to introduce some of the more common Basic language commands to any newcomers to computing.

This issue I will start with For – Next Loops, they save a great deal of programming effort and once set up will run themselves as you will soon see.

For Next Loops are used on variables, which, using an apology, are like cheques – they can have an value you give them. As an example I can say FOR A=1 TO 5 which means, the variable A will start with a value of 1 and have a higher value of 5.

I simply need to say Next A. To show it actually working lets type in -

```
10 For A = 1 to 5
20 Print A
30 Next A
```

Now run the short program. Great! You say, so my Vic can count! What use is it??

Say you want the Vic 20 to draw a line from the left of the screen to the right, instead of 22 print statements try the following.

```
10 FOR A = 1 to 22
20 PRINT CHR$ (197);
          (Don't forget semicolon)
30 NEXT
```

Or, alternatively you want the computer to print out the full 12 times table! With two loops it is very easy!! Once a variable has reached its maximum (or minimum value – as you can decrease a value by using 'Step – 1' etc) it will 'drop' out of the loop to the next basic command!

```
10 FOR A = 1 to 12
                    :REM 1ST LOOP
20 FOR B = 1 to 12
                    :REM 2ND LOOP
30 PRINT A*B,
                  : PRINT A TIMES B
40 NEXT B
                  : INCREMENT B LOOP
50 NEXT A
                  : INCREMENT A LOOP
```

The result of this program will print out the full 12 times table.

For Next Loops can be stepped through by fractions or even by negative values, for example:-

For A = 1 to 5 step .2 will increment A by 1/5 each time.

Or For A = 5 to 1 step –1 will bring a from 5 to 1 in five loops – which would be similar to FOR A=50 TO 10 Step–10

To put a condition in your programs you may leave the loop using an if command between the For Next Loop.

```
10 For A = 1 to 50
:REM Loop A from 1 to 50
20 If A = 25 then 100
        :IF A = 25 GOTO LINE 100
30 NEXT A
                    :INCREMENT A
40 END
                  :PROGRAM END
100 PRINT "A" is now 25
                :REM PRINT A = 25
110 GOTO 40
     :REM GOTO PROGRAM END
```

Finally, you may use variables for the minimum and maximum values of the For Next variable eg:-

```
10 A = 10
                              :B = 30
20 FOR C = A TO B
30 PRINT C
40 NEXT C
```

JOIN THE CLUB

The VIC CENTRE COMPUTER CLUB:

- ★ Exchange cartridge service
- ★ Bonus discounts from the VIC CENTRE CATALOGUE
- ★ Club member specials

GET YOUR NAME IN FOR OUR 1983 CATALOGUE OF VIC-20 AND C64 HARDWARE AND SOFTWARE.

COMMODORE CARTRIDGES

(Choose one when joining the club)

Vic Avengers	Star Battle
Super Slot	Super Alien
Super Lander	Draw Poker
Road Race	The Sky is Falling
Mole Attack	Raid on Fort Knox
Sargon Chess	Pin Ball
Supersmash	Cosmic Cruncher
Gorf	Omega Race
Money Wars	Clowns
Seawolf	
ADVENTURES	
Mission Impossible	Pirate Cove
Adventure Land	The count

Reg. Retail $39.95

TRADE IN YOUR OLD MACHINE

on a Vic-20 and C64 microcomputers

AUTHORISED DEALER

commodore COMPUTER

From the VIC CENTRE in conjunction with CW Electronics
416 Logan Road, Stones Corner, Brisbane, 4120
P.O. Box 274 Sunnybank, Q'ld 4109, Australia
Tele.: (07) 397 0808 397 0888 Telex:AA40811 Brian Beamish
Authorised Commodore Dealer

MBG 83

C

THE CLUB

The VIC CENTRE COMPUTER CLUB APPLICATION

NAME: .....

ADDRESS: .....

.....

SUBURB: ..... P/CODE: .....

PHONE: .....

Membership Fee $55.00/$30.00

☐ Pay by Bankcard/
Bankcard No:

Expiry Date:

☐ Pay by Cheque/Money Order

Cartridge Title required:

Alternative

Signature:(Member and Card Holder) .....

NOTE: The signing of this form indicates agreement to the rules of membership.

Date: / /1983.

RECEIPT & RULES

CONDITIONS OF MEMBERSHIP:

(1) Membership fee of $55.00 (incl. first Commodore cartridge) Non-cartridge member $30.00, both renewable annually. $10.
(2) Reduced membership rates are available with additional cartridges purchased at the time of making membership application.
(3) Exchange of Cartridges allowed at the rate of $7.00 at time of exchange. (+ $1 certified postage if applicable).
(4) The variety and number of cartridges available for exchange will be controlled by The VIC Centre.
(5) All Cartridges returned for exchange must be working and undamaged.
(5A) A service charge will be incurred on all repairable cartridges.
(6) Packaging and instructional material must be returned with the cartridge and in good condition.
(7) The cartridge returned must be the same cartridge borrowed.
(8) MAXIMUM time exchange is 1 calendar month, after this time the cartridge is unreturnable and deemed the property of the member.

Date: / /1983.

SIGNATURES:(Client) .....

(The VIC Centre) .....

NOTE: The signing of this form indicates agreement to the rules of membership.

CLUB MEMBER BONUS

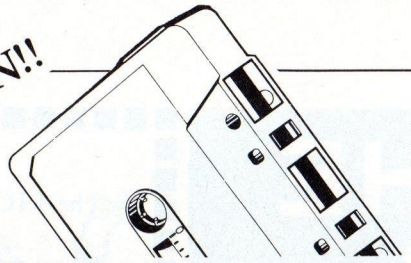
MEMBERSHIP DISCOUNT

For every one dollar spent with the Vic Centre you will receive a credit certificate for 12 cents against future purchases i.e.

Vic-20 $299 =	Credit Certificate of $35.88
Vic - Printer $ 479 =	Credit Certificate of $57.48
2 Vic Cartridges $79.90 =	Credit Certificate of $ 9.59
Commodore 64 =	Credit Certificate of $83.88

This does not apply to orders below $50, Club Specials, items purchased using Credit Certificates, or on trade-ins.

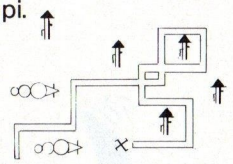
HEY LOOK AT THIS SELECTION!!



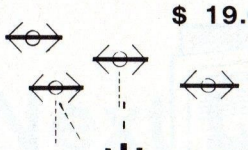
ROMIK SOFTWARE (UK)

MARTIAN RAIDER – Skim as close as you dare to the surface of the planet, devastating the Martian cities, destroying ammunition dumps (gaining more time), shooting down the ground to air missiles and U.F.O's, dodging or blasting meteorites. **$ 19.00**

SEA INVASION – Fight off the attacking sea creatures as long as you can. Shoot the whale for a surprise score. Watch out for the crabs, starfish and octupi. **$ 19.00**

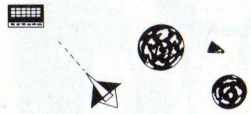


SHARK ATTACK – You are in shark infested waters – your only protection an atomic net which you trail behind you – protecting yourself and ensnaring sharks – careful of the octupi! **$ 19.00**



SPACE ATTACK – You are the pilot of an intergalactic battleship, fight your way through waves of alien fighters. **$ 19.00**

MIND TWISTERS – Four games to stretch your brain. **$ 19.00**



MOONS OF JUPITER – Send out your fighters from the mother ship to blast a way through the moons of Jupiter – look out for the U.F.O.'s and Grologs. *Requires 3, 8, 16 or Super Expander (3K)* **$ 19.00**

PIXEL (UK)

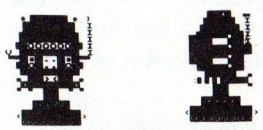
HARVESTER – unexpanded VIC – A cut throat strategy game to reap valuable Boosterspice around the planet Delta. HiRes graphics and lots of fun for two to four players. **$TBA**

BRAINSTORM – A battle of wits as you try to get three human explorers across a river of goo with three alien lifeforms that can literally blow your mind! **$TBA**

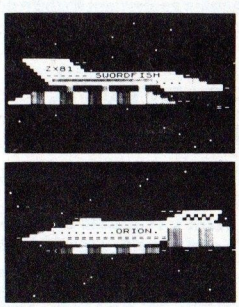
PIXEL POWER – To create user definable characters and save them with your own programmes. *8k expansion* **$TBA**

ENCOUNTER – Would you know what to do if you encountered extra-terrestial beings? In this exciting adventure , you are abducted by aliens and the space invaders play YOU! *16K expansion.* **$TBA**

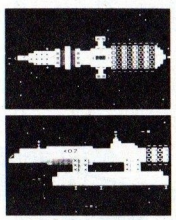
STARQUEST – A voyage of discovery and adventure in the cosmos. With the help of your onboard computer, you seek a habitable planet amidst the perils of deep space. *16K expansion.* **$TBA**



ZOR – Two mighty robots are designed for one purpose – to fight to the death. In the style of a mediaeval dual, you must do battle with the 'Champion of Zor' to save the planet Earth. *16K expansion.* **$TBA**



SUBSPACE STRIKER – It comes from out of nowhere and then vanishes back into the ether. With your deadly antimat torpedos, you unleash havoc in the Federation's spacelanes! *16K expansion.* **$TBA**



TRADER – Actual 3 x 16K parts to this program. It is hard enough to look at an amorphus hydrosilicon blob from PSI, never mind swing a deal with one, but when they ask to pick your brains. *16K expansion.* **$TBA**

MR MICRO LTD. (UK)

GREAT BALLOON RACE
Graphics Game for keyboard or joystick **$ 19.00**

MYSTERIOUS ISLAND
Recommended multi-tape adventure game for keyboard or joystick – needs *16K* **$ 19.00**

VIC VALUE NO 1
Keyboard or joystick – four games – Helicopter Lander, Dragon, Hunter, Alien Pilot **$ 19.00**

RAYFLECTION & MICROMIND
One game involves firing out rays to find the atoms, other is a variant of 'Mastermind'. **$ 19.00**

RABBIT SOFTWARE (UK)

RAID ON ISRAM (SCRAMBLE)
Raid on Isram is possibly the most challenging game ever introduced for the VIC-20. You must guide your craft through many perils to eventually get to your home base! **$ 17.00**

ORBIS
Defend your Uranium Fuel Dump from the invading "Zylons" by laying "space mines" in their path as they slip through the "energy field" but be careful as you only have a limited number of mines. **$ 17.00**

KRELL
Your mission is to defend the poor "Zymwatts" from the evil "Tharg" which sends "energy bolts" against their brick defences. You must destroy the "Tharg" before he kills all the "Zymwatts" which is not so easy as to get at the dreaded "Tharg" you must fend off his guardians. **$ 17.00**

UMI GAMES (USA)

SPIDERS OF MARS
Space battle. Protect your homeland against the Saturnian Bats and the Jovian Hornets. **$ 54.95 cart**

METEOR RUN
Conquer planet Aldebaran, fight the aliens on the way and negotiate the deadly meteor fields. **$ 54.95 cart**

AMOK
You are trapped in Amok, a deserted space station, try to find your way out but look out for the deadly robots. **$ 39.95 cart $ 16.00 cass**

CREATIVE SOFTWARE (USA)

coming soon

Serpentine	**$ 55.00 cart**
Astroblitz	**$ 55.00 cart**
Trashman	**$ 55.00 cart**
Terraguard	**$ 55.00 cart**
Videomania	**$ 55.00 cart**
Choplifter	**$ 55.00 cart**
Apple Panic	**$ 55.00 cart**

AUTOMATED SIMULATIONS (USA)

coming soon

Richochet	**$ 30.00 cass**
Datestones of Ryn	**$ 30.00 cass**
Invasion Orion	**$ 36.00 cass**
Sword at Fargoal	**$ 42.00 cass**
Rescue at Ryel	**$ 42.00 cass**
Crush, Crumble, Chomp	**$ 42.00 cass**
Monster Maze	**$ 55.00 cart**
Plattermania	**$ 55.00 cart**

HES SOFTWARE (USA)

Victrek	**$ 24.00**
Lazer Bltiz	**$ 24.00**
Tank Wars	**$ 24.00**
Concentration	**$ 24.00**
Dam Bomber	**$ 24.00**
Fuel Pirates	**$ 24.00**
Pak Bomber	**$ 24.00**

LLAMASOFT (UK)

ABDUCTOR
A classic new space game! ZAP the swirling alien hordes before they ram you and abduct your humanoids.
IN MACHINE LANGUAGE. REQ. JOYSTICK **$17.00**

THE VIC CENTRE

In conjunction with CW Electronics
416 Logan Road,
Stones Corner Brisbane
4120 P.O. Box 274
Sunnybank Q'ld 4109,
Australia Tele: (07) 397 0808
397 0888 Telex: AA40811

ABACUS SOFTWARE (USA)

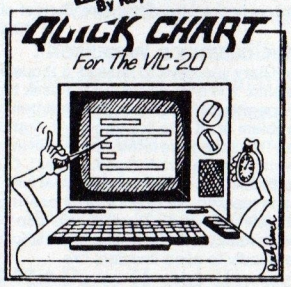
ALL ABACUS software is available on DISK or CASSETTE and most also include demonstration programmes.

SCREEN-GRAPHICS-64 – Super graphics for the COMMODORE-64 Here's the finest graphics software for your COMMODORE-64. SCREEN-GRAPHICS-64 gives you High Resolution, Multicolour and Sprite graphics all in one package! You get 320 x 200 points in High Resolution mode, 160 x 200 in Multicolour and Sprite Graphics in either HiRes or Multicolour modes. This is the most powerful package for the COMMODORE-64 yet developed. SCREEN-GRAPHICS-64 gives you two screens – one for normal text and the other for graphics displays. You can switch back and forth between the two screens in either command mode or from another program control mode. And you can mix both HiRes and Multicolour and Sprites on the same screen.

$28 cass
$31 disk

QUICK CHART for the VIC-20 and Commodore-64 – QUICK-CHART is a tool for creating bar charts on either your VIC-20 or COMMODORE-64 (*VIC-20 requires 8K expansion*). It is designed to give you presentation quality charts from the data that you enter at the keyboard. QUICK-CHART is menu-driven which makes it very easy to use. QUICK-CHART prompts you for data and then edits the data you enter against your minimum and maximum values. It does automatic scaling and if you wish to create monthly bar charts it does automatic prompting. Whats more is that you can save up to 10 charts onto either cassette or disk (3 charts with the VIC-20) so that you can later recall them for redisplay.

$ 20.00 cass
$ 23.00 disk



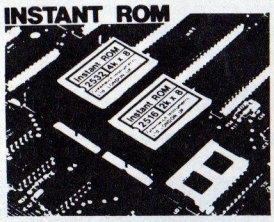
PIPER – The Music Machine for your VIC-20 or PET/CBM micro PIPER is an exciting way to turn your VIC or PET/CBM micro into a music machine. PIPER lets you compose, conduct and play your own musical scores with ease. And you'll be amazed at the speed with which you can enter your music using PIPER. PIPER works on the VIC-20 and PET/CBM micros, including the 8032.

$ 23.50 cass
$ 26.50 disk

HIRES/MULTICOLOUR GRAPHICS for the VIC-20 – The VIC has two additional graphic display modes available for anyone to use. Here's a package that gives you immediate access to these impressive VIC features. The HIRES/MULTICOLOUR GRAPHICS utilities make it simple and inexpensive to add these graphic display capabilities to your VIC. And best of all they require no additional memory. Without any additional hardware the high resolution utility provides a display area of 104 x 152 plot positions. You can draw and or erase points, lines, rectangles or write text onto the HiRes screen. The multicolour utility offers 52 x 76 plot positions in multiple colours.

$ 23.50 cass
$ 26.50 disk

GREENWICH INSTRUMENTS (UK)



INSTANT ROMS ETC.

GA-20 VIC 32K Development System..... **$119.00**
28K G-RAM for VIC..... **$139.00**
4K G-RAM for VIC **$105.00**
8K Instant ROM for VIC..... **$219.00**
4K Instant ROM for VIC..... **$145.00**
8K Instant ROM for VIC..... **$109.00**

ASK ABOUT ANY SPECIFIC PET/CBM SOFTWARE OR HARDWARE

ALL ITEMS ARE SUBJECT TO WITHDRAWAL OR PRICE AND SPECIFICATION CHANGES WITHOUT NOTICE

SPRITE-AID for the COMMODORE-64
HiRes or Multicolour graphic were never this easy!
$ 20.00 cass
$23.00 disk

SYNTHY-64
Music and Sound Synthesizer for COMMODORE-64.
$ 33.00 cass
36.00 disk

SKIER-64 for the Commodore 64
Test your skill on the slopes with SKIER-64, a fast moving game for your CBM-64. This arcade-quality game requires good hand-eye coordination for you to score high. And you can play without even getting cold in the snow.
$ 20.00 cass
$ 23.00 disk

BUDGETEER for the VIC-20, COMMODORE-64 or PET/CBM
Now you can use your VIC-20, CBM-64 or PET/CBM to plan and track your personal budget for the year with our visual planner BUDGETEER. Its features are surprisingly wide including bar chart and printout options
BUDGETEER requires a VIC with a *3K, 8K or 16K memory expander, a* PET/CBM *with at least 8K of memory or a* COMMODORE-64.
$ 23.50 cass
$ 26.50 disk

CRIBBAGE for the VIC-20 or COMMODORE-64
Another fine game to run on your VIC-20 or CBM-64. This is the same classic card game that has been played for many years. Your computer does the dealing and scoring – quickly and accurately.
$ 20.00 cass
$ 23.00 disk

TINY BASIC COMPILER for the VIC-20, CBM-64 or PET/CBM
Here's a useful and educational tool for users. The TINY BASIC COMPILER brings you the high-speed advantages of a compiled language at a very modest cost. If you want to gain the advantages of a compiled language, learn more about the workings of compilers or delve into the VIC-20, CBM-64 or PET/CBM's firmware, then try our TINY BASIC COMPILER. For the VIC-20 with a *3K, 8K or 16K memory expander; for* PET/CBM *with OLD, NEW or 4.0 ROMS or* 8032; for the COMMODORE-64.
$ 23.50 cass
$ 26.50 disk

GRAPHVICS super graphics for the VIC-20
You must have heard about the High Resolution and Multicolour graphics that are built into each and every VIC. HiRes gives you 152 x 160 points and Multicolour 76 x 80 right on your VIC's screen. Now you can utilise both with the most versatile graphics package yet available for the VIC. GRAPHVICS gives you 2 screens, one for normal text and one for graphic displays. The function keys allow you to switch back and forth between the two screens. On the GRAPHVICS screen you have control over 24,000 individual points! And you can mix both HiRes and Multicolour modes on the same screen to create spectacular graphic pictures on a $299 computer.
$ 28.50 cass
$ 31.50 disk

VIGIL for the VIC-20 or PET/CBM
Includes nine games. VIGIL stands for Video Interactive Game Interpretive Language. Vigil is a simple, new language for writing game and graphics programmes for your VIC or PET micro. Vigil brings you more than 60 powerful commands for manipulating graphics figures in double density on your VIC or PET screen. Complete with manual. *It requires a 3K, 8K or 16K memory expander on the VIC 20.*
$ 33.00 cass
36.00 disk

JOYSTICK PAINTER for the VIC-20
Here's one of the easiest ways yet to become an artist. VIC JOYSTICK PAINTER places a paintbrush in your hand. Use your joystick to draw pictures, graphs, figures.
$ 20.00 cass
$ 23.00 disk

I-CHING the Fortune Teller for the VIC-20
I-CHING is one of the oldest and most revered of fortune tellers. Now you can delve into your future with this colourful version of I-CHING based on ancient Oriental philosophy. The beauty of I-CHING is that you can go as far into it as feels comfortable. I-CHING comes as a complete package: the cassette program (or disk) with living colour and sound; the users introductory manual to get you started; and a 275+ page book to guide you through the hexagrams and Yin and Yang lines.
Requires either 8K or 16K expansion.
$ 28.50 cass
$ 31.50 disk

TINY PILOT
PILOT stands for Programmed Inquiry Learning or Teaching.
PILOT is an easy to learn language in which to program computer aided instruction lessons. Our version of TINY PILOT is so simple to use that you'll be writing your first program in minutes.
$ 22.00 cass
$ 25.00 disk

JOIN THE CLUB

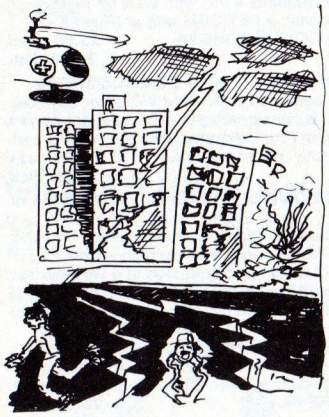
★ Exchange cartridge service
★ Bonus discounts from the VIC CENTRE CATALOGUE
★ Club member specials

AARDVARK **(USA)**

Adventure games for the VIC-20 – 16K expansion required Available soon for the Commodore-64

ADVENTURES are interactive fantasies. It's like reading an exciting book, except that you're one of the characters. You explore a new world as you try to think or fight your way out of a jam. You give the computer plain English commands such as "look in coffin" and "light the torch" and it carries out your bidding.
Each ADVENTURE normally takes from 15 to 30 hours to play, spread out over several days. If the FDA ever catches us, we are going to have to add a warning label. These are definitely addictive!!!
These ADVENTURES are in BASIC – but they are full featured, full plotted, fast action adventures. They come with listings and, as they are in BASIC, you can modify them yourself.
ADVENTURES ARE $20.00

EARTHQUAKE BY ANDERSON & RODGER OLSEN – A second kids adventure. You are trapped in a shopping centre during an earthquake. There is a way out, but you need help. To save yourself, you have to be a hero and save others first. Authors note to players – This one feels good. Not only is it designed for the younger set (see note in Haunted House), but it also plays nicely. Instead of killing, you have to save lives to win this one. The player must help others first if he/she is to survive. **$ 20.00**



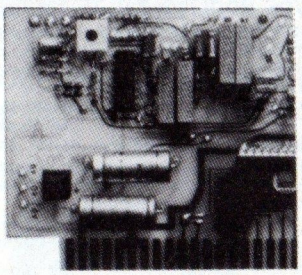
QUEST BY BOB RETELLE & RODGER OLSEN – This is different from all the other games of ADVENTURE!!! It is played on a computer generated map of Alesia. You lead a small band of adventurers on mission to conquer the Citadel of Moorlock. You have to build an army and then arm and feed them by combat, bargaining, exploration of ruins and temples, and outright banditry. The game takes 2 to 5 hours to play and is different each time. This is the most popular game we have ever published. **$ 20.00**

andiogenic **(UK)**

WORDCRAFT 20 – Wordprocessing cartridge –includes 16K ROM and 8K RAM, supports any parallel printer off user port or RS232, fully compatible with big Wordcraft. (Also includes 8K memory expansion for use with other programmes etc.) **$239.95**

WORCRAFT–64 available soon

computer world® **(HOLLAND)**



TDK-20 RTTY/MORSE Interface for VIC-20 – Send and receive RTTY/MORSE code, listen to Radio Amateurs and International News Radio Teletype (RTTY) with your VIC, TDK-20 and receiver. **$229.00**
Requires 1/12 volt centre taped transformer **$6.50**
Requires 1 user port plug **$ 6.50**
If required AFSK board **$ 52.00**

EPROM PROGRAMMER FOR VIC-20 – complete with software and manual. **$203.00**

VIC 64K EXPANSION BOARD – all one board, no mother board required. **$286.00**

80/40 COLUMN DISPLAY BOARD (black & white) Expand your system to 40 or 80 columns. **$260.00**

CURRAH COMPUTER SERVICES (UK)

CURRAH CHATTERBOX – make your VIC talk. Developed using the latest electronics technology this speech synthesis module allos an infinite vocabulary using the allophone speech technique. Includes free demo program. **$149.00**

CURRAH 16K MEMORY EXPANSION – the best buy around – *no club credit* **$ 99.95 apprx**

CURRAH MINI MOTHER BOARD – an economical solution **$ 54.00**

MORE ADVENTURES!!

PYRAMID BY RODGER OLSEN
This is one of our toughest Adventures. Average time through the Pyramid is 50 to 70 hours. The old boys who built this Pyramid did not mean for it to be ransacked by people like you.
Authors note to players – This is a very entertaining and very tough Adventure. I left clues everywhere but came up with some ingenious problems. This one has captivated people so much that I get calls daily from as far as New Zealand and France from bleary eyed people who are stuck in the Pyramid and desperate for more clues **$ 20.00**

MARS BY RODGER OLSEN
Your ship crashed on the Red planet and you have to get home. You will have to explore a Martain City, repair your ship, and deal with possibly hostile aliens to get home again. Authors note to players – This is highly recommended as a first adventure. It is in no way simple playing time normally runs from 30 to 50 hours – but it is constructed in a more "open" manner to let you try out adventuring and get used to the game before you hit the really tough problems. **$ 20.00**

CIRCLE WORLD BY BOB ANDERSON
The alien culture has built a huge world in the shape of a ring circling their sun. They left behind some strange creatures and a lot of advanced technology. Unfortunately, the world is headed for destruction and it is your job to save it before it plunges into the sun!! Editors note to players – In keeping with the large scale of Circle world, the author wrote a very large adventure. It has a lot of rooms and a lot of objects in them. It is a very convoluted, very complex adventure. One of our largest. **$ 20.00**

DERELICT BY R. OLSEN & B. ANDERSON
For Wealth and Glory you have to ransack a thousand year old space ship. You'll have to learn to speak their language and operate the machinery they left behind. The hardest problem of all is to live through it. Authors note to players – This adventure is the new winner in the "toughest Adventure at Aardvark Sweepstakes". Our most difficult problem in writing this adventure was to keep it logical and realistic. There are no irrational traps and sudden senseless deaths in Derelict. This ship was designed to be perfectly safe for it's builders. It just happens to be deadly to alien invaders like you. **$ 20.00**

VAMPIRE CASTLE BY MIKE BASSMAN
This is a contest between you and OLD DRAC – and it's getting a little dark outside!!! **$ 20.00**

HAUNTED HOUSE BY BOB ANDERSON
This one is for the kids. The house has ghosts, goblins, vampires and treasures – and problems designed for the 8 to 13 years old. This is a real adventure and does require some thinking and problem solving – but only for kids. Authors note to players – This one was fun to write. The vocabulary and characters were designed for younger players and lots of things happen when they give the computer commands. This one teaches logical thought, mapping skills, and creativity while keeping their interest. **$ 20.00**

INTERCEPTOR MICRO'S (UK)

FROG
An amazing version of Frogger in the unexpanded VIC-20. With driving turtles and plenty of vehicles to run you down. Fast action and high resolution graphics. **$17.00**

PUCKMAN – MACHINE CODE
The old favourite back again. Joystick or keyboard control. Fast action. High resolution colour graphics on the unexpanded VIC 20 **$17.00**

GALAXZIONS – MACHINE CODE
This is the most amazing game ever seen on the VIC-20. Galaxzions swarming in attack formation to destroy your planet. The nearest program to the real arcade game for the unexpanded VIC-20 **$17.00**

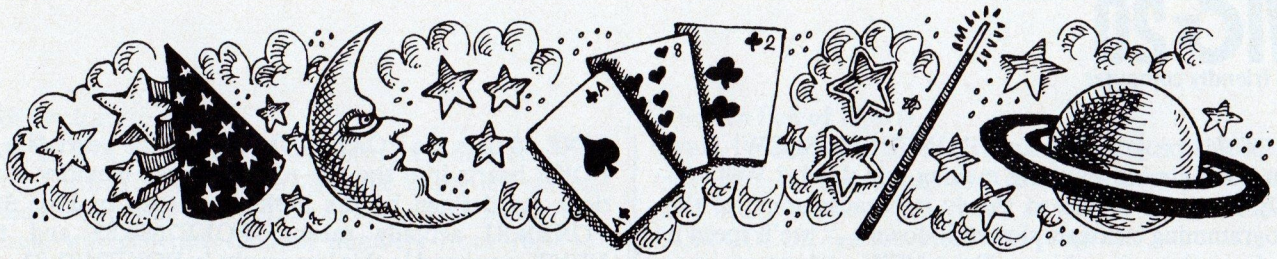
WORDHANGER
A highly educational hangman game with vocabulary and 2 player or play against the computer option **$17.00**

VIC BOMBER
An extremely fast action Bomber game for the VIC-20. With high resolution colour graphics. Flatten the enemy city before it's too late. **$17.00**

STACK (UK)

LIGHT PEN
plus 1 game additional games available **$ 70.00**
ANALOGUE JOYSTICK **$ 40.00**

THE VIC CENTRE
In conjunction with CW Electronics
416 Logan Road, Stones Corner Brisbane
4120 P.O. Box 274 Sunnybank Q'ld 4109, Australia
Tele: (07) 397 0808 397 0888
Telex: AA40811



Writing Games in BASIC

Part 1 . . . Animation & Random Numbers

by
Michael S. Tomczyk

Where do you learn how to write game programs? There aren't any schools you can go to. The few books on the subject are written in computerese (an obscure dialect understood only by engineers and wizards).

This 3-part series will try to fill a tiny part of the void of gamewriting information, by giving you some interesting and creative programming techniques you can use to write games.

In Part I, we'll use the VIC's random number feature to animate VIC graphic symbols so they move randomly, change color randomly or make random sounds.

In Parts II and III, we'll explore ways to move objects across the screen and design a simple game . . . then we'll design a game using your own PROGRAMMABLE CHARACTERS.

A Quick Animation Lesson

Here's a quick list of the steps we are going to follow . . . in order . . . in the animation programs we'll be explaining:

1. **Clear the screen at the beginning of the animation.**
2. **Set up a random number formula.**
3. **Display a symbol on the screen by POKEing a SCREEN MEMORY LOCATION with the POKE NUMBER of the symbol we're using.**
4. **Match the SCREEN MEMORY LOCATION of the symbol we're using with a COLOR MEMORY LOCATION. Every time you POKE a symbol into a different location, you have to POKE a matching color memory location.**
5. **Don't forget to keep ERASING the symbol as you move it. You do this by POKEing a blank space (POKE NUMBER 32) into the symbol's previous screen memory location.**
6. **Insert sound effects as appropriate.**

We'll begin by CLEARing the screen. Type this line and press RETURN:

10 PRINT" SHIFT CLR/HOME "

Now . . . turn to Page 144 in your VIC user's guide ("Personal Computing On the VIC20"). The SCREEN CHARACTER CODES and COLOR CODES MEMORY MAP represent all the locations on your television screen where you can place graphic symbols (including letters and numbers). Note that the TOP grid is used to position the symbol, and the BOTTOM grid is used to set the color at that position. So if you POKE a symbol into location 7680 (the top lefthand corner of your screen), you ALSO have to POKE a color setting into location 38400 which matches it.

For our example, we're going to use the heart on the "S" key on your keyboard. We begin by turning to Page 142 and finding the POKE NUMBER of the heart. From the chart, you can see that the number for the heart is 83.

Now let's find a SCREEN LOCATION where we want to put the heart. How about the top lefthand corner? The grid on Page 144 tells us the LOCATION is 7680. So our first line number includes the command POKE 7680,83. Try it:

20 POKE7680,83

Type RUN and press RETURN. Nothing happens! That's because you still haven't set the COLOR MEMORY LOCATION! Remember we said you have to set the matching color location EVERY TIME you POKE or MOVE a symbol on the screen? Type the word LIST and press RETURN.

Now look at Page 144. The COLOR MEMORY LOCATION which matches 7680 is 38400. Next you need to choose a color from the ones listed on Page 143. We'll choose green. The number of green is 5 (one less than the numbers on the keyboard color keys). So we have to POKE 38400,5 to make our heart turn green. Type this:

30 POKE38400,5

NOW type RUN and press RETURN! Presto, you have a solid green heart in the top lefthand corner of your screen!

For our first "random animation" example, we're going to POKE a row of green hearts at the top of the screen; but first we'll do it WITHOUT using random numbers. Type NEW to erase your previous program and enter this program:

```
10 PRINT" SHIFT CLR/HOME "
20 FORB=7680TO7701
30 FORBC=38400TO38421
40 POKEB,83:POKEBC,5
50 NEXTBC:NEXTB
```

VIC 20
The friendly computer

RUN this program. It runs PAINFULLY SLOW! How can you animate anything moving this slowly? Well, the only reason it moves so slowly is because some of the programming elements slowed it down . . . we'll speed it up by rearranging things. Type NEW and retype your program like this:

```
10 PRINT" SHIFT CLR/HOME "
20 FORBC=38400TO38421:POKEBC,5:NEXT
30 FORB=7680T07701
40 POKEB,83:NEXT
```

Now RUN it! A lot faster, right? That's because we took some shortcuts. The following comments will explain what we did:

LINE 10 clears the screen.

LINE 20 contains the secret of why the second program RUNs faster. The second version uses a FOR . . . NEXT loop to POKE the entire top row of COLOR memory locations BEFORE we poke the symbol into the screen memory locations. The previous program POKEd one color memory location, then one screen memory location, then repeated the process one location at a time, which was too slow.

LINE 30 sets up a FOR . . . NEXT loop for the SCREEN MEMORY locations of our heart.

LINE 40 uses the POKE NUMBER of the heart (83) to POKE hearts into the top row of screen memory locations (as defined by the FOR . . . setup in Line 30). POKEB,83 actually means: POKE7680,83 and the NEXT command in this line results in POKE7683,83 . . . POKE7682,83 . . . and so on until the heart has been POKEd into the top row of memory locations. The hearts appear fast here because all the color memory locations have already been set.

The Heartmaking Program

This program POKEs a row of hearts across the top of the screen . . . RANDOMLY! First it POKEs a red heart, then "counts to 800" and POKEs a green heart . . . and keeps going until the top row is filled with green hearts. After the row is filled, it keeps on POKEing hearts. This is your first "random animation." Type NEW and hit the RETURN key to erase your previous program and enter this program and RUN it:

```
10 PRINT" SHIFT CLR/HOME "
20 L=INT(21*RND(1))+7680
30 POKEL,83:POKEL+30720,2
40 FORT=1TO800:NEXT
50 POKEL+30720,5
60 GOTO20
```

LINE 10 clears the screen.

LINE 20 is a random number formula which sets the LOWER LIMIT at 7680 which is the upper left screen location. It sets the RANGE at 21 . . . which means the formula will generate random numbers from 7680 to 7701, which is the TOP ROW of screen locations.

LINE 30 puts a heart at the random screen location generated by Line 20, by POKEing that location with the value of the heart, which is 83. Then it colors the heart RED by POKEing the matching color location with the number 2 (red). There's an important programming tip here. We use L as the variable to define the constantly changing screen memory location, and we use L + 30720 as the matching color setting. You can match the color setting to ANY SCREEN LOCATION simply by adding 30720 to the screen location number. For example, if the first screen location is 7680, the color location is 7680 + 30720, which is 38400 . . . a quick check of the grid in your user's manual confirms this.

LINE 40 is a time delay loop which has the VIC "count" to 800 before changing the red heart to green (see Line 50).

LINE 50 changes the color of the heart we just POKEd to green. We're doing this because we want to show you the heart when it is first randomly POKEd onto the screen, and the easiest way to do this is to color it red, then change the color. Notice that the red hearts keep appearing even after the entire top row is filled with green hearts, because the program keeps going back to put more random hearts in the top row. Sometimes before the row is completely filled, a red heart appears on top of a green heart that's already there . . . again, because the hearts are being placed on the screen randomly, and can appear anywhere in the range of locations from 7680 to 7701.

You can add RANDOM SOUND to your heartmaking program by adding these 2 lines to your program:

```
35 POKE36878,15:POKE36876,INT(100*RND(1))
   +150
45 POKE36876,0
```

LINE 35 POKEs the volume setting (36878) to its highest level (15). Then it turns on speaker number 3 (36876) by POKEing that number with a random number whose LOWER LIMIT is 150 and whose range is 100. This musical setting causes the VIC to POKE music location 36876 with ONE musical note value from 150 to 250. Normally a time delay would be required here to hold the note for some DURATION, but by putting this line before the time delay in Line 40, we can use the delay in Line 40 as our note duration.

LINE 45 turns off the speaker after it has played the note. If you didn't turn off the speaker here it would keep playing.

The result of this program is to create a random musical note every time a new heart is POKEd onto the screen. This creates a random "bleeping" sound which is very "computerese" and makes a nice accompaniment to our random hearts.

A Screen Full of Hearts

This program puts hearts across the ENTIRE SCREEN (506 different locations) and uses the same principles used in the previous programs, as well as a new RANDOM COLOR feature. Here's the program:

```
10 PRINT" SHIFT CLR/HOME "
20 L=INT(506*RND(1))+7680
30 POKEL,83
35 C=INT(7*RND(1))+0
40 POKEL+30720,C
45 POKE36878,15:POKE36876,INT(50*RND(1))+150
50 FORT=1TO100:NEXT
55 POKE36876,0
60 GOTO20
```

Most of the elements are similar to those we used in the previous programs, so we'll just explain the parts that are different:

LINE 20 changes the locations from a range of 7680 to 7701 (the top row) to a range of 7680 to 8186 (the full screen, 506 locations). The LOWER LIMIT is 7680 and the RANGE is 506.

LINE 30 puts a heart at the random location determined by Line 20 by POKEing that location (L) with the heart POKE NUMBER (83).

LINE 35 sets up RANDOM COLOR by setting the LOWER LIMIT to 0 and the RANGE to 7 . . . these are the color settings you have to POKE to make your hearts turn color. We have defined the variable "C" as this random color setting, so C will always be a color setting number from 0 to 7 and will be randomly generated.

LINE 40 POKEs the color location with the random value of C.

LINE 45 generates a random musical note each time a heart is displayed. The LOWER LIMIT is 150 and the RANGE is 50 so the musical note values generated will be from 150 to 200.

LINE 50 has a faster time delay than the previous program.

LINE 55 turns off the musical "speaker."

LINE 60 goes back to Line 20 to get another random location where the next heart will be POKEd.

Summary

This brief introduction should give you some programming techniques to start experimenting with. Our next issue will contain more gamewriting techniques and . . . a real game! In the meantime, keep experimenting and see what you can come up with. The VIC's sound and graphics capabilities will surprise you when you start using them to write games.

Finally . . . a plug for Commodore . . . one of our best gamewriting tools is an inexpensive program on tape called the "Programmable Character Set/Gamegraphics Editor." The program comes with a detailed booklet that shows you how to create your own programmable characters and use them in your program. It's one of Commodore's best programming bargains . . . see your Commodore dealer for more information.

LOGICAL ALGEBRAS

In 1854 a book was published by a George Boole, the son of a Shoemaker, which set alight the mathematical world with curiosity. The book, called "An Investigation of The Laws of Thought on which are Founded The Mathematical Theories of Logic and Probability" sets out Boole's theories on algebra. The logic he developed is now known as simple Boolean algebra. His logic is now implemented in almost every digital computer in the world.

At the age of 20 he began his own school where he taught himself algebra and differential equations. He formed his theories based on the formula X * X = X to which the only numeric solutions in modern algebra are 0 and 1; this is the same in Boolean algebra. In computers this form of logic is ideal, as a 0 can be represented by no current flow and a 1 as a condition of current flow in the circuit. Then from this, other formulas are possible.

1 + 1 = 1 1 * 1 = 1
0 + 1 = 1 0 * 1 = 0
0 + 0 = 0

Basically, there are three types of operation that can be performed. These are AND, OR and NOT. These operators give programs the ability to make logic decisions. The way they work can be illustrated by two children in a sweet shop trying to decide which type of ice cream to buy. We will supervise the operation and supply the money to buy the sweets, but only under our terms. The terms will be dictated by the logical oeration we decide to use. This is if wewe remember there are only two types of sweets.

We can say that a type of sweet is selected if child A AND child B select the sweets.

Using the OR oeration we can say that a type of sweet is selected if either child A OR child B selects the sweets.

The NOT operator will always generate the opposite. If child A insists on disagreeing with child B, then child A's decision will always be that which is NOT which was selected by child B.

This can be represented using numbers, again only 0 or 1 can be used. The AND operation results in 1 only if both bits are 1:-

1 AND 1 = 1
0 AND 1 = 0
1 AND 0 = 0
0 AND 0 = 0

The OR operation will result in a 1 if either of the operands is a 1, or it can be said that a 0 result will only be produced if both operands are 0.

1 OR 1 = 1
0 OR 1 = 1
1 OR 0 = 1
0 OR 0 = 0

The NOT operation simply produces the opposite:-

NOT 1 = 0
NOT 0 = 1

Being the logic of 0 and 1's it can be directly compared to binary numbers which also only operate in 0 and 1's. Binary numbers consist of more than one digit, usually 8 digits. So numbers such as 10 in base 10 can be represented as 00001010 in base 2 (binary). For numbers like this the same logic can be applied to each bit. So the three operations can be illustrated as follows:-

```
43 or 137 = 9
  or 10001001  137
     00101011   43
  ------------  ---
     00001001    9

43 or 137 = 171
  or 10001001  137
     10101011   43
  ------------  ---
     00101011  171
NOT 43 = 212
     00101011   43
  ------------  ---
     11010100  212
```

These can be done very simply on a Commodore Computer just by using the BASIC commands AND, OR and NOT.

```
PRINT 43 AND 137
PRINT 43 OR 137
PRINT NOT 43 - 256
```

In the last example we needed to subtract 256 from the result, this becomes necessary because of the way in which Commodore BASIC stores numbers. Normally a NOT would give the result -44 which is the two's complement of 43 which is in fact 212.

These general ideas can be applied to branching and variable assignments.

An expression like the following can be broken down to give it's true meaning in terms of Boolean operations. Consider this IF/THEN instruction:-

```
IF A = B AND C = 0 THEN 50
```

Using the general rule that applies to these types of expressions – which is that all the expressions can be in a 0 or 1 state (either condition false or true). If we assume that the first expression is true and the second one is false. So in effect the above expression can be evaluated in terms of Boolean expressions. The above can then be changed to represent the follwoing:-

```
IF -1 AND 0 THEN 50
```

As a contrast to this type of expression we can use the ability of the IF/THEN statement only to branch on a non-zero answer. This leads on from the above which can be compressed again. The result of the AND operation above yields a 0 and so the above could be read as:-

```
IF 0 THEN 50
```

This then implies according to Commodore BASIC the following:-

```
IF 0<>0 THEN 50
```

Thus the branch is not taken as 0 = 0 and therefore do not meet the condition. This can be put to good use,

while also saving valuable memory space. The only disadvantage is that the program becomes hard to undestand. Which then requires the need for a REMARK statement to explain the action of the line; this then will take up more memory space. For example, the following two lines mean the same:-

```
IF 128 <> 0 THEN 100
IF 128 THEN 100
```

Compare these two variable allocation statements:-

```
20 LET B = A < 10
30 LET C = A > 10
```

These two examples are not errors, but a form the same type of logic. The first example will be evaluated to produce a -1 if A is less than 10 and a 0 if A is equal to 10 or greater. The second is evaluated as 0 if A is equal to 10 or less an -1 if A is greater than or equal to 11.

BASIC 4.0 String Bug

This error only occurs under BASIC 4.0 when there are less than 768 bytes free (or 3 times the largest string size), after all variables and arrays have been assigned by a program. The error is that BASIC fails to detect an 'Out of Memory' condition soon enough, causing corruption of string data and sometimes program text. An example of this bug on a 32k system follows:

```
10 DIM A(6330)
20 BUG$ = BUG$ + "W" + "x" :PRINT BUG$ :GOTO20
```

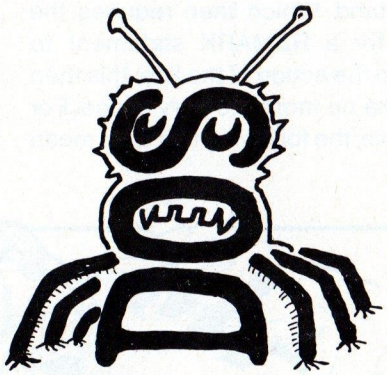
This program will build a string of alternating characters "WxWxWxWx". It will terminate correctly with an 'Out of Memory in 20' error, but the string will be corrupted after only a few passes.

The easiest solution to the problem is to trap the error from BASIC before it occurs:

```
IF FRE(0) < 768 THEN PRINT "Out of Memory'
:STOP
```

DOS BUG IN RELATIVE RECORDS

A bug has been uncovered in the Relative Records scheme that is used in all Commodore Disk Systems, 2031, 4040, 8050, 8250, 9060, 9090, and 8250. The bug of itself will not corrupt any data files, nor will it cause errors in the disk system, however, if a particular program sequence (as described later) is being used to update files, the file will then be corrupted.



This bug has been in existence since the introduction of Relative Record capability in Commodore Disk Systems without prior discovery, so the odds of it causing or having caused problems in programs is quite small.

PROBLEM DISCRIPTION

Associated with Relative Files is a parameter defining the length of the records in the file. With the exception of record lengths 1, 2, 127 and 254, the records in the file will span or overlap the boundary between sectors on the disk.

In any group or series of even length records (10, 40, 66, etc) that begins on a sector boundary (record #1, #128, #255, #509, etc), the record that spans the 2nd and 3rd records in the series will be in error during reading when all of the following three conditions are true:

The total data field in the first record of the series (record 1, 128, 255, etc) is shorter in length than the overlapped data (or spill) portion of the record which starts in the second sector. (e.g. for a record length of 100, record 6 will span sectors 2 and 3. The data field of record 6 will have all bytes in excess of byte 9 stored in sector 3. Thus, this condition will be met with a record length of 100 and a data field in record 6 that is 10 or more bytes longer than the data field of record one.

1) The program sequence that is running is a general update of records, where a record is read in, modified or examined and then written back out to the file. The following sequence will satisfy this condition:

```
1100 Record #1,(I)
  : REM POINT TO ITH RECORD
1200 INPUT #1,A$
          : REM READ DATA IN
1300 (MODIFY OR EXAMINE A$)
1400 RECORD #1,(I)
: REM POINT TO RECORD AGAIN
1500 PRINT #1,A$
          : REM WRITE NEW (OR
            UNCHANGED) DATA
```

2) The records are being updated sequentially, beginning with a record that starts in the first sector

eg: In the above example, adding lines:

```
1000 for I=2 to 10
1600 NEXT
```

Will satisfy this condition.

Under the above conditions, the data sent to the computer for record 6 will be truncated so that the data overlap of record 6 into sector 3 will exceed the length of the data in record one. The following program will demonstrate the bug:

```
100 A$ = "1234"
200 B$ = "123456789ABCDE"
300 DOPEN#1, "TEST", L100
400 PRINT#1, A$
          : IF DS THEN STOP
500 FOR I = 2 TO 10
600 PRINT#1, B$
          :IF DS THEN STOP
700 NEXT: DCLOSE#1
800 DOPEN#1, "TEST"
      :B$="123456789ABCDE"
900 FOR I=2 TO 10
1000 RECORD#1, (I)
          : IF DS THEN STOP
1100 INPUT#1, C$
          : IF DS THEN STOP
1200 IF LEN(C$) {} LEN(B$)
THEN PRINT "REC #"I; B$;
 LEN(B$);C$;LEN(C$):GOTO 1500
1300 RECORD#1, (I)
          : IF DS THEN STOP
1400 PRINT#1, B$
          : IF DS THEN STOP
1500 NEXT
1600 DCLOSE#1
```

Running the above will product a printout of:

```
REC #6 123456789ABCDE 14
       123456789ABCD 13
```

The data in record 6 on the disk is not corrupted, as can be shown by running the program again, starting from line 800. The error is occuring in the DOS RAM because an end of data pointer is not beng set correctly. However, if a blind read, update, rewrite is being followed then the truncated data from record 6 will be operated on and rewritten, resulting in corruption of the file.

Note also that the truncation is not detectable by error monitoring as no read error occurs.

PROBLEM RESOLUTION

Since the bug can occur with the general conditions as described, it is not possible to detect any unique conditions, however there does exist a simple programming technique to bypass the bug. Adding to the example program the line:

```
1450 RECORD#1, (I)
```

Will cause the DOS to reset the pointer for the sensitive record, eliminating the problem.

The addition of this statement immediately after the print statement will not materially affect the run time of a program as the DOS will not have to reaccess the sector.

Using DOS with the Waterloo microSystems

1. Introduction

DOS (Disk Operating System) is the operating system of the Commodore disk drive. Commodore disk units contain microprocessors which are programmed to support a number of commands for managing the disk. The commands are used to prepare a diskette for use (formatting a diskette) and to maintain the files on the diskette (erase, rename, and copy files). Most of the standard DOS functions for file manipulation are available as commands in the Waterloo microEditor, but diskette commands such as formatting are not.

With the 6502 microprocessor using Commodore BASIC 4.0, a DOS function was accomplished by issuing a special BASIC command or by opening the command channel (channel 15) as a file and writing the DOS command to that file.

With the 6809 microprocessor the Waterloo microSystems, DOS commands can also be issued using the command channel. The file "disk" is equivalent to the command channel. Therefore, by opening the file "disk" in a program and writing to that file, the disk drive will execute the DOS command. The Waterloo microEditor can also be used to issue DOS commands. While in the Editor, a file can be created that contains the DOS commands; this file can then be "put" to the file "disk".

Example:

Before files can be written on a new diskette, the diskette must be formatted in the soft sector format recognised by the disk drive being used. Formatting a diskette in 6502 is accomplished by using the DOS "New" command or the Commodore BASIC 4.0 "HEADER" command.

(1) OPEN #1,8,15
PRINT #1,Ndr:dname.xx

(2) HEADER "dname",Ddr,Ixx.

where:

dr is replaced by the drive number

dname is replaced by the name to be given to the diskette

** is replaced by a disk identifier.

Using the Waterloo microSystems, the diskette can be formatted using one of the following two methods:

(1) OPEN #1,"DISK",OUTPUT
PRINT #1,"Ndr:dname,xx"
Although this example is written in Waterloo microBASIC, any of the other Waterloo microSystems (i.e. APL, COBOL, FORTRAN, PASCAL, or 6809 Assembler) could be used.

(2) Create a 1-line file in the Waterloo microEditor containing:
Ndr:dname,xx
and issue the command:
put disk

Again "dr", "dname", and "xx" are replaced by the appropriate values.

2. DOS Commands

This section describes some of the Commodore disk commands and how they may be used. All examples are illustrated using Waterloo microBASIC statements. Note that the operation portion of the disk command must be specified in upper case. The Commodore disk manual should be used to supplement the descriptions that follow. Note that several of the operations described below are available as commands or statements in the Waterloo microLanguages.

2.1 Diskette Preparation (Formatting)
print #2,'NEWdrive:diskname,id'

where

"NEW" or "N" specifies the operation to be performed

"drive" is the drive number (0 or 1)

"diskname" is any valid string up to 16 characters in length

"id" is any 2 character diskette number

print #2,'NEW0:LEDGERS,01'
print #2,'N1:payroll,02'

This operation causes the diskette in the specified drive to be formatted and labelled for subsequent file operations. It is important to realise that this operation will cause the previous contents of the diskette to be lost.

2.2 Diskette Access Initialisation
print #2,'INITIALIZEdrive'

where

"INITIALIZE" or "I" specifies the operation to be performed

"drive" is the drive number (0 or 1)

print #2,'INITIALIZE'
print #2,'INITIALIZE1'
print #2,'I0'

This operation causes the specified drive or drives to be re aligned for file operations. If the drive number is not specified then both drive 0 and 1 are re aligned. This operation may not be necessary for certain types of Commodore disk drives. This operation should be performed whenever a new diskette is placed in a 2040 disk drive. (See also the Editor's MOUNT command.)

2.3 Reconstructing the Diskette Block Availability Map
print #2,'VALIDATEdrive'

where

"VALIDATE" or "V" specifies the operation to be performed

"drive" is the drive number (0 or 1)

print #2,'VALIDATE0'
print #2,'V1'

This operation causes the disk drive 0 to reconstruct the block availability map (BAM) of the specified diskette. Any allocated but unused blocks are freed for use by the disk file system.

2.4 Copying Files and Diskettes
print #2,'COPYdrivea:newname=driveb:oldname'

where

"COPY" or "C" specifies the operation to be performed

"drivea" is the drive number for the new file(s)

"newname" is the new file name

"driveb" is the drive number for the old file(s)

"oldname" is the old file name

```
print #2,'COPY0:save.file=1
                    :product.file'
print #2,'C1:PAYROLLBACKUP=1
                    :PAYROLLFILE'
print #2,'C0=1'
```

This operation causes a file to be copied and stored under a new file name. The new file must not already exist in order for this operation to complete successfully. If the file name portion of the specification is omitted then all files on the diskette will be copied to the destination disk drive. Note that this latter operation will fail if the diskette numbers do not match. (See Diskette Preparation for an explanation of diskette numbers.)

2.5 Backing Up a Diskette

```
print #2,'DUPLICATEdrivea=
                    driveb'
```

where

"DUPLICATE" or "D" specifies the operation to be performed

"drivea" is the drive number to which the diskette is backed up.

"driveb" is the drive number of the diskette you wish to back up.

```
print #2,'DUPLICATE1=0'
print #2,'D0=1'
```

This operation causes the diskette on the specified drive to be copied to the alternate drive. It is important to note that the equal sign is the diskette that will be backed up. The backup diskette will first be "prepared" by the disk. The previous contents of the backup diskette are lost. The backup diskette will have the same label as the source diskette. The files on the source diskette are then coied to the backup diskette.

2.6 Renaming a File

```
print #2,'RENAMEdrive
                :newname=oldnmame'
```

where

"RENAME" or "R" specifies the operataion to be performed

"drive" is the drive number (0 or 1)

"newname" is the new file name

"oldname" is the old file name

```
print #2,'RENAME1:oldfile=
                    current file'
print #2,'RO:JOURNAL.DATA=
                    Ledger.Items'
```

This operation causes the name of the file to be changed. The operation will fail if a file with the new name already exists. (See also the Editor's RENAME command.)

2.7 Deleting a File

```
print #2,'SCRATCHdrive:filename'
```

where

"SCRATCH" or "S" specifies the operation to be performed

"drive" is the drive number (0 or 1)

"filename" is the name of the file to be deleted

```
print #2,'SCRATCH1
                :OLDPAYROLL.FILE'
print #2,'S0:January.Records'
```

The specified file will be deleted from the diskette. (See also the Editor's SCRATCH command.)

Working with Random Numbers.
continued from page 21

This pokes various symbols on to the screen in various colours (both in the normal and multi colour mode).

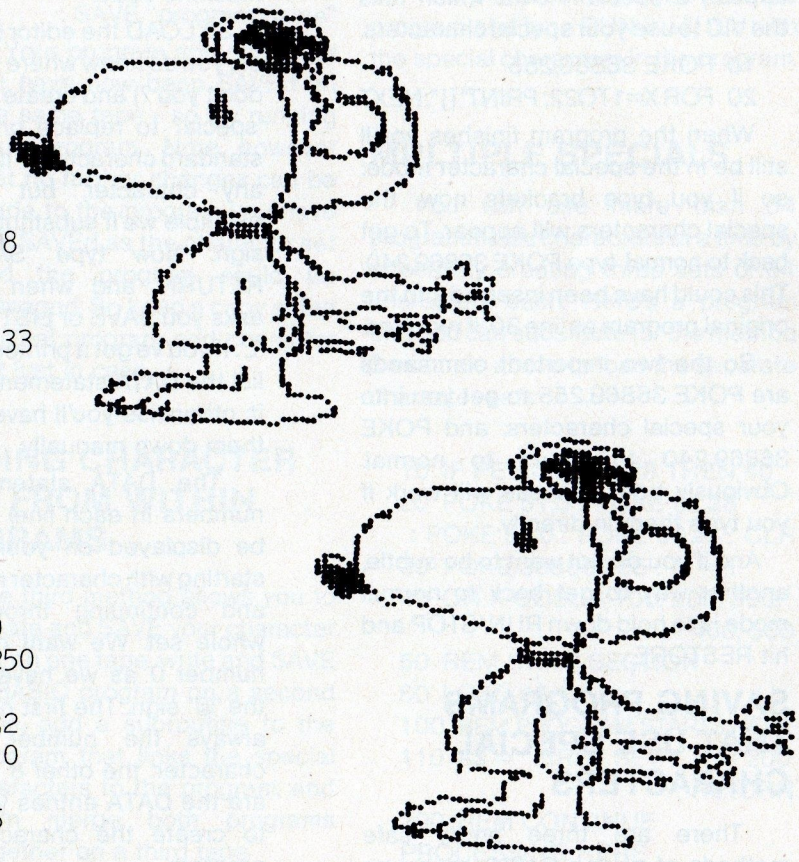
The screen display now is 200 x 240 pixels. This allows the programmer to use all the screen in his displays not just the centre section.

```
100 REM 25 BY 30 DISPLAY
        WITH NO EXPANSION
            PAST $1FFF
110 REM BY COLIN PRICE
120 REM
130 REM SET LIMIT OF MEMORY
140 REM POKE 51,255
    :POKE 52,27 :POKE 55,255
            :POKE 56,27
150 REM
160 REM PUT SPACES INTO
        NEW VIDEO MATRIX
170 FORI=7168TO7917
            :POKEI,32:NEXT
180 REM TELL SYSTEM WHERE
VIDEO MATRIX IS ($1C00)
190 POKE 642,28
200 REM POSITION DISPLAY ON
            T.V. SCREEN
210 REM THESE VALUES MAY
        HAVE TO BE CHANGED
        DEPENDING UPON THE
            T.V. ITSELF
220 POKE 36864,9
            :POKE 36865,24
230 REM
240 REM SET NUMBER OF
COLUMNS AND BIT 7 OF VIDEO
250 REM MATRIX ADDRESS FOR
            VIC CHIP ($1C00)
260 POKE 36866,25
270 REM
280 REM SET NUMBER OF
    ROWS AND 8*8 CHARACTER
            MAPPING
290 POKE 36867,60
300 CLR:PRINT"[clr]"
```

SNOOPY – *OR THE CASE OF THE PROGRAMMED MUT!*

Just for fun ... Snoopy on a Vic20!

```
100 rem pet benelux
110 rem exchange
120 rem netherlands
130 poke56,peek(56)-2:run140
140 cs=peek(56)*256
150 c$=chr$(34)
160 poke36879,42:poke36869,255
170 print"[clr,ctrl2,8cd,7sp]@abcdef
180 print"       hijklmn
190 print"       pqrstuv
200 print"         z[\]
210 print"        !"c$'&#$%&
220 print"       ()*+,-."
230 readx:ifx=-1then260
240 fori=xtox+7:reada:pokei,a:next
250 goto230
260 geta$:ifa$=""then260
270 print"[clr,ctrl7]";:poke36879,27
275 poke36869,240:poke56,peek(56)+2
278 end
280 data7424,0,0,0,0,0,0,0,0
290 data7168,0,0,0,0,0,3,12,16
300 data7176,3,4,9,19,247,23,11,12
310 data7184,192,48,200,228,247,231
315 data200,48
320 data7192,0,0,0,0,3,252,0,64
330 data7200,0,0,7,56,192,0,0,0
340 data7208,0,0,192,48,12,3,0,0
350 data7216,0,0,0,0,112,136,232,248
360 data7232,32,64,128,131,132,132
365 data132,68
370 data7240,15,24,48,225,65,33,33,33
380 data7248,192,128,128,0,0,0,0,0
390 data7256,64,0,31,32,64,128,128
395 data128
400 data7264,0,0,255,0,0,0,0,0
410 data7272,0,3,252,0,0,0,6,57
420 data7280,240,0,0,0,0,0,0,128
430 data7296,66,33,16,12,3,0,0,0
440 data7304,33,192,0,0,0,224,31,0
450 data7312,0,128,64,48,15,0,255,0
460 data7320,64,32,31,0,31,248,68,250
470 data7328,0,0,224,64,129,2,4,8
480 data7336,74,105,99,132,7,8,16,32
490 data7344,64,224,32,192,128,0,0,0
500 data7376,1,1,3,3,6,6,9,9
510 data7384,209,160,96,100,194,225
515 data208,168
520 data7392,208,56,7,0,0,0,128,64
530 data7400,64,128,0,224,24,4,2,1
540 data7432,0,0,0,0,0,0,1,254
550 data7440,18,18,38,38,76,148,20,40
560 data7448,228,230,232,103,49,30,1
565 data1
570 data7456,120,8,48,16,224,2,2,2
580 data7456,120,8,48,16,224,2,2,2
590 data7464,1,1,2,4,56,8,8,7
600 data7472,0,0,0,0,0,0,0,252
610 data7488,1,63,15,255,0,0,0,0
620 data7496,0,255,128,255,0,0,0,0
630 data7504,40,200,16,224,0,0,0,0
640 data7512,63,64,128,127,0,0,0,0
650 data7520,1,0,0,255,0,0,0,0
660 data7528,255,0,36,255,0,0,0,0
670 data7536,2,146,252,128,0,0,0,0
680 data-1
ready.
```



THE INSTANT CHARACTER EDITOR

O.K. you have all used the Games/Graphics Editor. Now is the time to set straight the way in which the character set that you generate using this program can be used in your own programs. This article discusses three methods that can be employed to use these sets. As you prodably know the games graphics editor only uses character sets of 64 characters, this is to allow the greatest amount of space for the BASIC program.

After you've created your characters and saved them using the Games/ Graphics editor. It makes sense to start by removing the editor program and keep only the character set you've created, so you can use the memory space to write the BASIC program around your new characters.

Write your new program with the new characters. For example, if you've redefined the bracket keys (character numbers 27 and 29) you would write a program in BASIC, using bracket keys where you want your newly created symbols and shapes to appear.

You also have to include on the first line (or wherever the special characters appear) a special POKE which tells the VIC to use your special characters.

```
10 POKE 36869,255
20 FOR X=1TO22: PRINT"[]":NEXT
```

When the program finishes you'll still be in the special character mode: so if you type brackets now, the special characters will appear. To get back to normal, type POKE 36869,240. This could have been inserted into the original program as line 30, if required.

So the two important commands are POKE 36869,255 to get you into your special characters: and POKE 36869,240 to return to normal. Obviously both of these will work if you type them in directly.

And if you do not want to be subtle, another way to get back to normal mode is to hold down RUN/STOP and hit RESTORE.

SAVING PROGRAMS THAT USE SPECIAL CHARACTERS

There are three pretty safe methods of saving BASIC programs that use your special characters. For a start, here's the cleanest method and probably the easiest to use for the beginner. It does take some time: but when it's completed you need no special actions to use the program.

1). Essentially, what you're going to do is list out the eight-bit codes for each special character and copy them manually into your program in a form the VIC can understand. This is actually how most of the professionally – produced cartridge – based programs are created and tested before they are put into machine code.

So LOAD the editor (oh come on, you do know where you left it, don't you ?) and create your first 'special' to replace one of the standard characters– it could be any character, but for this example we'll substitute the '@' sign. Now type 'SAVE', hit RETURN, and when the VIC asks you SAVE or LIST? answer 'L'. If you've got a printer, you can list the DATA statements out on it; otherwise you'll have to copy them down manually.

The DATA statements (9 numbers in each line) will then be displayed on your screen, starting with character number 0 and continuing through the whole set. We want character number 0 as we have defined the '@' sign. The first number is always the number of the character: the other 8 numbers are the DATA entries you need to create the character in a program.

Now turn your VIC off and on (or type SYS64802 and RETURN). Key in this program:-

```
10 IF PEEK(52)=28 THEN 60
20 POKE 51,0: POKE 52,28
: POKE 55,0: POKE 56,28: clr
                    : CB=7168
30 READ A: IF A= -1 THEN 60
40 FOR N=0 TO 7: READ B
  : POKE: CB+A*N,B: NEXT
50 GOTO 30
60 POKE 36869,255
70 PRINT CHR$(144) "@@"
    :FORX=1 TO 100: NEXT
80 PRINT CHR$(156) "A"
                    :GOTO 70
100 DATA 0,255,189,219,
      255,255,195,189,255
200 DATA -1
```

Note the base address is always 7168 (1C00 HEX), when using the games/graphics editor.

This program will SAVE and LOAD like any normal BASIC program, as it does not use any special methods to load in the character set. The first seven lines let you use the special character(s): the DATA statements define the first (zero) character which is the '@' sign.

More DATA lines can be used to define more characters. For example, here's a second DATA line to redefine character 1 (the letter A) in just the same way:

```
200 DATA 1,255,189,219,
      255,255,195,189,255,-1
```

Line 10 is the customised part and can be any BASIC program. Simply use the '@' sign or the CHR$ number to use it.

There are several uses for this technique. One is to redefine the

entire letter/number character set as foreign language characters - some budding entrepreneur might profitably market foreign language character sets for the VIC 20 using this editor to create them, by writing a program to redefine the entire character set with the games/graphics. Although this technique is best used for applications where only a few characters are going to be used.

The one problem that you may come across is that using this method to define characters all the other characters will be garbaged. So, for example, if you have to display that at the same time as the games characters you would have to define all of these characters as well.

One way to get around this problem is to first copy all or some of the ROM character set into the new RAM character set. Then the data statements can be overwritten onto this copy. This then gives you the original character set to use as well as the defined character set.

The following program could be used to do this type of copy:-

```
10 IF PEEK(52)=28 THEN 70
20 POKE 51,0 : POKE 52,28
   : POKE 55,0 : POKE 56,28
   : CLR : CB=7168
   : CR=32768
30 FOR C = 0 TO 63
   : P = PEEK (CR + C)
: POKE (CB + C), P : NEXT C
40 READ A: IF A=-1 THEN 70
50 FOR C = 0 TO 7 : READ B
: POKE (CB + (A x 8) + N), B
   : NEXT C
60 GOTO 40
70 POKE 36869,255
80 REM PROGRAM
   STARTS HERE
```

PUTTING CHARACTER SETS INTO PROGRAMS

2). The other standard approach is to design a game using your very own graphics. In addition to the one-character graphics we've shown, you can also create multiple-character shapes by setting up several characters to be printed together as a single large-size image. In other words, you can also create multi-character images on a larger scale.

Method number two adds characters to an existing BASIC program easily enough – but it does require that the original character set and the BASIC program are on separate tapes. LOAD in your program from tape, and add this as its first line:-

```
1 POKE 52,28: POKE 55,0
   : POKE 56,28 clr
```

Now find the end of the program by typing PRINT PEEK (45), PEEK (46).

Take the tape on which you SAVEd your special characters and put it in the datasette, type LOAD"",1,1 and RETURN – do not forget the extra ',1' on the end.

To save the entire BASIC program with the character set you can type this (in direct mode, with no line number needed):-

```
:POKE 45,0: POKE 46,30
   : SAVE "program name"
```

Your program and character set have now been SAVEd on the same tape - so try running your program. Note however that no further changes can be made to the program you have just SAVEd as the character set and the program could be damaged. So keep a copy of the original program and character set just in case.

LOADING CHARACTER SETS FROM WITHIN PROGRAMS

3). The third method allows you to create and SAVE your character set on one tape, write and SAVE a BASIC program on a second tape, add a subroutine to the program that links the special characters to the program, and then merge both programs together on a third tape.

So produce your special characters and SAVE them. Remove the tape and insert a new one. Write a BASIC program using the special characters and include the following LOAD subroutine:-

```
1 IF PEEK (52)=28 THEN 4
2 POKE 51,0: POKE 55,0
  : POKE 56,28: LOAD" ",1,1
3 POKE: clr
4 POKE 36869,255
   : REM SETUP CHAR
```

SAVE the BASIC program on the fresh tape and remove it. DO NOT REWIND THE TAPE!

Now reset the VIC (off/on or SYS64802), LOAD and RUN the editor, and LOAD the special character from tape one. Remove that tape and put tape two (the program tape) back into the cassette player. Now type S to SAVE your programmable characters on the same tape as the program: they will be automatically LOADed immediately after the BASIC program on the same tape with the program.

From this point on, if you load the program and type RUN it will run with the special characters in the program.

MULTIPLE SPECIALS

You can use more than 64 programmable characters at a time by effectively creating three sets of 64 characters each. Here's a program that you can substitute for the method 3 example above to load the separate character sets:-

```
10 IF PEEK (52) = 28 THEN 40
20 POKE 51,0 : POKE 52,28
 : POKE 55,0 : POKE 56,28 : CLR
30 POKE 36869,255 : N = 1
40 ON X GOSUB 100, 200, 300,
                400, 500
50 REM SWAP SECTION
60 LOAD A$,1,1
100 REM PROGRAM STARTS HERE
110 A$ = "FIRST SET" : N = 200
                : RETURN
200 REM CONTINUE
PROCESSING HERE
```

USING SOME STANDARD CHARACTERS

When using 64 programmable characters, the normal VIC characters are still available for your programs. When you POKE 36869, 255 and the character set is changed to, for example, game graphics, you can still include upper-case characters in your programs by PRINTing the reversed upper case characters.

USING THE VIC DISK

The VIC 1540 disk naturally makes it easier (and quicker) to store and retrieve programs that feature user-defined characters. All the instructions given should work, except that filenames for SAVEs and LOADs must be prefixed by an ampersand '&', when in the editor. And to adjust the programs to reflect this change by changing the device number to 8 from device number 1 – the cassette unit.

SAVING VARIABLES ON DISKETTE

The BASIC 'save' command saves the area of memory between the pointers 40,41 and 42,43 as a program format file. Separate pointers at locations 42,43 and 44,45 point to the start and end of simple variables respectively. Pointers at 52,53 point to the end of user memory. If one was to replace the pointers at 40,41 and 42,43 with those at 42,43 and 52,53 respectively the whole of a programme variable area can be saved on diskette as a program file that can be easily be re-loaded into this or subsequent programs. All of the variable names used in the first program will be preserved as will the variables value. If you wish to only serve simple variables and not arrays and strings then 42,43 should be replaced with the contents of 44,55. Below is an example program that will allow the whole variable area of a program to be saved as a file.

```
100 POKE 53,31 : CLR : REM lower top of memory to simulate an 8K machine.
120 A=1:B=2:C=3:D=4:E=5:F=6 :G=7:H=8:I=9:J=10 : REM create some variables.
130 A$="A"+"":B$="B""" :C$="C"+"":D$="END"+""
140 :
160 FOR A=0 TO 15
180 POKE 634+A,PEEK(40+A) : REM Save current pointers
200 NEXT A
220 :
240 POKE 40,PEEK(42) : POKE 41,PEEK(43) : REM Set new start
260 POKE 42,PEEK(52) : POKE 43,PEEK(53) : REM Set new end
280 DSAVE D0,"VARIABLES" : REM save variables - can use SAVE "0:VARS",8
300 :
320 POKE 42,PEEK(40) - POKE 43,PEEL(41) : REM Reset start of vars
340 POKE 40,1 : POKE 41,4 : REM Reset start of BASIC
360 END
```

To use the variables saved by the above program the following instructions would be used:

```
10 IF PEEK(32768) = 255 THEN 260 : REM Make sure we only load once
20 POKE 32768,25 : REM Set load flag
100 POKE 53,31 : CLR : REM Set top of memory to simulate 8K
120 :
140 FOR A=0 TO 15
160 POKE 40+A, PEEK(634+A) : RAM Set of memory pointers to previous values
180 NEXT A
200 :
220 DLOAD "VARIABLES",D0 : REM Load variables from disk
240 :
260 PRINT A,B,C,D,E,F,G,H,I,J
280 PRINT A$,B$,C$,D$
300 END
```

Either of these two routines could be incorporated as part of a BASIC program. Note the necessity in the first program to add a null ("") to the contents of the string variable. The reason for this is to force the variable to appear in the string storage section at the top of the user memory, rather than have the string pointers pointing into the BASIC text. In either of the above routines the POKE at line 100 is to set the machine up as an 8K, this is not necessary and can be easily left out.

ATTENTION – Super Expander Owners.

Some people with Super Expander Cartridges seem to have gained the mind over matter capability. As it is possible to DIMension a large Array – and still use high resolution graphics which takes away another 3200 bytes of storage.

For examaple, if we DIMension a single-dimension of array of say 600 numeric elements thus taking up (600+1) = 3005 +5 totaling 3010 bytes of storage.

(This is worked out as being 600 elements plus the 0 element, the array being 0 – 600 in size, each of these elements taking up 5 bytes of storage. All this plus 5 bytes for the header information, gives a total of 3010 bytes for the array.)

Then if we go into GRAPHIC 2 which also takes up 3455 bytes. This then gives a total memory consumption of 6465 bytes. Add to this approximately 109 bytes for the program and we get a total of 6574.

The significance of this is as follows. Remembering that when we turned on the machine we were left with just 6519 bytes. So if we subtract this from 6574 we are left with 55 bytes too few. These 55 bytes are going to be used twice. Once by the Super Expander for a graphics area, and once in our program for the storage of our array.

Look at the following program and see what happens:-

```
10 DIM N(600)
20 FOR C=0 TO 600 : N(C)=-999
                  : NEXT
30 GRAPHIC 2
40 SCNCLR
50 GRAPHIC 4
60 FOR C = 0 TO 600
        : PRINT C,N(C) : NEXT
```

The DIMension statement in line 10 stores 5 bytes of header information, and 5 bytes for each floating point number or element in the array. This is all written in ascending order from the end of program text. Moving up in memory.

Line 20 fills all of this array with a value so we can see any difference.

Lines 30 to 50 tells the Super Expander to get into the act, and clear a section of memory. The GRAPHIC 2 statment courses the Super Expander to allocate memory to itself for use in graphics. Line 40 forces the Super Expander to write this allocated memory with zeros, so clearing the screen. Which it writes from the limit of memory down, moving down towards the program and our array. The purpose of line 50 is to go back into text mode to display the array. This also has the effect of de- allocating the graphics area.

Once out of graphics mode the program lists the entire array, which has should be filled totally with the value -999.

Then we try to list all of the values stored in the array, we run into trouble. All goes well until we hit the border, the start of those 55 bytes. Then the VIC starts finding zeros in the array. Elements numbered from 589 to 600 will then be set to zero.

lines 30 to 50 then overwrite those 55 bytes, which only remain part of the array for a very short time. As they then get over-written by the Super Expander during the execution of lines 30 to 50.

Remembering that the VIC uses 5 bytes to store each element:-

55 divided by 5 = 11, so it all works out.

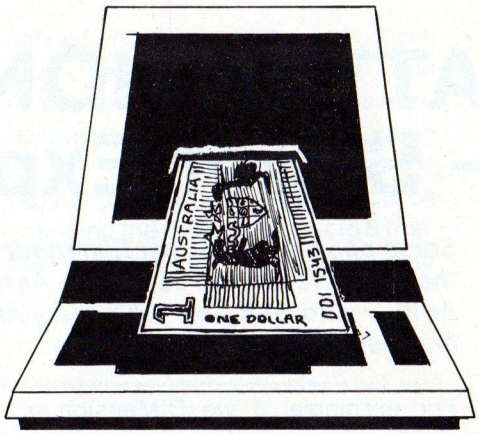
This will not happen with the 16K or 8K cartridge in place, as memory is configured so that one loses the graphics area before one starts. But, what it will do is to execute a CLR when it comes across the first GRAPHICs statement.

continued on page 41

"... T12X31LC, this is A561205,LC4A ... I have to make pee pee..."

THE PRINT MINT

Jim Butterfield
Toronto



The usual point of programs is that they produce output. The normal way of producing output is by using the PRINT command or it's cousin, PRINT#. We can abbreviate PRINT with a question mark (?), but oddly enough, PRINT # can't be shortened that way: typing ?# will produce a program line that lists as PRINT#, but doesn't work.

Other Ways.

We can generate output without using PRINT. It's not always good practice, but we can POKE to the screen memory area. This can be good for graphic games and animations, but there are several bonus things we get by using PRINT. First, PRINT keeps track of the line for us, and starts a new line as necessary. Secondly, PRINT doesn't limit output to the size of the screen: when the screen fills up, scrolling is automatic. Finally, and most important, PRINT can easily be changed to PRINT# to allow output to be directed to other devices such as printer, modem, disk or cassette tape. In contrast, screen POKEs are absolutely limited to the size of the screen, and can't be easily redirected anywhere else.

Punctuation.

If you say 'PRINT X' you will print the value of X and start a new line. The absence of punctuation at the end of the PRINT command signifies, "That's the whole thing; print it and wrap the line up." In contrast, if you say 'PRINT X;' you will print the value of X but you won't go to the next line: the invisible cursor will wait behind the printed value. This sounds a little backwards: you do something extra if you have no punctuation, but you do nothing if you have a semicolon.

There's one other form of "formatting" punctuation: the comma. If you type 'PRINT X,' you will print the value of X and then skip ahead to the next "column". Columns are considered to start at positions 11, 21, 31 and so on up to position 71. They exist only on the screen; saying 'PRINT#4,X,' to send to printer or other device won't set up columns properly. The comma will produce quick and convenient output to the screen, but it may be a bad habit since you can't use it anywhere else.

We can use this punctuation within a Basic PRINT statement as well as at the end. 'PRINT A;B$;C;' will generate the values of variable A, string B$, and variable C one behind the other and will leave the cursor positioned behind the value of C.

Neat Input.

We can use this punctuation to generate prompting for INPUT statements. For example, if we wanted to add ten numbers, we might code:

```
100 PRINT "INPUT EACH NUMBER:"
110 FOR J=1 TO 10
120 PRINT J;
130 INPUT X
140 T=T+X
150 NEXT J
160 PRINT "TOTAL IS";T
```

We prompt for the ten numbers with 1? ... 2? ... 3? ... and so on. The prompting number is printed by line 20 – J is stepping from 1 to 10 – and the question mark from the INPUT statement apears behind it because line 120 ends in a semicolon; after printing the number we wait on the same line so that the question mark will appear there. Question: What would happen if line 120 ended with a comma instead of a semicolon? Try it and see.

Number Formats.

Numbers are printed in special format. First, there is either a space for positive numbers or a minus sign for negative numbers. Then the number appears, as many digits as required plus a decimal point if needed and perhaps even "E" notation. (Never heard of E notation? Try PRINT 3E2 and see if you can figure it out). Finally, the number is followed by a cursor-right on the screen.

This seems at first to give you two spaces between numbers, but there are one or two fine points that are useful to know. If you type PRINT 2;3;4 you will see two spaces appear between each set of digits. Now try this: Type a bunch of x characters over the answer (a row of xxxxxx...) and then cursor back to the PRINT statement and press RETURN again. Some of the x's don't go away; that's because a cursor-right skips over that part of the screen without writing there.

There are a couple of ways to eliminate this difficulty if it bothers you. If you change a value to a string before printing, the cursor-right won't be performed. You could type PRINT STR$(2);STR$(3);STR$(4) – the same numbers will print with at least part of the problem solved. If you happen to have an 80-column or Fat-40 4.0 system, you may type: PRINT CHR$(16); CHR$(22);2;3;4 and you'll discover the problem is solved quite elegantly.

Here's another exception to the two-spaces rule: Type PRINT 2;-3;-4;5 and look at the result. The minus signs take up one of the two positions, and now there's only one space between some numbers.

Summary.

PRINT is handy and versatile. It takes a little while to get used to the formatting of the PRINT statement, but you'll soon have good control over your output.

There are some fascinating things you can PRINT which cause the screen to do unusual things. More about them another time.

PRINT- @ ROUTINE

Jacques Lebrun
Lennoxville, Quebec

Ever had to print at random locations on the screen? Most programmers probably have. The only way provided by the BASIC interpreter is to use cursor down, up, left and right enclosed within quotes in print statements. This could be impractical for some programs which constantly print something at different places on the screen.

Here is a cure: a machine language PRINT-AT routine that allows a BASIC program to print anywhere on the screen with a simple SYS command. The routine is location independent which means that it can execute well, no matter it's location in memory. The best place is probably the first cassette buffer for those who never use cassettes. The routine is for BASIC 4.0 but could be modified for other BASICs.

The routine can be used two ways. The first one is to position the cursor only. This allows further PRINTs, INPUTs and GETs to be done at that new location on the screen. Format is:

SYS ADDR,ROW,COL

The second way positions the cursor and prints whatever test follows the SYS. It's format is:

SYS ADDR,ROW,COL,TEXT

'ADDR' is the starting address of the routine, 'ROW' and 'COL' are two numeric expressions. 'ROW' ranges from 0 to 24 and 'COL' ranges from 0 to 79. 'TEXT' can be any numbers of string, integer or floating-point expressions seperated by commas or semicolons just as a normal PRINT statement.

```
100 REM *** BASIC LOADER
      FOR PRINT-AT ROUTINE ***
110 REM *** SET VARIABLE AD
          TO DESIRED STARTING
                    ADDRESS ***
120 AD=634
130 FOR I =AD TO AD+41
140 READ A
150 POKE I,A
160 NEXT I
170 END
180 DATA 32, 245, 190, 32, 212,
                    200, 224, 25
190 DATA 176, 28, 134, 216, 32,
                    111, 224, 32
200 DATA 245, 190, 32, 212, 200,
                    224, 80, 176
210 DATA 13, 134, 198, 32, 118, 0,
                    240, 9
220 DATA 32, 245, 190, 76, 168,
                    186, 76, 115
230 DATA 195, 96
```

---

continued from page 39

So this is the cause of all our problems and complaints. All because the Super Expander allows one to take memory space without saying OUT OF MEMORY. This is really our fault, as we should have gone into GRAPHICs mode before we declared the array, which would then result in OUT OF MEMORY.

To overcome this problem we require some sort of initialisation routine which will grab the memory away, so that it will not bother us again.

```
10 GRAPHIC 2
20 GRAPHICS 4
30 DIM N(600)
```

Unfortunately this little program does not solve the problem either. This is because the GRAPHIC 4 statement gives us back the memory that GRAPHIC 2 took away.

The easiest way I could think of getting around the problem is to not use GRAPHIC 4, but to simulate the effect of saying GRAPHIC 4 – but without deallocating memory.

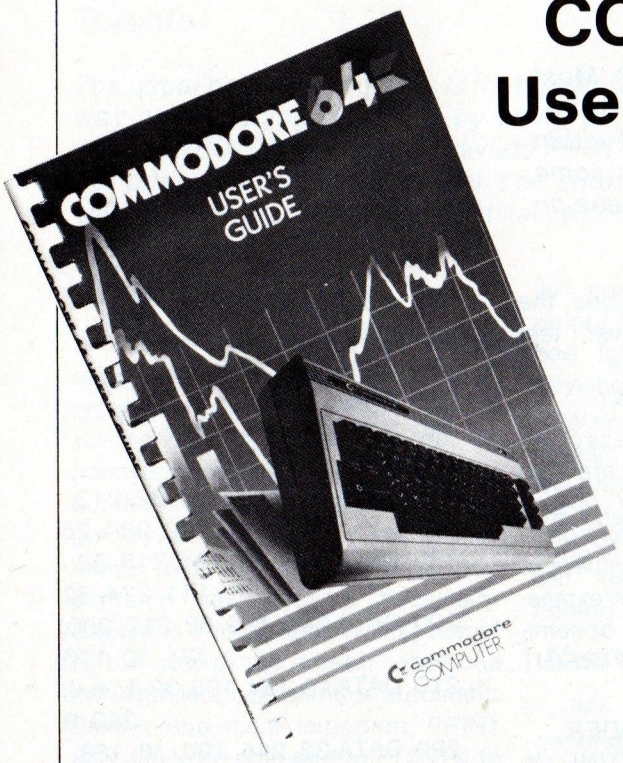
```
10 GOSUB 100 : GRAPHIC 2
                  : GOSUB 200
20 REM PROGRAM STARTS HERE
99 STOP
100 REM READ VIC CHIP
                    REGISTERS
110 FOR C=0 TO 6 : A(C) =
        PEEK(36864+C) : NEXT C
120 RETURN
200 REM RESTORE VIC CHIP
                    REGISTERS
210 FOR C=0 TO 6
          : POKE (36864+C), A(C)
        : NEXT C : PRINT "[CLR]"
220 RETURN
```

The aim of the program is to setup the Super Expander so it takes away the memory required and then to restore the screen only to normal mode. Once this is done all the graphics modes can be used, with the exception of GRAPHIC 4. Substitute GOSUB 200 instead, as this will avoid deallocating memory. Another use of this routine could be when one wants to go 'in' and 'out' of GRAPHICs modes without clearing the screen, as the Super Expander normally would.

This type of routine could also be used if you are using the 16K or 8K RAM catridges with the Super Expander. This is to avoid the operation of a CLR as could be expected on the execution of the first GRAPHICs statment in the program. But this is not strictly true. As the following program if executed at the very start of the program would have the same effect.

```
10 GRAPHIC 2
20 GRAPHICS 4
30 DIM N(600)
```

COMMODORE 64
User's Guide Update



Those of you who read both COMMODORE and POWER/PLAY may be perturbed with us for publishing this article in both magazines. But we're caught in a terrible dilemma. The 64 is an incredibly versatile machine, lending itself to both business-educational-scientific use AND home use. Although we can generally sort out which kinds of information go to which audience (a new game for the 64 obviously goes into POWER/PLAY, while a review of an electronic spreadsheet goes into COMMODORE, for instance), we sometimes have to get the same information to BOTH audiences. This is just such a case. Please forgive us, loyal fans, but that seems to be the nature of this computer.

Introduction

Commodore is constantly trying to bring you the most efficient and reliable computer in the world today. Along with the hardware improvements that come from practical applications of the Commodore 64 in the marketplace, the documentation should also reflect any changes and/or improvements that occur. This is the most up-to-date information available for your Commodore 64. The changes listed here should be used to replace the comparable information in your User's Guide. Future updates will normally be available through the Commodore User Magazines (COMMODORE and POWER/PLAY) as well as the COMMODORE INFORMATION NETWORK on CompuServe.

The format of this update is as follows:

A 1. Page and Paragraph or Section of the Commodore 64 User's Guide.
2. Old information.
3. New information.

EXAMPLE:

A 1. P.2, SIDE PANEL CONNECTIONS, 3. Game Ports
2. Each game connector can accept a joystick, game controller, or lightpen.
3. Each game connector can accept a joystick or game controller paddle, while the lightpen can only be plugged into the game port closest to the front of your computer.

The following list is in numerical order by page.

A 1. P.vii, INTRODUCTION, paragraph 3
2. . . . the SPRITE EDITOR lets you animate as many as 8 different picture levels at one time.
3. . . . at one time. The SPRITE EDITOR will soon be available as a software program that you can load directly into your Commodore 64.

B 1. P.vii, INTRODUCTION, paragraph 4
2. . . . a programmable ADSR . . . generator, an envelope generator, . . . filters for each voice
3. . . . a programmable ADSR . . . envelope generator and a programmable high, low, and bandpass filter for the voices

C 1. P.2, SIDE PANEL CONNECTIONS, 3. Game Ports
2. Each game connector can accept a joystick, game controller, or lightpen.
3. Each game connector can accept a joystick or game controller paddle, while the lightpen can only be plugged into the game port closest to the front of your computer.

D 1. P.3, figure 1
2. Control port at (3)
3. CONTROL PORT 1 CONTROL PORT 2 at (3)

E 1. P.3, figure 2
2. (5) CHANNEL SELECTOR
3. above (5) <−Ch.3 Ch.4−>

F 1. P.3, CONNECTIONS TO YOUR TV, 1.
2. . . . push it in. The cable will only go in one way.
3. . . . push it in. Either end of the cable can be used.

G 1. P.4, 6.
2. . . . channel selector switch (channel 3 or 4)
3. . . . channel selector switch (channel 3 move the switch to the left, channel 4 move the switch to the right)

H 1. P.8, USING THE COMMODORE 64, 1.
2. . . . the rocker switch on the left-side panel
3. . . . the rocker switch on the right-side panel when you're looking at the computer from the front.

I 1. P.10, TROUBLESHOOTING CHART
2. Picture with excess background noise
3. Sound with excess background noise

J 1. P.10, CURSOR
2. The flashing square next to READY . . .
3. The flashing square under READY . . .

K 1. P.14, KEYBOARD, paragraph 4
2. . . . the graphic character on the right side of the key.
3. . . . the graphic character on the right hand side of the front part of the key.

COMMODORE 64 COMMODORE 64 COMMODORE 64

L
1. P.14, KEYBOARD, last sentence
2. . . . special function keys . . . marked on the upper part of the key.
3. . . . special YELLOW function keys . . . marked on the front of the key.

M
1. P.18, TIP, paragraph 1
2. . . . normal display. Simultaneously depress:
3. . . . normal display. First press the RUN/STOP key and then press the RESTORE key. RUN/
   STOP must always be held down in order to use the RESTORE key function.

N
1. P.18, TIP, Last paragraph
2. . . . type, SYS64738 and press RETURN.
3. . . . type, SYS64759 and press RETURN.

O
1. P.18, 1. CARTRIDGES
2. . . . REMOVING CARTRIDGES OR YOU WILL DESTROY THE CARTRIDGE!
3. . . . REMOVING CARTRIDGES OR YOU MAY DAMAGE THE CARTRIDGE AND/OR YOUR COMMODORE 64!

P
1. P.19, 3. DISK, sentence 4
2. . . . protective gate by pushing in on the lever.
3. . . . protective gate by pushing down on the lever.

Q
1. P.20, paragraph 1
2. . . . blank, turning the background color of the screen . . .
3. . . . blank, turning the border color of the screen . . .

R
1. P.20, LOADING PROGRAMS FROM DISK
2. After LOAD "PROGRAM NAME,8"
3. The 8 is the code for the disk, so you're just letting the computer know that you want the program loaded from the disk.

S
1. P.29. Paragraph 2
2. You must always separate the parts of a mixed print statement with some punctuation for it to work properly.
3. You can separate the parts of a mixed print statement with punctuation for various formats.

T
1. P.34, EDITING TIPS, last paragraph
2. Try editing our sample . . .
   10 PRINT "COMMODORE";
3. Try editing our sample program on page 33 by changing line 10 and adding a comma to the end of the line. Then RUN the program again.
   10 PRINT "COMMODORE",

U
1. P.45, First paragraph
2. Pretty neat!
3. Pretty neat! To stop the program hold down RESTORE and hit RUN/STOP.

V
1. P.45, Last line
2. PROMPT MUST BE LESS THAN 40 CHARACTERS.
3. PROMPT MUST BE 38 CHARACTERS OR LESS.

W
1. P.46, Program
2. 20 IF A$ = "" THEN 10
   30 IF A$ = "F" THEN 100
   40 IF A$ = "C" THEN 50
3. 20 IF A$ = "" THEN 20
   30 IF A$ = "F" THEN 100
   40 IF A$ <> "C" THEN 10

X
1. P.48, Program Line 20
2. 20 GET A$: IF A$ = "" THEN 20
3. 20 GET A$: IF A$ = "" THEN 20 ←——No space between quotes

Y
1. P.49, Sample program
2. IF YOU . . . APPEAR AS 1 ROW
3. IF YOU . . . APPEAR AS 1 COLUMN

Z
1. P.50, General formula for random numbers
2. NUMBER=INT(UPPER LIMIT*RND(1))+LOWER LIMIT
3. NUMBER=INT(LOWER LIMIT+(UPPER−LOWER+1)*RND(1))

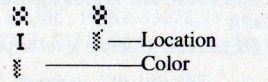
AA
1. P.51, Sample program
2. 60 IF GU = NM THEN PRINT "GREAT! YOU GOT MY NUMBER"
3. 60 PRINT "GREAT! YOU GOT MY NUMBER"

BB
1. P.51, Sample program
2. 100 IF AN$ <> "N" THEN 80
3. 100 IF AN$ <> "N" THEN 70

CC
1. P.57, KEYBOARD COLOR DISPLAY Chart
2. CTRL 1 BLACK +1
3. CTRL 1 BLACK +1

DD
1. P.58, COLOR CHR$ CODES, screen display
2. 10 PRINT CHR$(30);"CHR$(30) CHANGES ME TO?"
3. 20 PRINT CHR$(30);"CHR$(30) CHANGES ME TO?"

EE
1. P.64, Top of the page
2. POKE 1524,81
3. POKE 1524,81
   POKE 55796,1



FF
1. P.69, SPRITE CREATION, paragraph 5
2. Sprites are displayed in a special high-resolution mode which turns the screen into a 320 dot wide by 200 dot high area.
3. Sprites are displayed in a special independent 320 dot wide by 200 dot high area. However, you can use your Sprite with any mode, highresolution, low-resolution, text, etc.

GG
1. P.71, Line 230 in program
2. 230 DATA 1,62,64,0,15,128,0,156,128,0,73,0,0,73,0
3. 230 DATA 1,62,64,0,156,128,0,156,128,0,73,0,0,73,0

HH
1. P.72, first sentence
2. . . . you need to know from which 64 block section each series of 8 memory blocks will get sprites in their data . . .
3. . . . you need to know from which 64 byte section (1 whole sprite) each series of 8 memory blocks (1 row of sprite making data) will give each sprite its data . . .

II
1. P.72, Step 1
2. . . . location 21 which turns on the sprite.
3. . . . location 21 a 1 for the bit which turns on the sprite.

JJ
1. P.76, Paragraph 2
2. Now start to move in the usual X direction register for sprite 2 (which is in location 4 of the map) starting from 1 again.
3. Now start from 0 again in the usual X direction register for sprite 2 (which is in location 4 of the map).

KK
1. P.76, Additional notes on sprites, additional paragraph
2. . . .
3. To define multiple sprites, you may need additional blocks for the sprite data. You can use some of BASIC's RAM by moving BASIC. Before typing or loading your program type:
   POKE44,16:POKE16*256,0:NEW
   Now, you can use blocks 32 through 41 (locations 2048 through 4095) to store sprite data.

COMMODORE 64 COMMODORE 64 COMMODORE 64

LL
1. P.77, BYTE, second sentence
2. . . . you can actually have a total of 255 different combinations . . .
3. . . . you can actually have a total of 256 different combinations . . .

MM
1. P.80, STRUCTURE OF A SOUND PROGRAM
2. VOLUME, WAVEFORM CONTROL . . . EACH NOTE you play.
3. VOLUME, ATTACK/DECAY, SUSTAIN/RELEASE(ADSR), WAVEFORM CONTROL and HIGH FREQUENCY/LOW FREQUENCY. The first three settings are usually set ONCE at the beginning of your program. The high and low frequency settings must be set for EACH NOTE you play. The waveform control starts and stops each note.

NN
1. P.80, SAMPLE SOUND PROGRAM
2. 1. Set VOLUME at highest setting:
   10 POKE54296,15
3. First clear sound chip
   5 FORL=54272TO54296:POKEL,0:NEXT
   1. Set VOLUME at highest setting:
   10POKE54296,15

OO
1. P.80, SAMPLE SOUND PROGRAM
2. 2. Set ATTACK/DECAY levels
3. Set ATTACK/DECAY rates

PP
1. P.80, SAMPLE SOUND PROGRAM
2. 3. Set SUSTAIN/RELEASE rate to prolong note at a certain volume and release it.
3. 3. Set SUSTAIN/RELEASE to define level to prolong note and rate to release it.

QQ
1. P.80, SAMPLE SOUND PROGRAM
2. 40 POKE54273,17:POKE5427237
3. 40 POKE54273,17:POKE54272,37

RR
1. P.81, SAMPLE SOUND PROGRAM
2. 5. Set WAVEFORM to . . .
3. 5. Start WAVEFORM with . . .

SS
1. P.81, 7. WAVEFORM . . .
2. 7. Turn off the WAVEFORM CONTROL and ADSR settings.
   70 POKE 54276,0:POKE 54277,0:POKE54278,0
3. 7. Turn off note. 70 POKE54276,16

TT
1. P.81, MAKING MUSIC ON YOUR COMMODORE 64, program
2. 5 REM MUSICAL SCALE
   10 POKE 54296,15
   . . .
   50 READ A
3. 5 REM MUSICAL SCALE
   7 FORL=54272TO54296:POKEL,0:NEXT
   10 POKE 54296,15
   20 POKE 54277,7:POKE54278,133
   Sets Attack/Decay, Sustain/Release level (each note)
   50 READ A

UU
1. P.82, Sample program
2. 60 READ B
   . . .
   100 GOTO20
3. 60 READ B
   70 IFB=−1THENEND
   80 POKE 54273,A:POKE54272,B
   85 POKE 54276,17
   Start note
   90 FORT=1TO250:NEXT:POKE54276,16
   Let it play then stop note
   95 FORT=1TO50:NEXT
   Time for release
   100 GOTO20

VV
1. P.82, Last complete paragraph
2. To change the sound to a "harpsichord," change Line 30 . . . and RUN
3. To change the sound to a "harpsichord," change Line 85 to read POKE54276,33 and line 90 to read FORT=1TO250:NEXT:POKE54276,32 and RUN

WW
1. P.82, Last complete paragraph
2. . . . Changing the WAVEFORM can drastically change the sound by the COMMODORE 64 . . .
3. . . . Changing the WAVEFORM can drastically change the sound produced by the COMMODORE 64 . . .

XX
1. P.82, last sentence and POKE statement
2. . . . to a more "banjo" sound try changing line 20 to read:
   20 POKE54277,3.
3. . . . to a more "banjo" sound try changing lines 20 and 30 to read:
   20 POKE54277,3
   30 POKE54278,0 <——Sets no sustain for banjo effect.

YY
1. P.83, 1. VOLUME
2. . . . POKE 54296,15. The volume setting ranges from 0 to 15 but you'll use 15. The volume. . .
3. . . . POKE54296,15. The volume. . .

ZZ
1. P.83, 2. ADSR and WAVEFORM CONTROL SETTING
2. A sample waveform . . .
3. A sample waveform start setting . . .

AAA
1. P.83, 2. ADSR and WAVEFORM CONTROL SETTING
2. . . . the second number (17) represents a triangular . . .
3. . . . the second number (17) represents the start for a triangular . . .

BBB
1. P.83, ADSR and WAVEFORM CONTROL SETTINGS
2.

CONTROL SETTING	TRIANGLE	SAWTOOTH	PULSE	NOISE
. . .				

3.

	CONTROL REGISTER	Note Start/Stop Numbers TRIANGLE	SAWTOOTH	PULSE	NOISE
VOICE 1	54276	17/16	33/32	65/64	129/128
VOICE 2	54283	17/16	33/32	65/64	129/128
VOICE 3	54920	17/16	33/32	65/64	129/128

CCC
1. P.84, 2. ADSR and WAVEFORM CONTROL SETTING
2. . . . look at line 20 . . . this gave the scale a "harpsichord" effect.
3. . . . look at lines 85 and 90 in the musical scale program. In this program, immediately after setting the frequency in line 80, we set the CONTROL SETTING for VOICE 1 in Line 85 by POKEing 54276,17. This turned on the CONTROL for VOICE 1 and set it to a TRIANGLE WAVEFORM (17). In line 70 we POKE 54276,16, stopping the note. Later, we changed the waveform start setting from 17 to 33 to create a SAWTOOTH WAVEFORM and this gave the scale a "harpsichord" effect.

DDD
1. P.84, 3. ATTACK/DECAY SETTING.
2. The DECAY is . . . level back to zero.
3. The DECAY is the rate at which the note/sound falls from its highest volume level back to the SUSTAIN level.

EEE
1. P.84, 3. ATTACK/DECAY SETTING
2. . . . YOU CAN COMBINE ATTACK AND DECAY SETTINGS . . .
3. . . . YOU MUST COMBINE ATTACK AND DECAY SETTINGS . . .

FFF 1. P.85, 3. ATTACK/DECAY SETTING
2. 10 PRINT"HIT ANY KEY"
. . .
90 GOTO20
3. 5 FOR L=54272TO54296:POKEL,0:NEXT
10 PRINT"HIT ANY KEY"
20 POKE54296,15
30 POKE54277,64
40 POKE54273,17:POKE54272,37
60 GETK$:IFK$=""THEN60
70 POKE54276,17:FORT=1TO200:NEXT
80 POKE54276,16:FORT=1TO50:NEXT
90 GOTO20

GGG 1. P.85, 4. SUSTAIN/RELEASE SETTING
2. . . . Any note or sound can be sustained at its volume peak . . . you can even set the sustain level at its maximum (240) with no release to make a note play "indefinitely".
3. . . . Any note or sound can be sustained at any one of 16 levels.

HHH 1. P.86, 4. SUSTAIN/RELEASE SETTING
2. . . . how long the note will be held at peak volume . . .
3. . . . how long the note will be held at SUSTAIN volume . . .

III 1. P.86, 4. SUSTAIN/RELEASE SETTING
2. . . . combine a HIGH SUSTAIN LEVEL with a LOW RELEASE LEVEL . . .
3. . . . combine a HIGH SUSTAIN LEVEL with a LOW RELEASE RATE . . .

JJJ 1. P.86, Sample program
2. 10 PRINT"HIT ANY KEY"
. . .
90 GOTO20
3. 5 FORL=54272TO54296:POKEL,0:NEXT
10 POKE54296,15
20 POKE54277,64
30 POKE54278,128
40 POKE54273,17:POKE54272,37
50 PRINT"HIT ANY KEY"
60 GETK$:IFK$=""THEN60
70 POKE54276,17:FORT=1TO200:NEXT
80 POKE54276,16:FORT=1TO50:NEXT
90 GOTO60

KKK 1. P.86, Paragraph following sample program
2. In Line 45, we tell the computer . . . the "count" in Line 70.
3. In Line 30, we tell the computer to SUSTAIN the note at a HIGH SUSTAIN LEVEL (128 from chart above) . . . after which the tone is released in Line 80. You can vary the duration of a note by changing the "count" in Line 70. To see the effect of using the release function try changing Line 30 to POKE54278,89 (SUSTAIN = 80, RELEASE = 9).

LLL 1. P.87, Sample program
2. 10 V=54296:W=54276:A=54277:
. . .
40 POKEH,0:POKEL,0:POKEW,0
3. 5 FORL=54272TO54296:POKEL,0:NEXT
10 V=54296:W=54276:A=54277:S=54278:
H=54273:L=54272
20 POKEV,15:POKEA,190:POKES,89
POKE volume, attack/decay, sustain/release
30 POKEH,34:POKEL,75
POKE hi/lo freq. notes
40 POKEW,33:FORT=1TO200:NEXT
start note, let it play
50 POKEW,32
stop note

MMM 1. P.88, MICHAEL ROW THE BOAT ASHORE—1 MEASURE
2. 5 V=54296:W=54276:A=54277:
HF=54273:LF=54272:S=54278:PH
. . .
80 FORT=1TOD:NEXT:POKEHF,0:
POKEW,0
3. 2 FORL=54272TO54296:POKEL,0:NEXT
5 V=54296:W=54276:A=54277:HF=54273:
LF=54272:S=54278:PH=54275:PL=54274
10 POKEV,15:POKEA,88:POKEPH,15:
POKEPL,15:POKES,89
20 READH:IFH=-1THENEND
30 READL
40 READD
60 POKEHF,F:POKELF,L:POKEW,65
80 FORT=1TOD:NEXT:POKEW,64
85 FORT=1TO50:NEXT

NNN 1. P.127, RND(X) formula
2. N = INT (RND(1)*Y) + X
3. N = RND(1)*(Y-X) + X

OOO 1. P.129, OTHER FUNCTIONS, FRE(X)
2.
3. . . . the value of X. Note that FRE(X) will read out n negative numbers if the number of unused bytes is over 32K.

PPP 1. P.132, Paragraph 3
2. From BASIC, POKE 53272,29 will switch to upper case mode and POKE 53272,31 switches to lower case.
3. From BASIC, POKE 53272,21 will switch to upper case mode and POKE 53272,23 switches to lower case.

QQQ 1. P.136, numbers 129,149-155
2. 129
149
150
151
152
153
154
155
3. same as 97 129
same as 117 149
same as 118 150
same as 119 151
same as 120 152
same as 121 153
same as 122 154
same as 123 155

RRR 1. P.141, PINOUTS, Control Port 1 & Control Port 2
2. Note . . . MAX. 100mA
3. Note . . . MAX. 50mA

SSS 1. P.142, Cartridge Expansion Slot
2.

Pin	Type
22	GND
21	CD0
20	CD1
19	CD2
18	CD3
17	CD4
16	CD5
15	CD6
14	CD7
13	DMA
12	BA

COMMODORE 64 COMMODORE 64 COMMODORE 64

3.

Pin	Type
12	BA
13	DMA
14	D7
15	D6
16	D5
17	D4
18	D3
19	D2
20	D1
21	D0
22	GND

TTT 1. P.142, Cartridge Expansion Slot

2.

Pin	Type
11	ROML
10	1/02
9	EXROM
8	GAME
7	1/01
6	Dot Clock
5	CR/W
4	IRQ
3	+5V
2	+5V
1	GND

3.

Pin	Type
1	GND
2	+5V
3	+5V
4	IRQ
5	R/W
6	Dot Clock
7	I/O 1
8	GAME
9	EXROM
10	I/O 2
11	ROML

UUU 1. P.142, Cartridge Expansion Slot

2.

Pin	Type
Z	GND
Y	CA0
X	CA1
W	CA2
V	CA3
U	CA4
T	CA5
S	CA6
R	CA7
P	CA8
N	CA9

3.

Pin	Type
N	A9
P	A8
R	A7
S	A6
T	A5
U	A4
V	A3
W	A2
X	A1
Y	A0
Z	GND

VVV 1. P.142, Cartridge Expansion Slot

2.

Pin	Type
M	CA10
L	CA11
K	CA12
J	CA13
H	CA14
F	CA15
E	S02
D	NMI
C	RESET
B	ROMH
A	GND

3.

Pin	Type
A	GND
B	ROMH
C	RESET
D	NMI
E	S 02
F	A15
G	A14
H	A14
J	A13
K	A12
L	A11
M	A10

WWW.

1. P.142, Cartridge Expansion Slot

2. 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

	A	B	C	D	E	G	H	J	K	L	M	N	P	R	S	T	U	V	W	X	Y	Z
3.	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

	Z	Y	X	W	V	U	T	S	R	P	N	M	L	K	J	H	G	E	D	C	B	A

COMMODORE 64 COMMODORE 64 COMMODORE 64

XXX 1. P.147, Program
2. 220 K$="Q2W3ER5T6Y7UI9O0P@—*↑↑"
3. 220 K$="Q2W3ER5T6Y7UI9O0P@—*+£ ↑↑"

YYY 1. P.150, FORMULA TOO COMPLEX
2. . . . work with.
3. . . . work with or a formula has too many parentheses.

ZZZ 1. P.154, MUSIC NOTE VALUES
2. . . .

3. Filter Settings

location	*contents*
54293	low cutoff frequency (0-7)
54294	high cutoff frequency (0-255)
54295	resonance (bits 4-7)
	filter voice 3 (bit 2)
	filter voice 2 (bit 1)
	filter voice 1 (bit 0)
54296	high pass (bit 6)
	band pass (bit 5)
	low pass (bit 4)
	volume (bits 0-3)

AAAA

1. P.157, APPENDIX O, SPRITE REGISTER MAP
2. . . . . .
. . . . . .
. . . . . .
17 11 RC8 ECS BSM BLNK RESL YSCL2 YSCL1 YSCL0
3. . . . . .
. . . . . .
. . . . . .
17 11 RC8 ECM BMM BLNK RSEL YSCL2 YSCL1 YSCL0 Y SCROLL/MODE

BBBB

1. P.158, APPENDIX O, SPRITE REGISTER MAP
2. . . . . .
. . . . . .
. . . . . .
22 16 N.C. N.C. RST MCM CSEL XSCL2 XSCL1 XSCL0
. . . . . .
. . . . . .
. . . . . .
24 18 VS13 VS12 VS11 CB13 CB12 CB11 CB10 N.C. SCREEN Char Mem
3. . . . . .
. . . . . .
. . . . . .
22 16 N.C. N.C. RST MCM CSEL XSCL2 XSCL1 XSCL0 X SCROLL MODE
. . . . . .
. . . . . .
. . . . . .
24 18 VS13 VS12 VS11 VS10 CB13 CB12 CB11 N.C. SCREEN Char Mem

CCCC 1. P.161, VOICE NUMBER 2, WAVEFORM POKE
2. 54288
3. 54283

DDDD 1. P.161, VOICE NUMBER 3, SUSTAIN/RELEASE POKE
2. 54286
3. 54285

EEEE 1. QUICK REFERENCE CARD, SYSTEM COMMANDS
2. WAIT X,Y,Z Program waits until contents of location X, when FORed with Z and ANDed with Y, is nonzero.
3. WAIT X,Y,Z Program waits until contents of location X, when EORed with Z and ANDed with Y, is nonzero.

continued from page 11


```
CTRL CYN CTRL PUR CTRL BLUE CTRL YEL"
30 PRINTMID$(C$,X,1)"#";
40 GOTO10
```

Type RUN and hit the RETURN key to see the program. (Press RUN/STOP to STOP the program)

Now your crazy quilt makes RANDOM colors . . . the graphic symbols don't keep repeating the same order. Instead, they have an unpredictable pattern which really makes the quilt crazy! Here's what we did to change the program:

LINE 10 is the standard random formula, with a lower limit of 1 and a range of 8.

LINE 20 is the same as before. (For an interesting effect, try using 2 or more graphic symbols inside quotation marks instead of just the single "#" symbol)

LINE 30 is also the same.

LINE 40 makes the program go back to Line 10, where X becomes a new random number. The program keeps going back, using a new different random number (and color) each time.

Like those colors? The LOWER LIMIT is 1 and the RANGE is 8. Just for fun, try changing the lower limit to 3 . . . this eliminates the first two colors (black and white) and only PRINTs the colors from the third color in the string to the 8th color. If you want to eliminate the color white from the group (which only shows up on the screen as a blank square) you can eliminate the "WHITE" setting from Line 20 and reduce the RANGE to 7 in Line 10.

RANDOM Guessing Game

Here's a little guessing game which asks you to guess a number generated randomly by the computer, and tells you if you're too high, too low, or if you guess right . . . you might want to try adding your own sound effects, colors, graphics or other cosmetic touches:

```
10 N=INT(10*RND(1))+1
20 PRINT" CLR/HOME GUESS A NUMBER FROM
   1 TO 10 AND PRESS RETURN":INPUTA
30 IFA>NTHENPRINT" CLR/HOME TOO
   HIGH!":FORT=1TO700:NEXT:GOTO20
40 IFA<NTHENPRINT" CLR/HOME TOO LOW!":
   FORT=1TO700:NEXT:GOTO20
50 IFA=NTHENPRINT" CLR/HOME RIGHT!"
60 POKE36878,15:FORF=150TO250STEP2:
   POKE36876,F:NEXT:POKE36876,0
70 FORT=1TO700:NEXT:GOTO10
```

LINE 10 is the random number formula. Here we're guessing any number from 1 to 10 so the lower limit is 1 and the range is 10.

LINE 20 clears the screen, then PRINTs the opening "prompt" message that tells you what to do. INPUT A displays a question mark and waits for you to enter a number . . . then defines that number as the "numeric variable" A.

LINE 30 tells the VIC that if the number you typed in (called A) is larger than the random number the computer generated (called N), then PRINT the "Too High!" message, count to 700 (the FOR . . . NEXT loop) and go to Line 20 to repeat the message.

LINE 40 works just like Line 30, except here if your Answer (A) is less than the random Number (N) the VIC generated, the message "Too Low!" is PRINTed before going back to Line 20 to repeat the message.

LINE 50 determines whether the Answer (A) equals the random number (N) and if it does, it PRINTs the "RIGHT!" message and lets the program drop through to the next line . . .

LINE 60 sets the volume to its highest setting (POKE36878,15), then uses a FOR . . . NEXT loop to set up a short whooping sound effect. FORF=150TO250 STEP2 moves the VIC's sound generator through a series of musical note values from 150 (low) to 250 (high) 2 notes at a time (STEP2). When the program hits POKE36876,F, the VIC POKEs speaker number 36876 (see your user's guide) with the first note value (150). The NEXT command makes the VIC POKE the speaker with the next number in the FOR . . . NEXT loop which is 152, then 154, and so on up to 250. This creates the whooping sound effect. Finally, we turn off the speaker by POKEing it with a zero (POKE36876,0).

LINE 70 includes a time delay loop which keeps the "RIGHT!" message on the screen for a short duration before going back to choose another random number. Notice that here we use GOTO10 because we want to have the computer generate a new random number to guess. If you said GOTO20 here, the VIC would give you the "GUESS" message but use the same number you just used.

PROGRAMMING NOTE: Using IF . . . THEN Statements

When using IF . . . THEN statements, you can put MORE THAN ONE ACTION after the THEN portion and let the program "fall through" to the additional actions contained in the lines that follow. Here, the program is set up so it branches back to an earlier line in the program when it hits the first two IF . . . THEN statements in Lines 30 and 40. But when it hits Line 50, it CONTINUES BEYOND that line and performs the actions on the lines that follow (i.e. musical sound effect, time delay) until the program reaches the end of Line 70 which tells it to GOTO Line 10 and start over.

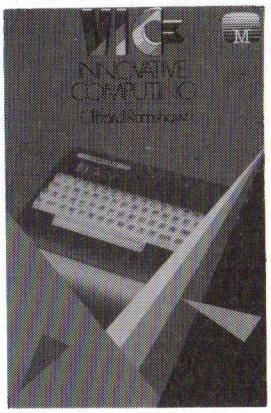
Using IF . . . THEN statements this way can help you set up whole series of actions (graphics, sound effects, etc.). The structure works like this (using imaginary program lines):

```
100 IF (condition) THEN (first action)
110 (More actions)
120 (More actions)
130 GOTO or GOSUB (line number)
140 (Next IF . . . THEN statement or continue program)
```

ATTENTION

VIC GAME HUNTERS

VIC INNOVATIVE COMPUTING



Opens new dimensions in your VIC–20. 30 sparkling and dynamic games written by the highly acclaimed author Clifford Ramshaw. Included are: *Space Flight; Hi–speed Maze; High–Res Draw; Warlock; Dragon's Lair; Synthesizer; Ganymede; Hoppy; City Bomber; Battleship; Maths; Duck Shoot; Grand Prix; Rat Trap; Earth Attack; Bomber Attack; Blackjack; Squash; Save the Shuttle; Hangman; Alien Overun; Invasion; Dumper; Seige; Snakes & Ladders; Dungeon; Golf; Nuclear Attack; Chess and Assassin.*

VIC INNOVATIVE CASSETTES

Three cassettes are also available in conjunction with the book. Eash cassette has captured seven prize winning ready to run games from the book.

VIC INNOVATIVE CASSETTE 1: *City Bomber; Dumper; Nuclear Attack; Ganymede; Space Flight; Battleship; Duckshoot.*

VIC INNOVATIVE CASSETTE 2: *Alien Overun; Rat Trap; Grand Prix; Warlock; Bomber Attack; Hangman; Seige.*

VIC INNOVATIVE CASSETTE 3: *Hoppy; Save the Shuttle; Invasion; Dragon's Lair; Dungeon; Blackjack; Squash.*

THE WIZARD AND THE PRINCESS

This program is a multi–part graphic adventure, taking you and your VIC–20 to the very ultimate in computer challenges. As you overcome each chapter of the story, you come closer to fulfilling the task of rescuing the princess from the clutches of the evil wizard. Trolls, Dragons, Dungeons and Danger! Some can even meet the challenge.

THE GAMES PACK

A collection of five new and exciting games.

"Alien Blitz" demands nerves and quick reflexes to destroy the enemy hovering above.

"Invaders" an exciting version of the arcade favorite.

"Storm" armed with your neutron missiles and anti–matter net, prevent aliens from reaching the edge of the time–space funnel.

"Ground Attack" penetrate the enemy, destroy the fuel dumps, dodge enemy missiles and reach enemy headquarters.

"Space Rocks" space adventure at its most challenging.

I would like to order the following

TITLE	COST
VIC Innovative Computing Book ($19.45)	.....
VIC Innovative Cassette 1 ($20.50)	.....
VIC Innovative Cassette 2 ($20.50)	.....
VIC Innovative Cassette 3 ($20.50)	.....
The Wizard and the Princess ($20.50)	.....
The Games Pack ($20.50)	.....
Please mail this coupon or a copy to:	_____
	_____ TOTAL

Computer Reporting Services,
Suite 204, 284 Victoria Avenue,
Chatswood, NSW 2067
telephone (02) 419 3277

Name .....

Address .....

.....

Post Code ..... Telephone ( ) .....

☐ Cheque enclosed
☐ Bankcard
☐ Diners Club
☐ American Express

Card No. .....

Expiration .....

Signature .....