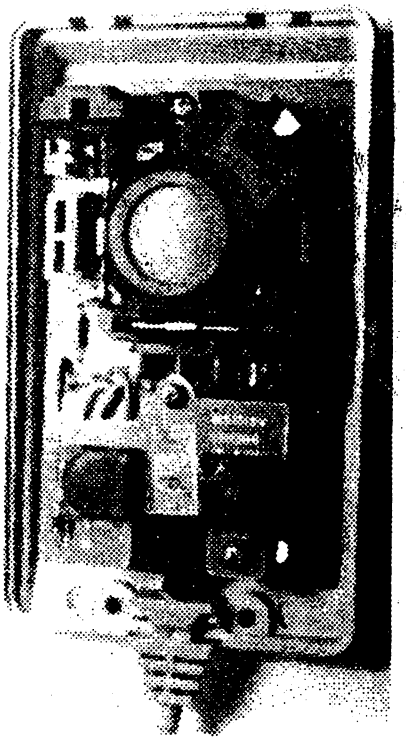
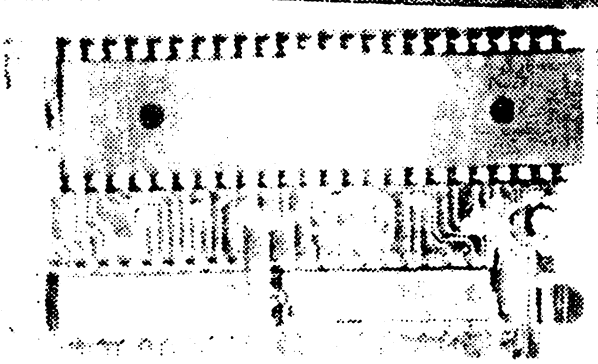
TWIN CITIES 128

THE COMMODORE 128 JOURNAL

PROUDLY PRODUCED COMPLETELY ON A C-128 IN NATIVE MODE !!!!

COMMODORE BUILT IT
WE SUPPORT IT!



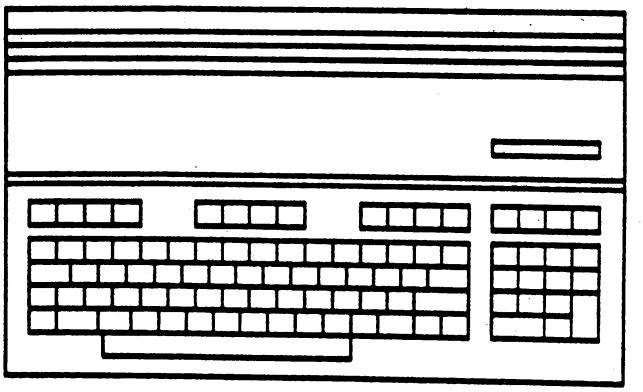


TWIN CITIES 128

COMPENDIUM BOOK #1

THE BEST OF ISSUES 1 - 18

$16.95 U.S.



Twin Cities 128...Then and Now

On December 27, 1985 the first issue of Twin Cities 128 was unveiled at a Metro-Area Commodore Computer Club meeting in Roseville, Minnesota. On that day, all thirty dot matrix printed and photocopied sixteen page issues were sold by a local Commodore dealer (Computer Stuff in Minneapolis) for $1.75 plus tax and I was elated. At that time I had no idea what this "Twin Cities 128" project of mine was going to entail or become. However, I knew that myself and my fellow owners of the then six month old Commodore 128 personal computer were hungry for every scrap of C-128 information that we could get our hands on. I was disappointed that the major Commodore magazines that had been feeding us a steady diet of preview and "teaser" information during the months before the 128's release were not following through with C-128 coverage now that the machine was in our hands. So I decided to do something about it.

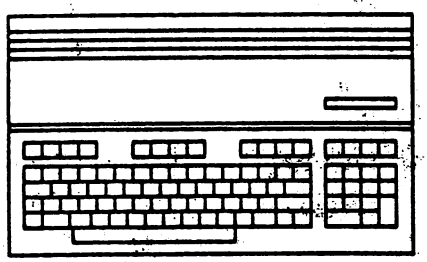
With lots of help from my wife Avonelle and two friends of mine who owned C-128s: Randy Margolis and Doug VanOrnum, I put together the first issue which included two C-128 software reviews (there were a grand total of six C-128 titles at the time) a BASIC 7.0 type-in programming example, some early thoughts on programming the C-128 80 column display controller, some usage tips, and a transcript of a conference with Commodore engineers Dave Haynie and Jeff Porter from Quantum link we got permission to reprint. I had no delusions about competing with the big magazines, after all it would be another three weeks before we would see our first "out of town" subscription.

Since those humble beginnings a lot has happened since Twin Cities 128 has grown and rose to prominence. While Twin Cities 128 will likely never achieve the kind of commercial success Commodore publications such as RUN, Compute's Gazette, and INFO have obtained, I am quite proud of what we have accomplished. To this day we are still the only publication in North America devoted exclusively to using and programming the Commodore 128. And there is no other publication that has come even close to publishing as much C-128 information. Each issue of Twin Cities 128, from that first issue produced in late December of 1985 to today (as I write this we are preparing our 21st issue) has been produced entirely with Commodore 128s as was the book you are reading right now. I have always felt that producing Twin Cities 128 with and only with the C-128 was an important symbol of our dedication to, and our prowess with, the C-128. Through the actual use of C-128s for every facet of our production and administration from our subscriber database to the actual printing of the issue we have gained invaluable insights into how to best use and program our subject. Necessity is indeed the mother of invention. Another thing I am proud of is that Twin Cities 128 has become THE PLACE to turn to as far as C-128 information is concerned. So much so that: Commodore personnel often call or write us regarding various C-128 matters, Quantum Link asked us to manage the C-128 section on their Commodore Information Network, and INFO Magazine wrote, "If you are serious about using or programming your C-128, you need this publication." (INFO Magazine, Issue #17, Page 22) Perhaps even more amazing is the fact that such a reputation has been gained primarily from the work of a volunteer staff, a non-existent advertising budget, and no outside sponsorship. Lastly, I am proud to say that throughout this endeavor, I believe that we have never stopped enjoying what we are doing, and we have remained accessible and responsive to our readers.

This book is a compilation of the best material from our first eighteen issues. I regret that due to space limitations we had to drop about 25% of the material we actually published. For the most part we dropped those articles that contained information that was either redundant or of questionable long term significance. My only regrets as far as this editing process is concerned is that we could not include any of the numerous interviews/conferences we had with C-128 newsmakers nor a C-128 commercial product listing as we did as issue 11 and as part of 16. Overall, however, I hope you will find that this book contains the same humor, insight, and technical prowess that has made Twin Cities 128 popular.

Lastly, I'd like to thank some of the individuals who have made this effort so successful. Perhaps the person I need to thank the most is my wife, Avonelle Lovhaug, who not only has contributed many wonderful articles, but also has kept this publication running smoothly by doing everything from proofreading and laying out articles to database management, and correspondence. There have been many occasions when she has persevered layout sessions, long phone conversations, and the all too frequent transformation of her home into meeting room, production center, and disaster area. Beyond that she has put up with my frequent mood swings, occasional lack of confidence, occasional over-confidence, and incomprehensible management style. Three other individuals I am deeply indebted to are Miklos Garamszeghy, Bruce Jaeger, and C-128 co-designer Fred Bowen who have continually given us first rate material without receiving the compensation they command when writing and working for other companies in the Commodore world. Additionally, this publication would not have made it past the first issue without the contributions of Randy Margolis and Bill Nicholson who have not only contributed a wealth of quality material but have constantly served as critics and advisors. It is their suggestions and analysis that have made each and every issue better than its predecessor. I also must thank Curt Swanson who was that first computer dealer that "sold out" our initial issue. Curt has constantly advised us on matters of style, marketing, and production and has put up with numerous production and delivery delays. Lastly, I'd like to thank my parents who helped finance my education, put up with my unorthodox career choice, babysat my children, and played taxi from time to time.

Loren Lovhaug, May 1988



ISSUE 1

ISSUE 2

ISSUE 3

ISSUE 4

ISSUE 5

ISSUE 6

ISSUE 7

ISSUE 8

ISSUE 9

ISSUE 10-11

ISSUE 12

ISSUE 13

ISSUE 14

ISSUE 15

ISSUE 16

ISSUE 17

ISSUE 18

Rumor/Opinion/Mayhem *by Loren Lovhaug*

Welcome to the first issue of Twin Cities 128 magazine! The entire over-worked, under-paid staff of TC-128 has worked long and hard to bring you what many of you have been starved for: coverage of the amazing Commodore 128 personal computer! Now at this point you may well be thinking to yourself, why does a group of supposedly sane, reasonably intelligent individuals take on such an endeavor? Well, to tell the truth I am actually not sure. I would like to say it involves pure profit motive, but to be perfectly honest I am not really sure whether or not we can turn a profit, and in reality this is a huge longshot. Of course, this is probably a direct result of the lack of any big-time corporate investment in this little venture. Believe it or not, I have yet to hear from any Fortune 500 companies interested in sponsoring us. But then again, sitting here in front of my trusty C-128, contemplating the future of our publication, I honestly believe that someone else besides my wife will actually read what we have to offer, and perhaps even like it! And besides, this is a Commodore computing journal, a journal for the masses, (masses? who am I kidding) a source of information for those of us on a budget, those of us with the knowledge that there does exist computing power outside the Ivory towers of IBM and the Avande Garde California Chic of Apple. Of course, it is going to be an uphill battle, but the future lies with us. I sincerely believe that the future of the computer industry does not lie with the glossy, the grand, or the leading edge, but in fact is encrusted in the lowest common denominator, that which people deem to be practical and affordable, and above all reliable. In my book, the C-128 fits the bill offering the "blue collar computerist" the best bang for the buck. And I guess, above all that is why we are doing this, because like you we appreciate a bargain and are truly committed to doing the most we can with what we've got!

Superbase 128 *by Randy Margolis*

With the advent of programs like Superbase 128 from Precision Software it seems the 128 has become legitimized rather more quickly than anyone had imagined. Try to remember any other new computer which had high quality software available for it a mere two to three months after its introduction.

Superbase 128 is basically an updated and improved version of the immensely popular Superbase 64, so anyone familiar with that program will have no trouble converting to the C-128 version. However, the degree of improvement is such that a person would, in my opinion, be perfectly justified in purchasing the new product. Unfortunately, there is no upgrade policy for changing from Superbase 64 to the 128 version at a discount, but I have no regrets at having done so myself at full price.

Superbase 128 is a programmable database-filing system which enables one to keep track of almost anything. You first set up a screen format for your record template starting with a blank screen and laying out "fields" into which you will type information for each different "record" of your file. Superbase 128 stores these records in alphabetical-numeric order automatically by the field which you designate as the "key field", which can occur anywhere in the record and is defined at the time you set up the file. If, at any future point, you want to sort your records by any other field, Superbase 128 will oblige easily and quickly. All you do is select "Sort" from a screen menu, designate the field you wish to sort on, and a filename to contain the sorted index. The program will read through the records with amazing speed and write the sorted index back to disk before you know it. Likewise, if you want to create a sub-list of only records that meet certain requirements you provide the specifications and Superbase 128 creates the subsidiary list very quickly. Indeed, the power of this program is limited only by your imagination.

Let me state something about the aforementioned speed of operations of Superbase 128. This lightning fast speed will only be fully realized if you use a 1571 disk drive with the program. It will work perfectly well with an older 1541, but much of the speed advantages will be lost, since the 1541 is capable of a data transfer rate of only 300 bits per second, while the 1571 can transfer up to 4000. Yes, Superbase 128 utilizes the burst mode of the 1571 and this accounts for much of its amazing speed.

One of the program's main features is to give the user the ability to write programs to control the many features of Superbase 128. Within the program there is a large area of memory set aside for these user written programs. Superbase 128 has its own built-in programming language which consists of many of the C-128's BASIC 7.0 commands in addition to its own instruction set. You have approximately 65k of space for your programs and variables, which is totally adequate for any conceivable application. For example, you can select records from files, sort them, and print out formatted reports on your printer, complete with underlined headings and added up columns of numbers. Now, learning to accomplish all this will require some effort on the part of the user, proving a theory I have that the better the program the more effort the user must expend to utilize the features to their fullest. It's really not all that difficult, and there will doubtless be some third party books written to help get the most from this program. So, you don't actually have to learn programming to use most of the features. For example, Superbase 128 has a built-in report generator in which the user answers several questions and a report program is automatically written.

Superbase 128 Continued

and saved to your disk. After you learn more about Superbase programming you can easily modify these reports and build in your own enhancements to make your printed copy look even better. As a matter of fact, I know users who have never once written a Superbase program but use the database capabilities to get most everything they need accomplished.

Superbase 128 has a very handy provision for "importing" and "exporting" data from your database files. By utilizing this capability, you can create regular sequential files containing all the records or a particular subset of selected records in your file for use by another program (export), or you can bring a similar file created by another program into the Superbase 128 database, thereby painlessly converting from your present database system to Superbase 128 (import). All you need for the latter is the capability of creating sequential files of information from your current database or word processing program.

Another unique feature is its ability to coexist in memory with Superscript 128. Precision Software's new word processor. This means that you can, for example, write a form letter on the word processor, draw the mailing list information directly from the database, and automatically insert it at the proper place in the document. When both programs are loaded a simple command switches between the word processor and the database, and back again as many times as you want. This does cut down the size of the text space in the word processor to about 150 80-column lines, but I feel this is sufficient for most applications in which you will need to have the programs co-resident.

Finally, for current users of Superbase 64, Precision has provided a utility which converts your present data files to the 128 mode automatically using one or two drives, and the manual contains complete instructions for converting the file definitions to 80 columns if you wish. Superbase 128 works with both 40 and 80 column files. This characteristic is carried within each file definition, so on the same data disk you can have both 40 and 80 column files and switch between them quite easily. Superbase 64 users will find much that is familiar in this program, but I think that the 80 column capability, the greatly enhanced speed, the ability to use double sided disks on the 1571 drive, and the vastly improved programming environment fully justify the upgrade to Superbase 128.

Paperback Writer 128 *by Loren Lovhaug*

Paperback Writer 128 is an extremely sophisticated, yet easy to use word processor by Digital Solutions Inc. (P.O. Box 345 Station A, Willowdale, Ontario, Canada, M2N 5S9, 1-416-221-3225) Paperback Writer was one of the first commercial programs available for the C-128, it is destined to become a classic!

Unlike most word processors previously designed for use on Commodore computers. Paperback Writer is a completely formatted 80 column word processor. This means you see on the screen exactly what you get on the paper; italics are italicized, boldface characters are bolded, underlined characters are underlined, centered text is centered, and text is aligned exactly according to the margins and offsets you specify on an external format window outside the body of the text. With Paperback Writer, gone are the days of editing around cryptic code characters and command lines embedded inside the actual text itself, and then using clumsy view modes in order to determine what your text is actually going to look like. This ability to "see what you get" is perhaps Paperback Writer's greatest asset, but it is certainly not its only one.

Paperback Writer includes the usual features you would expect from any professional quality word processor such as the ability to: move, copy, and delete ranges of text, automatically find and replace words and phrases, automatically indent, fully control margins, borders and text alignment, automate page numbering, create headers and footers, and join and link files. These features combined with on-screen formatted text alone would make Paperback Writer an excellent value, but there is even more!

Paperback Writer has the ability to automatically align and add columns of numbers, as well as the ability to sort columns of both numbers and text in ascending or descending order. This makes it extremely easy to prepare documents where charts are included. These features also come in handy when integrating data created by database and spreadsheet programs into your documents. Paperback Writer also includes a complete mail merge facility allowing you to create "variable blocks" (blanks) within your document and later fill those blocks with specific information from a list stored on disk.

Paperback Writer is written entirely in machine code and has been designed to take advantage of all of the special abilities of the C-128. Besides the use of the C-128's 80 column screen the program also makes optimum use of the C-128's 128k (128 kilobytes) of random access memory. The program itself requires only half of the C-128's random access memory for operation, leaving a whopping 64k of memory for document space; more than enough for most chores (I

Paperback Writer 128 Continued

have used Paperback Writer with a document as large 35 pages (double-spaced) with memory left to spare). Because of Paperback Writer's compact size the program itself loads extremely quick even on the 1541 disk drive, and on the newer and faster 1571 it seems like no time at all! Additionally, Paperback Writer incorporates a feature which allows you to partition your document memory into two 32k banks, giving you the ability to edit two separate documents concurrently as well as the ability to move ranges of text from one document to the other. This feature is simply outstanding, and is invaluable for anyone who is contemplating large projects involving major revisions.

Paperback Writer is also extremely easy to learn and use. Paperback Writer comes packaged with a complete though somewhat terse manual which covers all aspects of program operation. I found Paperback Writer's command structure to be very logical and smooth, with just the right amount of prompting without an over-reliance on menus which tend to slow down the advanced user. Paperback Writer also features an on-screen help window which provides useful command summaries and descriptions. The user can also get even more detailed information simply by placing the program disk into the drive and following instructions. I found these summaries so good that it is conceivable that someone could become proficient at using Paperback Writer without ever touching the manual. These on-screen help features keep the beginner from becoming frustrated. For the advanced user the on-screen help window can be turned off to allow more text to be viewed at one time.

Another advantage Paperback Writer offers is that it is completely compatible with many of the best word processors available for the C-64 such as Paperclip, WordPro, and others which store their documents in a similar format. Those of you who have experienced the misery of having to convert files between incompatible programs know what a boon Paperback's friendliness towards other word processors really is.

Like most programs, there is one drawback I have found with the Paperback Writer. The package I own (version 1.15) comes with an external spell-checker which is extremely slow, and tedious to use. The spell-checker, unlike the word processor itself operates in C-64 mode and appears as if it were put together somewhat hastily as a last minute "extra". The spell-checker has difficulty handling large documents and does not include its own dictionary file, so you must build your own dictionary using files, which can be a long process.

In light of all of the features I have mentioned above, I find Paperback Writer to be an excellent value and a superb addition to anyone's software library. I believe Digital Solutions has created a standard by which all other word processors for the C-128 will be judged and I recommend this product to both the beginner and the advanced user.

Sparrow's Slick Tips *by Sparrow James*

Slick tips; I thrive on them, I really get a kick out of making my computers do the utterly amazing, impossible, and the unique. I also insist on trying to take advantage of every little short cut and time saver I can, because you see I am extremely impatient and like "QUICK CARL" (of Marathon candybar fame) I Do Everything Fast (2.0 megahertz+). I think you will find the following "bag of tricks" interesting and helpful. thanks to all of those who contributed them. If you have found any neat little tricks that really make your C-128 sing, please drop me a line, either by sending your tip to our mailing address or leaving E-mail to: Sparrow on Quantum Link.

00000001: GHOST IN THE MACHINE Source: 128th precinct, Quantum Link.
A secret message left for you by the gremlins who designed the C-128. Just type the following in immediate mode and the message will magically appear on your 40 column screen: SYS 32800,123,45,6
Apparently, this little ditty got by the powers that be at Commodore, and when it was discovered caused quite a stir in West Chester.

00000010: DISK DOUBLER IN C-64 MODE Source:THE MASTER
It was first was reported that the 1571 disk drive would completely emulate the 1541 when the C-128 was in placed C-64 mode. This is not entirely accurate. While it is true that when you are using the 1571 in C-64 mode you are hampered by the 1541's much slower speed, you can take advantage the 1571's ability to read and write data to both sides of a diskette, therefore giving you twice as much storage capacity per diskette. There are two methods for ensuring the drive is locked into 1571 mode while using C-64 mode:
1) Enter C-64 mode via the GO64 command from BASIC 7.0 in 128 mode after doing any "fast serial disk access". Getting a disk directory from 128 mode will accomplish this.
2) Typing the following command before or after enter C-64 mode: (this command will lock the drive in 1571 "double sidedmode" until the computer or the disk drive is reset): OPEN 1,8,15,"U0>M1"
To lock the drive in 1541 mode (single-sided and SLOW) power up the computer while holding down the Commodore key or ype the following command: OPEN 1,8,15,"U0>M0"

Sparrow's Slick Tips Continued

00000011: SPEEDING THINGS ALONG! Source: Sparrow's secret file

You can significantly increase the execution speed of your BASIC 7.0 programs and disk operations by taking advantage of the C-128's FAST mode. The FAST mode doubles the clock speed of the 8510 microprocessor to 2.04 Megahertz. You can enter FAST mode by simply typing the command: FAST in immediate mode or adding it to any program. The cost of this increased speed will be the blanking out of your 40 column screen (not a big deal if you are using the 80 column screen). You can reverse the effects of FAST mode by typing the command: SLOW in immediate mode or adding it to a program.

00000100: C-64 MODE SHOT IN THE ARM Source: The 128th precinct, Quantum Link

You can take advantage of the same FAST mode speed increase as listed above by typing POKE 53296,1 or including it in your BASIC programs. POKE 53296,0 returns the microprocessor to SLOW mode. Again you will lose the forty column screen (your only screen in C-64 mode) but for non-screen intensive work this one does wonders. (This only works in C-64 mode on the C-128 and not in the normal C-64 since it does not have the 2.04 Megahertz 8510 chip.)

00000101: DUELING MONITORS Source: Sparrow's personal preferences

One of the things I like most about the C-128 is its ability to display two different text or graphic screens concurrently. This is possible because there are separate circuits and ports handling the processing for 40 and 80 column screens. When programming I suggest you take advantage of your dual display modes by using either a dual display monitor like the Commodore 1902 or (my preference) using a monochrome composite monitor for the 80 column output and a color composite monitor for the 40 column output. By taking advantage of your dual displays you find program editing and de-bugging much easier, especially when working with graphics.

00000110: KEYPAD CALCULATOR Source: Sparrow's frustrated fingers

Of course we all realize the benefits of having a numeric keypad available to us in C-128 and CP/M modes, but it would have been nice if Commodore would have provided some extra keys with the keypad group to better facilitate more advanced calculations, especially in immediate mode. Fortunately, thanks to C-128's programmable function keys this problem is easily solved with the following program:

```
10 KEY 1,"PRINT   "
20 KEY 3,"*"
30 KEY 5,"/"
40 KEY 7,"("
50 KEY 8,")"
```

00000111: ONE KEY SCRATCH Source: Sparrow's lack of disk space

When doing my semi-weekly disk purges I find defining a function key as following to be of great assistance:

```
KEY 4, "SC[shift r]"+CHR$(27)+"K"+"[3 cursor lefts]"+chr$(27)+"@"+chr$(13)
```

List a DIRECTORY to the screen and position the cursor on the line at the leftmost position of the line at which the program you wish to scratch is listed and hit the function key, you then will be prompted whether or not you wish to scratch the file.

Note: SC[shift r] is the abbreviation listed for the scratch command and "shift R" ought to produce a graphic character. The three cursor lefts must be enclosed in quotes and will be represented on the screen as reversed vertical lines. Also this key definition is tedious to retype so it is perhaps wise to save it to disk as a one line program.

00001000: DISABLING RUN/STOP & RESTORE

* In C-128 mode: The RUN/STOP key can be disabled via POKE 808,112
  The RESTORE key can be disabled via POKE 792,98
* The RUN/STOP key can be enabled via POKE 808,110
* The RESTORE key can be enabled via POKE 792,64

Rumor/Opinion/Mayhem *by Loren Lovhaug*

Here it is the second issue of Twin Cities 128. I cannot believe how fast things in the C-128 world are developing. It seems like it was only a short time ago. I was gazing at the preview in the May issue of RUN magazine wondering how well the "soon to be released" C-128 was going to do in the depressed personal computer market of 1985. I could see by the previews that this machine was going to offer a great deal of power, flexibility, and value. But, I realized that power, flexibility, and value were not enough to make the C-128 a success. Five years ago, any machine which boasted the C-128's statistics would have been an instant market leader, not only in the personal computer market, but in the business and educational computer markets as well. But things are much different now. Now there are many factors which serve to make or break a personal computer. Quality of the product, price, image and reputation, marketing strategies, and the machine's staying power in a rapidly changing and developing industry are certainly among some of the most important considerations. But I think perhaps the single most important contributing factor to the success or failure of a machine in today's marketplace is its SOFTWARE.

This fact becomes even more apparent when you examine the histories of various machines which have appeared, and in many cases disappeared during the past few years. Starting in our own backyard, the Commodore 64 was and remains an immensely profitable and popular computer, but looking back it did not become so until its price dropped (from its original $595) and a cavalcade of inexpensive quality software hit the market. Compare this scenario with the legacy of the Vic-20, the C-16, and the Plus/4, all Commodore computers which in their own right had their advantages, but died almost directly as a result of the lack of an acquired software base. Turning away from Commodore, we see the same scenario at Apple. The Apple II series computers have until recently been the flagship of the Apple line, these computers developed a strong following not because of their power, sophistication, or price (they tend to be weak on all three fronts) but because the Apple has acquired an immense software base, especially in the educational and entertainment markets. In comparison, Apple's new darling the Macintosh (until just recently) has suffered severely from a lack of viable, affordable software, even though it has been available for over 2 years. Machine to machine, company to company, the same seems to hold true: Atari, Matel, Coleco, Epson, Timex/Sinclair, Texas Instruments, and Radio Shack have all had computers fail miserably because of a lack of software support. Even the venerable "Big Blue" cannot seem to buck this trend. IBM's biggest selling point has always been, "There is more business software designed to run on the IBM PC than any other microcomputer". And that same selling point turned out to be out to be one of the greatest disappointments for the irate owners of the late PCjr. Because of the JR's design it really could not accommodate the existing IBM PC software base and as a new machine in and of itself could not muster enough of a software base to following in the wake of successful ancestor. This single factor, the ability for software to make or break a computer, is responsible for the tremendous surge in the IBM PC compatible market.

THE GOOD NEWS: It appears we have a software success story on our hands with the C-128. Of course this not by accident, and is mostly by design. After all, designing the C-128 to imitate its successful predecessor, the C-64, and packaging it with a built-in CP/M option was simply no more than "corporate bet hedging" to insure that the C-128 was not going to meet the silicon graveyard at a tender age because of a lack of software. But perhaps the most encouraging sign is the tremendous amount of quality, inexpensive software being designed and released at such an early point in the C-128's existence designed for the C-128's native mode. Titles from: Digital Solutions, Precision Software, Batteries Included, Timeworks, Access software, CMS software, Abacus Software, Solid State Software and others are making C-128 owners look very smart for investing in their machines at this early juncture. Let's hope things keep growing!

Superscript 128 *by Randy Margolis*

Superscript 128 is a word processor from Precision Software, the same company which provides Superbase. This program continues the tradition of Precision: providing incredibly powerful software at a decent price. Like previous Precision products, this program takes some study to use effectively. Superscript 128 is a post formatted word processor written by the same people who gave us Easyscript, and it has much in common with that excellent Commodore 64 program. Easyscript users should make the transition to Superscript 128 fairly easily. However, Superscript 128 does so much more that it fully justifies the rewriting.

The most exciting thing about Superscript 128 is the command structure. Anyone who is familiar with Lotus 1-2-3 will notice a striking similarity here. When you press the F1 key, a string of options appears on the top line of the screen, one of which is highlighted. By moving the cursor, the highlight bar moves through the choices, while on the line below a short explanation of each choice appears. When you select a choice by hitting RETURN you are presented with a secondary menu relating to that choice. Many of the menus have several layers of sub menus which narrow the operation down until you get to what it is that you wish to do. For example, if you choose 'Document' from the main menu, the secondary menu presents you with these choices: Load, Replace, File, Insert, Name, Block, Append, Directory, Merge, Spell, Utility. You choose one of these and are presented with a third layer of choices; and that is how you navigate Superscript 128. Similar to Lotus, you may press the initial letter of the menu choices in lieu of moving

Superscript 128 Continued

the cursor, and therein lies the most intriguing feature of the program. If there is a series of commands which you use often, you may set up menu strings in the text which are invoked by two key presses, exactly like 'keyboard macros' in Lotus 1-2-3. For example, since I use two disk drives, I often want to switch from drive 8 to drive 9. From the menu I would have to select this sequence: Document, Utility, Unit, 9-unit to switch to drive 9. However, I set up the character sequence in my 'defaults' file (loaded at the beginning of each session), so that every time I press ESC and the semicolon key (my choice of keys) the above sequence is invoked and the system reads from drive 9 from then on. There is a buffer of 1024 character reserved for these command strings. This facility sets up some intriguing possibilities for document handling.

Many of the menu sequences are also duplicated with CONTROL keys, giving further choices to the user. For example, CONTROL-g moves to the end of text and is exactly the same as pressing F1, Go, End. Also, CONTROL-r duplicates the previous set of command keystrokes, so if you want to search your document for a string of text, you enter F1, Set, Search, Find; and the next time you want to search just press CONTROL-r and the program types in the F1 sequence for you. All this may seem bewildering to many people, but several hundred thousand users of Lotus prove that it is not all that difficult. Sure, it helps to have the manual handy, but that's what manuals are for.

Your text area consists of 726 lines of 80 column text, which works out to twelve full pages. Also there is a subsidiary text area containing 251 lines in which you may have a completely separate document and transfer blocks of text between the two.

Superscript 128 saves your text as sequential files, which enables transfer by modem without any conversion, and the files can be read with any sequential file reader. This does cause slower disk saves than programs which write Program files, however. The program fully supports all printers by use of the 'defaults' file which you place on all your data disks. This file contains your printer definition, many of which are supplied on the program disk, and are very easy to modify. Also supported in the defaults file is screen color upon start-up, the default margins, justification, etc., any of which can be changed in a document, but their inclusion in the defaults prevents you from having to enter them into the text if you want the preset settings.

Superscript 128 also has a built-in numeric calculator which allows you to place results into text, or just view them on the top line of the screen. This makes it very easy to create, for example, financial reports in which the word processor adds up columns and rows for you. This is the most powerful number handler I have seen in a word processing program.

The built-in spelling checker is very similar to Easy Spell, which I consider the finest of the spelling checkers for the Commodore 64. Since this is always resident, there is no separate spelling program to load, and you can verify the spelling any time you want. The program comes with a 30,000 word dictionary to which you can add many thousands of your own words.

If you own Superbase 128, you can load both programs into memory at once and transfer data directly from you database file into your document. This is extremely helpful for form letters, preventing the necessity of creating a separate file of your mailing list to merge with you document: Superscript 128 will pull the information directly from your Superbase file! Your text area is diminished to approximately 2 1/2 pages when Superbase is resident, however.

Superscript 128 also contains a superb backup utility which easily copies your data disk on either a single drive or between two disk drives. This utility is so good that I use it for all my 1571 backups, text or not. This is a resident utility so you do not have to leave Superscript 128 to use it.

There are many more features of an advanced nature such as the ability to have different printer offsets for alternating pages for bound documents, alternating headers and footers for the same purpose, and many, many more. All in all, you can see that Superscript 128 is a very complete program which I would not hesitate recommending for the C-128 user. It is probably the last word processor you will ever need!

Paperback Planner 128 *by Loren Lovhaug*

Paperback Planner 128 is an electronic spreadsheet and the second of the Paperback series for the C-128 from Digital Solutions. And like the first program of the series, Paperback Writer, it is an excellent piece of productivity software combining power, flexibility, and ease of use with affordability.

Paperback Planner includes everything most people would want in a professional spreadsheet program. Paperback Planner 128, like Paperback Writer 128 is designed for use on the C-128's 80 column screen, and takes full advantage of the

Paperback Planner 128 Continued

C-128's 128k of random access memory. The program, written totally in machine code resides in one 64K bank of memory leaving a full 64K for user data storage. This is perhaps Paperback Planner's greatest feat. Many spreadsheet programs offer professional quality and performance, but very few do it and still leave you this much memory for manipulating your data.

Paperback Planner offers the standard spreadsheet functions such as addition, multiplication, division, subtraction, exponentiation, vertical and horizontal sums, averages, counts, minimums, maximums and modulo division and much, much more. All of the trigonometric functions are implemented, including both inverse and hyperbolic functions. Logical operations and boolean comparisons as well as conditional statements with alternatives (IF...THEN...ELSE) are included. The user has the option to choose between three matrix evaluation schemes: horizontal, vertical, and smart/automatic evaluation (making complex spreadsheets with intricate loops a snap). Base ten and natural logarithms are included along with the ability to obtain a pseudo-random number. With all of these functions and more available it is very difficult to think of a business, personal, financial, scientific, or statistical application that Paperback Planner cannot accommodate.

In addition to being a fine number cruncher, Paperback Planner is also designed with the user in mind. The package includes an on-screen help window which guides the beginner through every step of the way. Additional help may be obtained from the program disk as long as it is in the disk drive (which is only necessary if you want additional help). Paperback Planner also has a unique feature which puts grid lines on the screen to help define the cell matrix. This feature is especially helpful to the first time spreadsheet user. Both the help window and the grid lines may be turned off to allow unadulterated viewing of the cell matrix. The manual provided with the package is concise, helpful, and does a good job explaining the various features of the program. Manipulation and formatting of cell formulas and labels is made very easy thanks to a generous set of commands which allow the movement, deletion, and copying of user definable ranges. All range operations may be relative or constant which is a big plus when replicating formulas. Another excellent feature allows labels to overlap cell boundaries making titles and headings much easier to create. The cell width of the spreadsheet is easily changed and can be made as small as two characters and as large as SEVENTY-EIGHT characters, allowing a great deal of flexibility in the design of spreadsheets; in addition, individual cell widths may be programmed when printing a hardcopy or printing a layout to disk for integration with a word processor (making Paperback Planner an ideal tool for designing tables for documents). Individual as well as global cell formats are easily obtainable including: percentage, currency, floating point, and fixed formats, not to mention left and right justification and centering for both number data as well as labels. To further enhance spreadsheet readability and security, Paperback Planner boasts the ability to create windows for constant or occasional viewing of various areas of the spreadsheet or for the protection of formulas and titles.

(This may be starting to get hard to believe but there is even more!) To make an already stunning package even more attractive, Digital Solutions also has incorporated a complete search and replace facility and the ability to sort both rows and columns of text or values. Paperback Planner can also read sequential data files created by other programs (like Superbase or any word processor or database) for incorporation into a spreadsheet. This feature is especially attractive to anyone who needs to create reports comprised of large computer generated lists. But stealing the show, (and destroying all hope for the competition) Paperback Planner has the ability to automatically print your data sideways (without an external package) and produce amazing sharp graphic representations (pie, bar, stacked bar, and line charts) BOTH on the screen in full color 640 X 200 resolution, and to any printer with the ability to print graphics. All of these graphics may be scaled to your liking upon printout.

To tell the truth I have tried very hard to find a negative aspect of Paperback Planner but there simply isn't any. Perhaps the only weak point this program has is that its cell matrix is 250 X 100, and therefore slightly stifling vertically, but it takes a very large application to test this limit, and you have the ability to link spreadsheets for printout both horizontally and from disk, so this point seems extremely trivial.

In conclusion, I wonder, if through all of my rantings here whether I have truly done justice to this fantastic program. I highly recommend this program to everyone regardless of their computer or spreadsheet experience, especially to those in school or in a business where extensive number crunching and report writing is required. Its power, in combination with its flexibility, and low cost (under $50) make it a true winner.

Machine Language Differences of the C-128 *by Jim Butterfield*

The new Commodore 128 has a side that is virtually identical to the earlier Commodore 64, and a second side that is expanded with new features. BASIC is generally upward-compatible in the new side, dubbed 'C-128' mode. The machine's

Machine Language Differences of the C-128 Continued

new interfaces increase its overall usefulness, of course. An area that was of particular interest to me was its compatibility on the machine language level, and that's what I'll talk about here.I have a book in print - Machine Language for the Commodore 64 and other Commodore Computers - that tries to cover the whole Commodore product range Most of the exercises will run on almost any Commodore computer - PET, CBM, VIC 20, Commodore 64, Plus/4 or Commodore 16. My objective here was to teach transportable programming skills, so that a reader could move, say, from VIC to 64 without needing to scrap much of what had been learned.

This generality led to some limitations. I couldn't deal with graphics or sound in any detail, since each machine has quite different mechanisms for achieving these effects. And I had to use care in choosing the area of memory that would be used to stage student exercises: addresses from 828 decimal and up were free on all machines.

The B-128 created special problems. The existence of a memory banking structure, the curious bank-selection mechanisms, the completely different memory maps, the non-extensible machine language monitor, and the need for special 'transfer sequences' - all of these took the B-128 out of the mainstream of Commodore machines. Spending too much time on special considerations for this 'orphan' machine would clutter the book, yet I didn't want to leave it completely out. The B-128 ended up largely as supplementary material in the appendices, with extra material on the disk-that-can-be-bought-with-the-book. And there was a mild suggestion that a beginner would find the B-128 a more difficult machine with which to learn.

Now comes the Commodore 128. Will it be a mainstream machine, so that the book immediately applies to it? Or will it be another offbeat machine like the B-128?

The answer is: a little of both. There will need to be a revision of the book - it should be ready in early 1986 - to accommodate changes that are largely trivial but might confuse the beginner.

I would be easy to 'cop out' and suggest that exercises should be done on the C-64 side: these will certainly run without problems. But, heck, the C-128 side is too good. It's got a dandy built-in machine language monitor, and lots more 'playing around' room in memory. It has some limitations, too; for example, it's hard to put programs in high memory.

Let's talk about the major difference areas.

The monitor

The built-in monitor is good stuff. Say MONITOR and you'll be there. Most of the new features are convenient for beginners. Want to know what the decimal value of hex C3B is? Just type $C3B and press RETURN. Or, to change 1000 decimal to hex (or binary, for that matter) just type + 1000. Sure, old hands know how to do the conversion on a calculator, but it's nice to have it right there. For that matter, you can use odd number systems such as decimal and binary at any time, in any command. If you're doing an assembly and want to load the X register with a value of 40 decimal, all you need to type is A 0B00 LDX +40. No need to remember that 40 is hex 28: it will change in front of your eyes.

Ranges for memory displays and disassemblies can be given in three ways: with two addresses, to display between two locations; with one address, to display a number of bytes starting at any location: or with no address, to continue from a previously displayed address. No scrolling, but you'll find that typing a single character and RETURN is easy going. Oh, and memory displays also show the values as ASCII characters.

The new monitor is convenient, so much that I've written a parallel one for the Commodore 64. It's in the TPUG library as Supermon+64, and there's also a version for VIC 20. My biggest objective, again, is compatibility. Users switching between C-64 and C-128 sides shouldn't need to remember which formats are needed by which monitor.

The work area

The common work area in earlier computers - from $033C to about $03F0 - is not available for C-128 mode. In fact, it's a sensitive area and you'll crash if you use it. (Sigh). It's easy to grump about this, but the area below $0400 (decimal 1024) is at a premium because it's the only part of memory that is never bank switched.

Put your practice programs in the new cassette buffer ($0B00 to $0BFF).
When you're ready to attach your machine language programs behind your BASIC program (this comes up around Chapter Six), you'll find there are less pitfalls since variables are not stored behind BASIC any more (they're in a different bank). There is a new pointer called End-of-Basic at addresses $1210/$1211, and you'll need to adjust that one. After that, BASIC and ML programs will save and load as a single unit.

Machine Language Differences of the C-128 Continued

Don't try to put a machine language program anywhere above $4000 (+ 16384) unless you're quite familiar with the architecture. And before giving a SYS to your machine language program, it's wise to command BANK 15.

Variables

Chapter Six suggests that it's often convenient to get the value of a BASIC variable by reading it directly from its location in memory. This turns out to be a new challenge on the C-128, since the variables are in a different memory bank from the program you are writing.

There's too much detail to give here, but here's the core of it. Read memory in another bank by the following sequence: Set up an indirect address somewhere in zero page, pointing at the address you want to read, with Y containing the appropriate offset; load the address of this indirect address into the A register; load the bank number into the X register; and call JSR $FF74. Upon return, A will contain the value. Store to another bank as follows: Set up an indirect address somewhere in zero page, pointing at the address to which you want to write, with Y containing the appropriate offset; put the address of this indirect address in location $02B9; load the bank number into the X register; an call JSR $FF77. Upon return, the contents of A will have been stored to the appropriate location.

Whew! That's quite a job, and I gave some thought to dropping the corresponding exercise from Chapter Six rather than calling for the reader to sweat through it. It finally went in - as part of an appendix - the material is too useful to leave out.

Other

Don't let me scare you. Most of the things that you know from machine language on other machines still apply. You GET with $FFE4, you OUTPUT with $FFD2, and you'll find yourself in a familiar environment. BASIC starts at quite a high location - $1C01 or decimal 7169 - but there's lots of memory available, so there shouldn't be any problem.

The 80-column screen is mapped quite differently to memory, but that shouldn't worry the average ML user - $FFD2 sends to it if it has been elected. There are special characters that you can print to do things like screen windowing or scrolling - these work just as well in machine language as in BASIC.

It's a nice machine, and a new challenge. You'll enjoy it.

CP/M Update *by Todd Madson*

The C-128 has the unique ability to run in three separate modes. One of the three is CP/M - Control Program for Microprocessors. Originally written in 1973 to test eight-inch Shugart disk drives for Intel, CP/M later gained popularity when bundled with small 8080 based microcomputer systems targeted at hobbyists. CP/M is a standardized operating system that runs on the 8080 and other compatible microprocessors (The C-128 has a Z80A running at 4 MHz).

The nice thing about CP/M is that programs that operate on one CP/M machine should theoretically operate on another, hence the large amount of public domain CP/M software operable on the 128.

Many commercially successful software packages such as WordStar and dBase II work flawlessly on the 128. Plus, there are hundreds of megabytes of Public Domain CP/M software available such as terminal emulators, word processors, and classic games such as Crowther and Woods "Adventure". Most of this software is available from public access bulletin boards, or for a minimal copying fee from CP/M user groups. In the following article, I'll discuss how you can get some of this wealth of software for your C-128 system.

Unfortunately, the first two releases of CP/M for the 128 did not include support of the user port. The use of a modem in CP/M is vital, if you're ever going to get any of that good software out there! Many users complained to Commodore through the Commodore Forum on CompuServe. Finally, work was started on a new version of the CP/M BIOS (Basic-Input-Output-System) that allowed modem support.

In the interim, complex programs were written to allow the downloading of CP/M files in 128 mode, and would either save directly to a CP/M formatted disk, or allow porting to a CP/M disk from a 1571 formatted disk. While these programs worked, they were very awkward and frustrating to use.

The new CP/M BIOS is finished. It allows full use of the modem port in CP/M, and some of the best terminal programs from the CP/M world are now functional on the 128.

CP/M Update Continued

Commodore will be shipping the new BIOS, along with the MEX terminal program to all registered users (be sure and send in the white DRI card that comes with your 128!) in the first quarter of 1986. If you're like me, and don't like to wait, you can get a pre-release test OS, including the MEX terminal emulator. All you have to do is have a CompuServe account. The files you need can be found on DL3 of PCS47. You will need either Mike McLawhorn's update to Bombs Away's "SmallTerm", which downloads directly to single sided 168K CP/M disks, or, you can use an XMODEM terminal program to download the files to a 1571 format disk and use Joe Crumley's "Automemory" program to transfer them over to your CP/M disks. There are other methods, but I have tested both and they work.

Filenames:

Compuserve name:	Rename to:
WSYS.BIN	NEWSYS.COM
EX128.BIN	MEX.COM
F.BIN	CONF.COM

Once transferred to your CP/M disks, type NEWSYS C. NEWSYS then reads the CPM+.SYS file on your CP/M disk. Then it patches the BIOS and saves the old version under the name of "OLDCPM.SYS". It then saves the new version. At this point, you're nearly ready. Press reset and the CP/M disk should begin booting. Note the disclaimer that this version (4DEC85) has not passed quality assurance and is a pre-release version for testing only. At this point, type CONF. Conf allows you to set various parameters such as baud, keyclick volume, keyboard feel, and turning the 40 column screen off to speed up processing.

Once finished, type MEX. You have now entered Modem EXecutive, a full-featured terminal program. It is advised that you download MEX.HLP to add to the diskette for the online help utility. The HLP file is 50K in length, and will aid you greatly in running MEX, as it is a rather sophisticated piece of software.

Converting C-64 Programs to the C-128 *by Bruce Jaeger*

Obviously I'm referring to converting your own or public-domain software: compiled or machine-code commercial software, while technically possible to convert without the source code, would be just too much work to be worth it. (Why bother? The 128's Commodore 64 compatibility, while not the advertised 100%, is still pretty darn good.)

I've been doing my converting in two steps. First, I just "make it work," which involves going through the 64 program and removing all of the obvious (and not-so-obvious) crash points. Then, I go through the program again and add new features made possible by the 128 (80 columns, larger arrays, graphics, etc.)

MAKING IT WORK:

You can be pretty darn sure that any SYS calls are going to send the computer into hyperspace, as will many (but by no means all!) of the POKEs you may find in a 64 program. Often, the Commodore 64 used a SYS call or a POKE for something that the 128's BASIC 7.0 can do directly, particularly when dealing with sprites and high-resolution graphics. You pretty much have to be able to figure out the intent of the original programmer (if it wasn't you) to get around these parts.

POKES that protect parts of the 64's Ram from BASIC can almost always be eliminated, instead of trying to convert them to the 128's zero-page addresses. Since the 128 stores its variables in Bank 1, high memory in Bank 0 is automatically "protected" unless the program itself grows too big.

You will have to look out for variable names in 64 programs that are now reserved variables or keywords in 128 mode, like DS, DS$, DO, etc. You'll know when you hit one that needs changing--the operating system will tell you!

I generally leave the sound routines alone, because they'll work the same on the 128 as on the 64. (Why fix it if it ain't broke?) However, if all the sound routine does is produce a "beep" tone, I'll often replace it with the 128's Control-G beep to save space and make the program a little tighter. It is, by the way, important that you insert a BANK 15 command before every 64-style sound routine, to ensure that you're not poking into the middle of your program!

I also don't change the disk commands unnecessarily. First, because the 64 version of the commands already work, and, more importantly, because many of the BASIC 7.0 disk commands aren't implemented properly on my early prototype 128!

MAKING IT BETTER:

The number of things I do to "improve" a program depends upon my creative mood at the time, as well as the use I think I'll have for the finished result. Here are some of the things I might do:

Converting C-64 Programs to the C-128 Continued

Eighty columns. I might do this not only for the obvious screen appearance reasons, but because it allows me to use FAST mode. (Except that my prototype doesn't support the FAST command, and I have to POKE $D030 to start it, and it doesn't allow me to use the disk, and... You get the idea!)

Use BLOAD instead of the silly C64 IF A=0 THEN A=1 : LOAD "FILENAME",8,1.

Take advantage of the extra memory, and DIMension larger arrays.

Use graphics and sounds freely, because it's so darned easy.

Use the KEY command to preset the function keys for often-used words. (Particularly useful in database programs.)

Eliminate (if any) the cumbersome print-formatting subroutines from the 64 program, and use the 128's PRINT USING command for attractive output.

I have found that usually the hardest part of converting a 64 program to the 128 is figuring out what the original programmer wanted to do, and them duplicating that with BASIC 7.0 commands. Once you've done that, adding all the other neat 128 bells and whistles is easy--and a lot of fun.

Sparrow's Slick Tips *by Sparrow James*

SLICK TIPS: The ART of taking your C-128 farther, faster. Each month we try to present techniques which will save you time, enhance your programs or applications, or simply amaze you and your friends! If you have a trick or special technique you have pioneered or just heard of, why not share it with the rest of the universe? Send your SLICK TIPS to our mailing address or leave E-mail to: Sparrow on Quantum Link.

00001001: ONE KEY SCRATCH REVISITED

Last month, we presented a tip that really saves time when purging old disk files, or reorganizing your disks. The idea is to define a function key in such a way that a disk scratch can be accomplished simply by the pressing of that key. Last month we came up with:

```
KEY 4, "SC[shift r]"+CHR$(27)+"K"+"[3 cursor lefts]"+chr$(27)+"@"+chr$(13)
```

Just list a DIRECTORY to the screen and position the cursor on the line at the leftmost position of the line at which the program you wish to scratch is listed and hit the function key, you then will be prompted whether or not you wish to scratch the file.

Well, Randy Margolis, one of our contributing authors, has built the "better mouse trap". His technique is shorter and easier to enter. It takes advantage of the C-128's built-in tabs. Simply define a function key as follows and use the same technique presented above:

```
KEY 4, "SC[shift r]"+"[3 control ls]"+chr$(27)+"@"+chr$(13)
```

This technique can be used to produce other "one key" disk commands such as one key DSAVEs and DLOADs, simply replace the abbreviation for the SCRATCH command with the abbreviation for the appropriate disk command.

Note: SC[shift r] is the abbreviation listed for the scratch command and "shift R" ought to produce a graphic character. The three cursor lefts, as well as the three control ls in Randy's version must be enclosed in quotes.

Source: The Master's affinity for keyboard macros.

00001010: DETECTING KEY PRESSES

Memory location 213 (decimal) in Bank 0 of the C-128's random access memory is used for storing a value which denotes the current key being pressed. (This is similar location 203 on the C-64) Unfortunately, these values are not the same as the poke or ASCii values. This location is extremely handy for checking to see which particular key is being pressed (except the shift, commodore, and alt keys which are special). Particularly so, because it uses different values to denote the upper [set of four] cursor keys and the lower [set of two next to the shift key] as well as separating the numeric keypad from the numeric keys on the standard keyboard. I use the following function key definition to determine what the values are for any given keypress:

```
KEY 8,"DO:PRINT PEEK (213):LOOP"+CHR$(13)
```

Sparrow's Slick Tips Continued

When KEY 8 is pressed in the above example the values for any key you press are scroll onto the screen. If no key is being pressed, the value will be 88. To exit this infinite DO loop simply hit the RUN/STOP key.

Source: Sparrow's search for the perfect input routine.

00001011: DYNAMIC KEYBOARD

One of the slickest tricks that I have ever seen dates back to the early PET days. The trick involves placing ASCii values into computers keyboard buffer to simulate the pressing of keys. With this technique you can make the computer send commands to itself, or even make programs append program lines to itself. Try this example:

```
10 FOR I= 0 TO 4:READ X:POKE 842+I,X:NEXT I
20 PRINT CHR$(147)
30 POKE 208,5
40 DATA 76,73,83,84,13
```

Locations 842-851 are reserved for the keyboard buffer. Location 208 stores the number of characters waiting in the keyboard buffer. Location 2592 holds the maximum size of the keyboard buffer. which ordinarily set at 10. but can be increased to 20 if need be. However, increasing the keyboard buffer to 20 and POKEing ASCii values into locations 852-861 will take away the use of your TAB key.

Source: Sparrow's attraction to programs which write programs.

00001100: DISABLING FUNCTION KEYS

Perhaps one of the best features of the C-128 is its programmable function keys. But there are times when it would be nice to temporarily disable and then later re-enable the function keys. This can be done with the following lines of BASIC code:

```
10 FOR I= 0 TO 7:F(I)=PEEK(4096+I):POKE 4096+I,0:NEXT I:REM DISABLE KEYS/STORE
20 FOR I= 0 TO 7:POKE 4096+I,F(I):NEXT I:REM RE-ENABLE KEYS FROM F(I)
```

Source: Sparrow's love for the GETKEY command, and idiot proof code.

00001101: QUICK SEQUENTIAL FILE READER

This little program really comes in handy when you need to quickly examine a sequential file created by a word processor or database, or downloaded as documentation from a bulletin board system. You might want to incorporate it into your programs as a subroutine:

```
10 INPUT"FILENAME";F$
20 DOPEN#2, (F$)
30 DO UNTIL ST:GET#2,A$:PRINT A$;:LOOP
40 DCLOSE
```

Note: The parenthesis around the variable in the DOPEN command is necessary. Also, you might want add a error TRAP in case of disk error such as the notorious: 62, FILE NOT FOUND error.

Source: By request of new 128 users on TC-128 Archive BBS

00001111: DOODLE PICS IN 128 MODE

Pictures created by the wonderful C-64 mode graphic utility Doodle can be loaded directly into the C-128's GRAPHIC area by this simple sequence:

```
GRAPHIC 1:BLOAD"DOODLE NAME",B0,P7168
```

Source: Bruce Jeager and his infamous prototype C-128

Rumor/Opinion/Mayhem *by Loren Lovhaug*

Rumors, Rumors, Rumors...this last month has been perhaps the most rumor filled month in the history of Commodore International. It seems I cannot pick up a computer magazine, walk into a computer store, or log onto a BBS without being bombarded by all sorts of rumors. Just in case you have not had your ear to the ground lately I will fill you in: Perhaps the hottest rumor in the Commodore world is talk of a 10 megabyte hard-drive for the C-128 priced somewhere in the neighborhood of $750.00, in response to Jack Tramiel of Atari who supposedly is going to produce a similar item for Atari's new love the 1040ST (whatever happened to the 520ST?). Another always hot item is the speculation about when the expansion RAM memory for the C-128 is going to be available and how it is going to be actually configured...I have it from a reliable source (snicker) that we are going to see the memory units before you see the April issue of Twin Cities 128, of course, the same guy has been telling me that we'll be seeing them in three weeks for the past three months...don't hold your breath! Of course Commodore's financial situation is always good gossip. Just log onto any Commodore BBS and you can hear anything from "they just bought the farm and are going Chapter 11" to "Commodore has just announced that they have merged with the Australian Sheep Herders Association (ASHA) in an effort to produce the world's first machine washable computer. (By the way, for those of you keeping track, neither of the above are true; although Commodore had a tough fiscal 1985, it looks pretty good for 1986, and the Sheep Herders walked out of negotiations.) And then of course there is that persistent scuttlebutt about that empty ROM socket... Well, keep your eyes open, your fingers at the keyboards and enjoy this month's Twin Cities 128!

Paperclip 128 *by Loren Lovhaug*

You can't improve upon perfection, or can you? This is the question many software developers must ask themselves when they decide whether or not to translate (I say upgrade) their C-64 software packages for the C-128. For many software companies this is a difficult decision because of the huge expenses associated with developing and marketing a new product, coupled with the realization that the C-128's C-64 mode gives them a viable "C-128" product without ever changing a thing. At this point I would like to state that in my humble opinion any software company which attempts to deceive the public by advertising their product as for the C-128 under the guise of C-64 mode compatibility without declaring its true colors deserves to have their marketing staff stoned in the street by angry C-16 and Plus/4 owners (a rowdy bunch, that really know what it means to be "shafted"). The truth is that the C-128's native mode abilities really deserve to be exploited, and those companies which sit on their laurels will find out very quickly that they will be buried by the competition.

With that rather lengthy introduction, I hereby set out with my review of the newly C-128 enhanced Paperclip word processor by Batteries Included: the undisputed champion word processor of the C-64 world. I expected a lot from this program: after all, its ancestor was my steadfast text cruncher in the barbaric days before the introduction of the C-128. And even thereafter I was one of the faithful who purchased a dongle/joystick port extender to insure that Paperclip 64 would operate on my C-128. But times have changed, and 1986 brings Batteries Included some extremely stiff competition in the Commodore word processor world from such heavy-weight pros as Digital Solution's Paperback Writer and Precision's Superscript. How does Paperclip 128 stack up against the aforementioned competition? Read on...

First off, with Paperclip 128 is a completely faithful reproduction of Paperclip 64. All of the excellent features of the "champ" have been ported over to the C-128 version, plus as with most C-64 programs upgraded for the C-128 Paperclip 128 takes advantage of the C-128's added memory, speed, keyboard improvements, and of course the 80 column screen. This makes Paperclip 128 at the very least one of the top five word processor's for the C-128, because simply by emulating the C-64 version of Paperclip it can literally do everything that anybody would need to do with a word processor (for more about what Paperclip 128 does so well and the kinds of items one should look for in a good word processor see "How to choose a word processor for your C-128" in this issue). But function and utility are not the only things which make a word processor great.

Style is a very important consideration here. All of the best word processors can do the necessities such as: block moves and copies, mail merge, insertion and deletion of phrases and paragraphs, and manipulation of margins and output formats. The key here is whether or not the method by which word processor operates enhances or inhibits the actual writing/composition process. After all, the whole intent of using a word processor is to free the user from the drudgery of manual typing and hand-written drafts. This is where Paperclip's liabilities begin to appear. In comparison with some of the other word processors on the market for the C-128, Paperclip 128 is somewhat clumsy to use. Paperclip 128 is a post-formatted word processor, and by virtue of that fact it is inherently difficult to use when making even subtle format changes, as you are required to switch to an alternate view mode to examine your document as it is going to actually appear on paper. To make matters even worse, Paperclip 128 forces you to begin viewing at the very start of the document and scroll sequentially towards the end of the document. This can be extremely time-consuming when working with larger files. It would have been much better if Paperclip either was a formatted word processor or allowed you to view directly the spot of text you are working on when entering view mode.

Paperclip Review Continued

Another thorn in Paperclip's side is its use of keyboard graphic characters embedded within the text to represent special printing functions such as underlining, italics, and boldface. These characters are very difficult to get used to and are extremely ambiguous to all but the most experienced Paperclip user. Additionally these characters, because they are actually embedded within the actual text itself, make it difficult to create charts and tables within the text without constantly consulting the cumbersome view mode. Lastly Batteries Included has seen fit to design Paperclip without the benefit of word-wrap in its text entry mode. Because of this, words at the end of lines often are split between actual screen lines and typographical errors become more frequent and go undetected frequently.

This is not to say that Paperclip 128 is not a good word processing choice, quite to the contrary, Paperclip 128 is very good indeed, it is simply more difficult to use than many of the other programs on the market for the C-128. In fact, Paperclip 128 in some areas soars to heights not even dreamed of by most word processors. Without a doubt Paperclip 128 has the best column manipulation of any word processor I have used on a Commodore computer. Paperclip 128 can accommodate the viewing of formats with extra wide documents, up to 320 characters in length. Also, the Spellchecker for Paperclip, while not one of the world's fastest, performs reliably. Paperclip also is one of the most versatile word processing programs as far as printer's are concerned. On the Paperclip 128 program disk there are a variety of different printer files which will enable Paperclip 128 to communicate with almost every dot matrix or daisy wheel printer ever brought to market. Paperclip's manual is also very good and is complete, with a nice set of Appendices and a full index.

Paperclip 128 is an excellent word processor that while complete, does have some idiosyncrasies which make it difficult to use. For the experienced Paperclip 64 user, Paperclip 128 is a natural choice because Paperclip 128 does it all, and since it is virtually the same as the C-64 version there is nothing new that needs to be learned. However, for user's unfamiliar with Paperclip, especially those who have never used a command based, post-formatted word processor before, I would suggest you look elsewhere, because for you there are programs of equal quality which will do the job without as much hassle.

Data Master 128 *by Avonelle Lovhaug*

Database programs come in all shapes and sizes. Some databases are extremely complex, while others are overly simplistic. Data Master 128, by Bouncing Dog Software, is a new database program that falls between those two extremes. This database is proof that a program does not have to be difficult and cumbersome to be good software.

All of the features of Data Master 128 are displayed in the main menu, which is easily accessible from about anywhere in the program. The cursor keys move around the menu choices, and the return key selects an item from the menu. All of the menus offer the user an option to exit the menu, which makes returning to the main menu easy.

Data Master 128 distinguishes between four types of fields: textual data, logical data, numeric data and monetary data. Textual data fields are generally things like names, addresses, or other words or phrases. Logical data fields consist of only 1 character and can only be one of two values: true or false. Numeric fields are used when the data needs to be calculated, or some form of arithmetic needs to be done. Monetary data fields also use numeric data, except that it is presented in reports with added dollar signs, and values expressed to two decimal places. Although a textual field can be used to express a date, a separate date field could be extremely useful. This is probably the only missing field type.

Creating a database is like holding a conversation with your computer. The program first prompts for a file name, which can be up to 12 characters long. (Just pressing return here, like many other places in the program, automatically aborts the operation.) Next, the program will ask for the name of the first field, its type (textual, logical, numeric or monetary) and its length (how many characters). After all of this information for the first field is given, the computer will display the size of the current record. Then the process will repeat for all the fields until a null entry for a new field name is given, at which point the process will end. After all of the fields have been defined, the user is given an opportunity to change any information about any of the fields, until the record is satisfactory. For the person who is constantly changing his/her mind, this is a great feature.

Entering and changing information within the database is equally as painless. To add a new record to the file, use the "add" option from the main menu. The program will prompt the user field by field for the data. Inappropriate data will not be accepted. For example, only a "t" or an "f" can be entered in a logical field. After all the fields for a particular record have been entered, the computer will offer the user a chance to change any information that may be incorrect. When the data is all right, the record is appended to the file. If the user decides that some of the information needs to be changed after the save takes place, then the update option is used to correct the data. This section will let you change any record in the file.

Data Master 128 Review Continued

Data Master 128 offers some other important features. In addition to allowing the user to list the file (or a subsection of the file), the program also can create labels and reports. Although a default format is provided for the label process, the user may also create his own label format, which is fairly simple with this software. Not only are the directions for this complete and uncomplicated, but the author has gone so far as to tell us approximately how many lines per inch most printers use. The user is also given the opportunity to save the label format for later use. Reports are also a handy way to display information in the database, and Data Master 128 takes advantage of this. The report option allows you to choose a title for the report (which is displayed on each page of the report), the column information for the report, and headers and footers. In addition, the user can tell the computer where to make page breaks, data breaks, and if the computer should summarize or average a group of data given. An extra help that both the label and report option include is the possibility to see the display on the screen before making a hard copy. This can save on printer ribbons, paper, and noise pollution in the long run.

Data Master 128 offers the ability to merge similar files, link files with the same record structure, index files for easier reference, and sort files by any field in ascending or descending order. It also includes a record saving option, which gives the user the power to "undelete". Records can be marked for deletion, but are still recorded until the file is packed.

One note: the ability to link files is the apparent substitute for the fact that the file lengths are limited to 1000 records. Granted, not everyone has more than 1000 records in their database, but for some businesses this limit, even with file linking, may cause a difficulty. A more irritating limit is the maximum record length, which is only 250 characters. In a database with several fields, this may seriously hinder the usability of the database by forcing shortened field lengths and abbreviated records. This might be something for future revisions to be corrected.

The manual that accompanies Data Master 128 is fairly well written and complete. Something the manual lacked was an index, or a table of contents, to ease digging for forgotten information. Although most of the time, the user could easily call up a help screen by pressing the "HELP" key, occasionally more information was needed that only the manual contained. An easy way to find that material would be an index.

The software was evidently written with the business market in mind, as Data Master 128 contains one other important feature: file protection. This gives you option of protecting your files from unwanted eyes through a password entry system. Although not every user will be interested in this, some might consider this an important part of this software.

Support for this program might be one final reason to purchase this software. Bouncing Dog Software has promised to answer all questions relating to its software. In addition, updates of the program will be made available to the registered user at a nominal fee. This support, combined with its ease of use and reasonable price, make Data Master 128 a good purchase for the C-128 user.

CP/M Update *by Todd Madson*

Within the C-128 System Guide, there is a card that shouts in big, blue letters "Would you like to do more with CP/M on your Commodore 128"? The card is for documentation and utility disks offered jointly by Commodore and Digital Research, Inc. For $19.95, you receive a user's guide, programmer's reference guide, and system guide, plus two utility disks. Although the offer states that the documentation will arrive within six to eight weeks, mine took slightly longer than this to appear. When the package did arrive, I was pleasantly surprised. It contained a massive, plastic bound book of some 500 plus pages. It also contained two utility disks, plus a DRI license agreement - the same form as enclosed with the C-128 disk package! The CP/M plus book is comprised of the three sections mentioned earlier. The user's guide contains command summaries, plus useful advice and information on how to back up disks, how to create a file, how to protect files, and so on. This section also tells you things that are not made clear in the C-128 system guide, such as executing multiple commands on one line, how to protect files, plus how to redirect input and output. It also has a section for each and every CP/M command, and a lengthy section on how to use ED, the CP/M 3 Context Editor. The programmer's guide section includes information on banked and nonbanked CP/M systems (The C-128 uses a banked system). This section goes into detailed explanations of the BDOS and BIOS (Basic Disk Operating System and Basic Input Output System, respectively), The BDOS system interface, BDOS function calls and programming examples. The last section of the guide is the system guide. This section includes the operating system review, the BIOS overview, functional specifications of the BIOS, and sample BIOS listings. There is also a section on system generation and actually debugging the BIOS. The two diskettes included in the package include programs and libraries for software development. You are given a macro-assembler (MAC), a symbolic instruction debugger (SID) and other programs that will allow you to create programs tailored to your needs.

CP/M Update Continued

The second of the two disks actually contains the source and object code of CP/M itself so you can create customized versions for yourself.

The CP/M documentation represents a very good value for the user who will want to spend time programming the machine in Z80 assembly language. For the user who is more comfortable with C-128 mode and Basic 7.0, only the user's guide would really be of value. DRI should have separated each manual into three smaller books, as the manual is quite bulky and cumbersome to use.

Choosing a Word Processor *by Loren Lovhaug*

Those of us who use computers every day both in the work place as well as at home sometimes tend to forget that there might be folks out there who have not yet become initiated into the fold. The things I take for granted and have come to expect from my computer may not seem obvious or even apparent to those of you who may be just now learning about the benefits computers have to offer. With this in mind we at Twin Cities 128 have decided to begin a series of articles aimed at helping the beginner wade through the myriad of hardware and software choices for your C-128, our first installment focuses on choosing a word processor.

It has been claimed by computer industry analysts that the single most popular use for personal computers is word processing. This is probably true given the amount of written text that flies around this planet and the pure drudgery that typing on a standard typewriter tends to be. By virtue of these facts choosing a good word processor may be one of the most important choices one can make for their computer system.

Choosing the right word processor is generally not that difficult, but often it involves more than simply walking up to the sales counter at your local software retailer and saying "Gimme a text cruncher". The first thing you need to do is identify what kind of writing you will be doing. Many people are not aware that this is even a factor in choosing the right word processor. But the needs of the individual preparing large documents for formal presentations are vastly different from those of the person who is writing letters to a friend or from the person who is preparing a large mass-mailing. For instance, people preparing large formal documents should really have a word processor that has an ample amount of document memory so that they can easily view and work with large documents without heavy disk accessing. On the other hand, individuals who write letters often want a word processor that facilitates quick correspondence, something that makes writing letters easy and hassle free. In addition, individuals writing business letters often want the ability to create form letters using data lists (mail lists).

As you can see, the type of correspondence you will be working with is very important to the type of word processor that you choose. But this must not be your only consideration. Often many people limit their capabilities by choosing a program with a sole objective in mind, when in fact they discover that they will use their word processor for a variety of tasks. So whatever kind of writing you may be doing, it is best to choose a word processor that is both easy to use and yet highly versatile. Below and on the next page you will find a listing of the most important features one should look for in a good word processor, along with a brief explanation of why these factors are important. Use this guide as a check list when evaluating a word processing package.

EDITING FEATURES:

Ninety percent of the time anyone spends using a word processor is taken up by text entry and editing of the text. Therefore it is crucial that the word processor you choose allows you to enter and edit your text easily. First, your word processor should let you move around the screen easily
in all four cursor directions. Secondly, you should be able to move about your document by increments of words or sentences, this saves a lot of time and makes proofreading easier. In addition your word processor should allow you to make much larger movements such as by display screens or to the beginning or end of a document, this is referred to as profound cursor movement. Another feature you should look for is the ability to insert, delete, move, and copy ranges of text, which makes manipulation of phrases and paragraphs a snap. The ability for the user to define tab stops, especially numeric tabs is another feature which can save a great deal of time when, for example, creating tables which include columns of numbers. Of course every word processor designed for the C-128 ought to take advantage of something as basic as the Caps lock key or the insert/delete key on the upper right edge of the keyboard (for local insertion or deletion). Something that is found only on the finest word processors, is forward delete (sometimes called edit delete) which allows the removal of characters which follow the cursor and is a big help when cleaning up your document during proofreading. Both global and local search and replace that is both automatic or manual (depending upon the the user's preference) makes it possible for you to automatically search for a given word or phrase in your text and replace it with another word or phrase of your choice. Wordwrap insures that words are never broken up between screen lines; this is an important feature in a word processor.

Choosing a Word Processor Continued

Text formatting and text enhancement abilities are other important considerations when purchasing a word processor. The capability to print in italics, underline, and boldface should be available and easy. Another nice feature is the power to define characters, or the ability to use foreign characters. This can be vital in math or science reports, as well as documents which require foreign characters. In addition to other obvious features such as margin and tab controls, the ability to control things which affect the way the text appears on the page, such as pitch control, forced paging, auto-page numbering, and line spacing are a must for any good word processor.

File manipulation is an important part of word processing, and should be easy to use. A disk directory should be available at the touch of one or two keys, and loading and saving should be allowed on the directory's screen. Also, the ability to scratch, merge and link files are nice features. Merging files can be extremely important for the production of a form letter, which is merged with a mail list file.

Some special features that a good word process might include could be the ability to sort numbers and text, so that the user didn't have to alphabetize his own list, for example. Also, simple arithmetic abilities can help the user interested in adding small tables to his document. Help screens an excellent aid to the new user. And the ability to have multiple documents in memory can add a lot of flexibility to a program.

Of course, there is an endless list of qualities a word processor might have to make it a good purchase, but these have been some rudimentary suggestions of what most or all word processors should approximate. The most important thing to remember when purchasing a word processor is that it must fit your needs, so before you spend any money, try to borrow or rent a copy to make sure it is what you need. And always make sure that the word processor supports you printer type, or at least lets you create your own printer file for your particular printer.

KEYFIG *by Randy Margolis*

I have talked with a lot of CP/M users lately and I haven't found anyone who has experimented with the built-in user definable keyboard. This is one of the most unique features of the Commodore CP/M+ system, and one of the most enjoyable to fool around with.

There, residing on your system disk (the one which came with your C-128) are files called KEYFIG.COM and KEYFIG.HLP. These utilities are included for the purpose of redefining the CP/M keyboard. Within the CP/M+.SYS file (the actual operating system) are definitions of every key on the keyboard. Many users are unaware that every single key, with three exceptions, is redefinable. This is accomplished with KEYFIG.

When you first run KEYFIG, you are asked if you want help. If you answer yes, the KEYFIG.HLP file must also be present on the disk. You are presented with a menu of 14 help subjects which you cursor through and press Return to select. You are then presented with a page of help about each function. The first time you use KEYFIG you should read each screen in order.

After choosing 'done help', you are asked whether you want to use the default key definitions or the ones in the CP/M+.SYS file. At this point they will be the same. Then choose 'edit a key' from the next menu. Every time you press a key, you will be presented with the four current definitions for that key. These are normal, shifted, Commodore shifted, and control. Some people aren't aware that the Commodore key acts like another type of shift lock toggle, giving a completely separate set of characters. Initially, the Commodore shift definitions are identical to the shifted values, but they don't have to be. Anyway, cursor down to the one you want to change (use the upper keyboard cursor keys) and press RETURN. You may then do one of six things: assign a single character, assign a string, assign a special function, assign a hex value, assign a color, or leave the key as currently defined.

To assign a character, you will be prompted to type the character to assign to this key, and it will be done. For example, if you wish you may redefine your keyboard to the 'Dvorak' system if you want to practice that alternative typing system. Assigning hex values and colors are fairly self explanatory, but the help file will give you more information. If you choose to assign a string, you will be given a list of 32 different strings to assign. Most of them just say 'F10' or some such thing. Don't worry, because you may then edit the string to anything you want. This could be extremely helpful if, after application programs begin to appear for Commodore CP/M, you want to assign command strings to the keys and store them in a special CP/M boot disk for that application. For example, if you get a word processor that requires the sequence 'CNTRL-W,CNTRL-J' to move to the end of the text, you can assign that string to a function key (or any other key) and not have to remember it again. Many of the non-letter keys are unassigned for their CONTROL value, so you have a lot of choices to put your strings in.

There are 16 'Special Functions' of which 7 are pre-defined, but which you can reassign to a key. For example,

KEYFIG Continued

pressing CNTRL-ENTER (in the numeric keypad) will gracefully exit CP/M by performing the special function 'BOOT C128'. This points out another use for this program: exploring the default values of the keyboard. I never would have known about the special functions if I had not started using this program. These are some helpful things to know. For example, CNTRL-/ gives you a backslash, CNTRL-. and CNTRL-; give you curly braces, etc.

After you're done with your editing session, choose the 'quit and save' option and you are asked what you want to do with your new keyboard. You can save it to the CP/M+.SYS file on the currently loaded disk, and when you next boot from that disk your keyboard will be defined accordingly. Please note that you must reboot for the definitions to take effect if you choose this option. The second option is to make these your current definitions. This option will immediately put your new assignments into effect for the rest of this session only. Thirdly, you may trash the whole session by choosing not to save the new keys. Before exiting you are asked if you want to do anything else, which gives you a chance to reenter the program. So, you may save the new character set to the system file, reenter the program and also save it as current definitions before quitting, since you are still using the new definitions until you exit the program totally.

So, if you enjoy fooling around with CP/M you will definitely have a lot of fun with KEYFIG, and what's more it's FREE and available NOW!

C-128 REM FUN *by John Kress*

This article is aimed at those who have an 80 column monitor attached to their C-128. Although many of the items listed will work on the 40 column screen some of them won't, this is due to the way the C-128 handles the 80 column screen. There is a way of altering REM lines in your BASIC programs to get some different results when they are displayed on the screen. In the past, one big draw back to creating these effects has been that you have to do this in one of two ways:

1) Enter your program, save it in the usual way and alter it via a machine language program that looks for REM statements and then performs the alteration. This is the easiest way but you are limited by the confines of the M.L. program as to what type of alterations are performed. Usually they print the portion following the REM in reverse video. This is okay for finding things in your BASIC program, such as important routines but not much more than that.

2) Save your program, then load your M.L. monitor and sort through the listing, altering the REM's the way you want and saving the altered version. This is the better of the two, yet it takes a bit of time and work, with the multiple saving and loading. But some of us have found this worthwhile, since we can do things like protecting our programs from prying eyes.

But now, with the C-128, we have a computer with a built-in monitor so we can eliminate the reloads and re-saves. To those that don't know how to go about this, this article is intended to educate you on the methods on altering your REM statements.

Since BASIC is a large machine language program that allows us to work and talk to the computer in much more common terms than the computer's native language (that being numbers), there are certain limitations as to what we are able to do. That is why when we enter a command like PRINT the computer issues a syntax error. Also there are limitations as to what we can do with certain commands and key words. When entering a REM statement the computer enters the characters following the REM as ASCII characters and not as key words or commands. When executing the program BASIC sees the key word REM and interprets it to mean that the following characters are not executable commands and ignores them. But we can go into the BASIC listing and change some of the characters and make them into executable statements during the listing process. Therein lies the difference. If you have ever loaded a machine language routine and then tried to list it you can see some interesting results, cursors flying all over, characters change colors, screen being cleared and the like. That is what can be done by changing some characters following a REM.

Some important things to do to make this article work for you properly are:

1) Turn off your computer and then turn it back on. This is so that the BASIC RAM is cleared out. Just hitting the reset button doesn't clear out BASIC Ram, it just resets the pointers and puts three successive bytes to zero at the beginning of BASIC Ram. The intent here is so that if a mistake is made in entering a value in the monitor, when returning back to BASIC and listing the altered program, the computer won't crash.

2) Make sure that you don't allocate the graphics area. Doing so causes the start of BASIC to be moved upward by 9K and the monitor addresses below won't be valid.

REM Fun Continued

Enter the short program below, taking care to get the proper spaces in entered into the listing.

```
10 REM::: ** REM GOES HERE ***
20 REM::: ** NEXT REM HERE ***
```

Now enter the monitor and type M1C00 1C30 <CR>: and you should see:

```
>01C00 00 1F 1C 0A 00 8F 3A 3A 3A 20 2A 2A 20 52 45 4D:.....::: ** REM
>01C10 20 47 4F 45 53 20 48 45 52 45 20 2A 2A 2A 00 3D: GOES HERE ***.=
>01C20 1C 14 00 8F 3A 3A 3A 20 2A 2A 20 4E 45 58 54 20:....::: ** NEXT
>01C30 52 45 4D 20 48 45 52 45 20 2A 2A 2A 00 00 00 00:REM HERE ***....
```

Now position the cursor over the first 3A in the line 1C00 and enter 0D then 12, followed by 0F. These are the chr$() equivalents of the carriage return, print reverse and a special one for 80 column users, which causes the following text to flash just like the cursor does. Now hit return and go to the last of the 2A's in the next line and enter 0D, another carriage return, and hit return.

Now cursor over to the first 3A in the line starting with 01C20, and enter 22 8D 91, these are the same as a quotation mark, followed by a shifted return and a cursor up.

Exit the monitor by typing X followed by a return and list your program.
You should see the following:
•• REM GOES HERE •• : this should be flashing and in reverse print

•• NEXT REM HERE •••

The flashing REM will only be seen on a 80 column monitor and you will notice that the third '•' is gone; we used that for the second carriage return. The 20 is gone from the second REM line because with the 22 hex, 34 decimal, we entered print mode and the following two bytes caused a return and a cursor up. You can even enter more print mode characters, such as clr/home to clear the screen and prevent anyone from viewing what the program contains. You could change colors, delete characters, or almost anything you want by entering the print mode.

One thing about print mode. You cannot enter the value 0F hex after the 22 hex print mode has been entered. I don't know why but these won't enter into the monitor, at least with my C-128 they don't. To enter these value enter it before you go into print mode, or cancel print mode with a 8D, shifted return.

To create some interesting rem's you can enter a few colons, and then go to the monitor and search for the successive 3A bytes and alter them. Using the values listed below you will get the results listed next to them. All values are listed in hex for ease of entry in the monitor.

```
00 = no listing of REM statement
02 = underline of REM
05 = print REM (and remainder of listing) in white
07 = bell, same as control G
09 = print REM at first tab location, successive 09 bytes cause further tabs
0D = return, same as a chr$(13) cancels any prior features
0E = lists in lower case
0F = Flashing REM statement, won't enter after print mode; 22 hex.
11 = cursor down the remainder of line
12 = prints REM in reverse characters
13 = home the cursor but does not clear screen
14 = delete or rub out prior character, use to rub out a portion or all with      multiple bytes of 14
1C = prints red
1D = same as cursor right
1E = prints green
1F = prints blue
```

Values above 128 decimal, in the non-print mode will list as their BASIC keywords. Values between 32 and 127 will list as the ASCII characters they represent, when in non-print mode. When in print mode values between 32 and 128 will act the same as if you entered : PRINT CHR$(X)

REM Fun Continued

This method should make altering your REM statements a little easier than before, and some of the possibilities will make for fun for you and total frustration for those that are trying to figure out how you did that. Try experimenting and see what you can come up with.

Sparrow's Slick Tips *by Sparrow James*

00010000: QUICK BASIC KEYWORDS

As you know, the computer doesn't store basic keywords in the format that we enter them, but turns them into a number. The basic keywords all have a value from 128 on up. To get a listing of the keywords enter a one line program on a cold started computer (128 of course): 10 REM

Then in direct mode enter the following:

```
FOR T=128 TO 255:POKE 7174,T:LIST:PRINT:FOR D=1 TO 100:NEXT:NEXT
```

What you will get is the listing of the keyword followed by the value for that keyword. To speed things up, if you want, leave out the second FOR, NEXT and list this to your printer for a hard copy. Source: John Kress

00010001: QUICK SEQUENTIAL FILE READER IMPROVED!

Last month trick number 00001101, presented a short program which displayed the contents of sequential disk files on the screen. Well, our resident slick tip improver, Randy Margolis, suggested the following modifications which take advantage of the BASIC 7.0's error handling routines:

```
10 TRAP 40
20 INPUT"FILENAME ";F$
30 DOPEN #2, (F$)
40 DO UNTIL ST:GET#2,A$:PRINT A$;:LOOP
50 PRINT DS;" ";DS$:DCLOSE
```

00010010: MULTIPLE SCRATCHES UNDER ONE OPERATION

This one sure comes in handy when cleaning up disks and removing unwanted files! It is a little known fact that multiple scratches can be accessed through the use of the only one SCRATCH command. The format for the multiple SCRATCH is as follows:

```
SCRATCH"program1, program2, program3"
```

There is not limit to the number of files that can be SCRATCHed in this manner provided that the 160 character line limit is not exceeded. Imagine just how powerful this technique can be when combined with Commodore's pattern matching and wild card options in Commodore Dos. Source: Quantum Link

00010011: 1541 MODE DISASTER

It appears there is a strange bug in the 1571 disk drive's operating system that can really cause havoc for those of us who do a lot of switching between the 1541 & 1571 modes on the 1571. Never, repeat NEVER, attempt to validate (COLLECT) a 1571 formatted disk while in 1541 mode on the 1571. It will cause a loss of any data residing on the back side of the disk. Curiously enough, this flaw does not occur in the normal 1541, only the 1571 when in 1541 mode. Source: INFO magazine

Rumor/Opinion/Mayhem *by Loren Lovhaug*

Last month in this column I spoke about the fantastic amount of rumors which were invading the Commodore world. Well this month instead of rumors I am privileged to write on realities. Read the next few paragraphs carefully and remember you heard it here first!

DEATH OF THE C-64!

Some claimed it would never happen, others said it would not come about in the near future. Ben Dunnington of INFO magazine said it would not happen before the death of Our Favorite Machine, but it is going to happen, unless something drastic happens to prevent it, the production of the venerable C-64 will be discontinued gor good this summer! It appears now that the C-128 has gained world-wide acceptance as the lower-end personal computer. Commodore feels comfortable enough with the idea of turning the reigns of the consumer market over to the C-128. Of course the legacy of the C-64 lives on...inside of "our machines" in C-64 mode. I suspect that we will continue to see quality products designed for the C-64 far into the future, given the huge established base of C-64 owners and the ability of the C-128 to run C-64 software. By the way, the "final nail" in the C-64's coffin had nothing to do with the machine itself; without C-16, Plus /4, and Vic-20 keyboards and in light of advances which make the C-128's keyboard easier to produce, it simply has become too costly for Commodore to manufacture C-64 keyboards. Along this same line, there is some talk of a re-packaged C-64 type machine using a C-128 style keyboard maybe in late 1986 if there is perceived demand for such a product.

C-128 PLUS!

With the C-128 breaking all sales expectations, Commodore is planning to bring out a new upgraded version of the C-128 called the C-128 PLUS! The new machine will be essentially the same C-128 we have come to know and love with only one internal difference, the PLUS will come with an additional 512K of memory built in and will retail for around $400.00. Meanwhile, the retail price of the C-128 is expected to drop to around $200.00 by this fall. There is also speculation that in light of Louis Wallace's Ultra-Hires developments with the 8563 chip the C-128 PLUS may include 640 X 200 or 640 X 400 graphics available from an upgrade of BASIC 7.0. All of this and an expected Amiga price cut is viewed as Commodore's healthy response to Uncle Jack's 1040ST blitz.

EXPANSION RAM NOW AVAILABLE

As of this writing the 256K and 512K expansion units have been reported available on the east coast, and are selling like hot cakes! And fueling even greater speculation about these wonder devices is the discovery of an empty ROM socket in each of these units! Now where have we seen that one before?....

Sid Vicious Bytes *by Doug VanOrnum*

A lot has been published about the SID 6581 chip, and undoubtedly many of you have done some programming involving its sound capabilities. No matter. What I want to do is explain how the SID chip is accessed in C-128 mode, using both BASIC 7.0 commands and POKE'ed locations.

SID (Sound Interface Device) a 28 pin chip has three independently programmable voices, four wave types, three filters, two ring modulators, and two analog to digital converters for paddle use and external signal processing.

For a single voice, once you specify a certain wave form (triangle, sawtooth, pulse, and white noise) you can change how it sounds by manipulating the envelope. The envelope determines the volume, tone, attack, decay, and so on. Here is a diagram of a typical sound envelope:

```
   k  d
  c    e
 a       cay  sustain
att                          release
```

The BASIC 7.0 sound commands are very powerful, and yet fairly simple to use. The use of these commands are explained quite adequately in the System Guide, but a quick summary of these commands may prove helpful.

BASIC 7.0 sound command summary:

VOL (0-15)
SOUND (voice, frequency, duration, direction, minimum freq, step, waveform, pulse width)
ENVELOPE (instrument number, attack, decay, sustain, release, waveform, pulsewidth)
TEMPO (1-255)
PLAY (string, "PLAY A$")
FILTER (frequency, lowpass, bandpass, highpass, resonance)

Sid Vicious Bytes Continued

Since it would be easy to fill the entire magazine with information on programming the SID chip, and a large part of this information can be found in your System Guide (section 7, page 129), I will provide instead a few sample programs. Pay close attention to the given values, and then change them. You will quickly see how changing only a few variables will alter the sound quite a bit.

```
5  REM *** BOOM ***
10 VOL 12: X=49151
20 DO UNTIL X=8192
30 X=X-1
40 SOUND 1,X,8192,0,1024,340,3,0
50 LOOP: END
```

```
5  REM *** PHASAR CANNON ***
10 V=15: VOL V: I=24570: J=16378: K=32767
20 DO UNTIL X=100
30 X=X+1: IF FL=1 THEN IF X/2=INT(X/2) THEN V=V-1
40 IF V=1 THEN FL=0
50 IF V=15 THEN F=1
60 IF FL=0 THEN IF X/2=INT(X/2) THEN V=V+1
70 SOUND 1,I,10,0,J,350,1: SOUND 2,I,10,1,J,350,1
80 SOUND 3,I,10,2,J,350,1
90 VOL V: LOOP: X=0: GOTO 12
```

More complex examples again can be found in your system guide. In a large composition, the PLAY command is generally used, and the strings are read from DATA statements.

Machine Language programming:
The location $D400 (54272) is where the first SID chip register is located. There are some sounds that cannot be produced by the BASIC 7.0 commands (those that include ring modulation and continuous envelope control), and accessing the SID chip directly gives you added flexibility. The chip's architecture can be summarized as follows: the lowest 21 registers are divided into three parts -- seven control registers to control each of the three voices. We will take a look at the first set of seven registers:

SID 6581 Architecture:

Frequency: 54272-54273 ($D400-$D401) These locations control the pitch of the tone produced by voice 1. The first register is the low byte, the second is the high.
Pulsewidth: 54274-54275 ($D402-$D403) Use these two registers only when you are using the Pulse (or Square) waveform. It determines the width (low-byte and high-byte) of the pulse, which changes its tone. It is entered as a twelve bit number.
Waveform,ON/OFF: 54276 ($D404) Bit 0 turns the voice on or off. Bit 1 selects synchronization, bit 2 selects ring modulation. Bit 3 is a test bit and is seldom used. Bits 4, 5, 6, and 7 select the waveform.
Envelope: 54277-54278 ($D405-$D406) The high nybble of location 54277 controls attack, and the low nybble controls decay. The high nybble of 54278 controls sustain, and the low nybble controls decay. See above diagram.

The same parameters are used in their respective locations for controlling the second voice (addresses 54279-54285, or $D407-$D40D), and the third voice (addresses 54286-54292, or $D40E-$D414).

Filter: 54293-54294 ($D415-$D416) This 11-bit number controls the cutoff frequency of the filters. Low-order byte uses bits 0-2. The frequency is determined by the formula F=(30*W*5.8) Hz., where W is the 11-bit number.
Resonance: 54295 ($D417) The three low bits select the voice to be filtered. The first four high bits of this register set the resonant output of the SID filter. This effect enhances the frequency spectrum of the sound output, and is especially noticeable on the Sawtooth waveform.
Filter Type & Volume: 54296 ($D418) The first four bits control the total volume. Bits 4,5, and 6 select the lowpass, bandpass, and highpass filter respectively. More than one filter can be turned on at the same time. An instrument can be reproduced very effectively by changing the filter frequency during a tone -- something that cannot be smoothly done in BASIC 7.0.
A/D Converter (Paddle Inputs): 54297-54298 ($D419-$D41A) These two registers can be used to read paddles (or any other form of potentiometer), or they can digitize an audio input. The input cannot exceed 5 volts dc.

Sid Vicious Bytes Continued

Noise Generator, Voice 3: 54299 ($D41B) This read-only address is used primarily as a random number generator. The generator is dependent on the frequency and waveform settings of voice three.
Envelope Generator: 54300 ($D41C) This register returns the volume of voice three as a number 0-255. This number can be used to modify the parameters of other voices, but you may want to disable voice three first by setting bit 7 of location 54296.

Sample program:

```
10 V1=54272: V2=54279: V3=54286: REM THREE VOICES
20 FL=54293: REM FILTER LOW
30 FH=54294: REM FILTER HIGH
40 FC=54295: REM FILTER CONTROL
50 VO=54296: REM VOLUME
100 POKE S1+5,0: POKE S1+6,240: REM ATTACK/DECAY V1
110 POKE FC,0: POKE VO,15: REM FILTER CONTROL & VOLUME
120 WF=32: GOSUB 200: REM SAWTOOTH
130 WF=64: GOSUB 200: REM SQUARE
140 WF=16: GOSUB 200: REM TRIANGLE
150 WF=128: GOSUB 200: REM NOISE
160 END
200 POKE V1,0: POKE V1+1,0: REM FREQUENCY
210 POKE V1+4,WF+1: REM TONE & WAVEFORM
220 FOR X=0 TO 255: FOR Y=0 TO 255 STEP 50
230 POKE V1,Y: POKE V1+1,X: NEXT Y,X
240 POKE V1+4,WF: REM TONE
250 RETURN
```

Frankly, the sound potential of the SID chip is rarely exercised by most commercially available programs and their demo songs. The same pitches and tones are heard over and over again, and that can get tiring. There are a few public domain sound programs available that are masterfully done, including "Swinth" and "Dr.Who", but for the most part, nobody will take the time necessary to get the "right" sound for their songs. The reason? Most people play their compositions through their TV monitor's little speaker, which is not typically blessed with too much fidelity. Realizing this, the sound editor doesn't concern himself so much with how the music sounds, as long as he gets the notes and chords right.

However, it is surprisingly easy to hook up the C-128 to a home stereo. All you need is an extension cable from your local audio or electronic supply store. Once you do this, you will notice a couple of things. First, the output from the chip is noisy (due to, I believe, a close proximity to the CPU), and secondly, the music will sound a little dry. There isn't anything you can do for the first, and little you can do for the second. If you get serious about the musical capabilities of the SID chip (it has been used on many albums in conjunction with other instruments), you will want to purchase an external delay or chorus box. This will help make the music very rich and full-sounding.

Music Composition Using BASIC 7.0 *by Bill Nicholson*

Most of us experienced a certain amount of musical training in school, somewhere between do-re-mi and a Ph.D in Music Composition and Conterpoint. Using the former as the standard, I hope to introduce you to computer music in this and several articles that will follow in future issues of TC-128.

I will assume that most of you know how to find your way around a music staff and know what the different notes look like (a whole note looks like donut, a half note looks like a donut on a hook, a quarter note like a dot on a hook, an eighth note like a quarter note with 1 flag, a sixteenth has 2 flags, etc.)

Below is a visual representation of the range of notes that can be played on the C-128.

That is 6 octaves, which is much more than most will ever use. To play a note sharp, place a "#" just prior to the letter name of the note. An example of this would be: "V103 Q#D" which is a D# above middle C. To play a note flat, precede the letter name of the note with a ".". This is useful as BASIC 7.0 doesn't differentiate between the various musical keys.

Music Composition Using BASIC 7.0 Continued

Here is a small sample program that will show you how to take larger and more complex programs and place them into BASIC 7.0:

```
10 REM ** MARY HAD A LITTLE LAMB **
20 TEMPO 6
30 PLAY "V1O4T0U8X0"
40 A$="V1O3QEQDQCQDQEQEHE"
50 B$="V1O3QDQDHD"
60 C$="V1O3QEQGHG"
70 REM ** SAME AS A$ **
80 D$="V1O3QDQDQEQDHC"
90 PLAY A$:PLAY B$:PLAY C$:PLAY A$:
   PLAY D$
```

Line 10 tells us what tune it is we are entering, 20 sets the tempo, and 30 sets up the parameters of the voice that will play the notes. Lines 40, 50, 60, and 80 all tell the computer which tones to play, and in which order. "V1" gives which of the 3 SID voices will be playing the notes. "O3" tells the octave range that the note falls into. And each letter preceded by a "H" or a "Q" denotes the rhythmic value of each note.

Line 70 tells you that the next measure is the same as the first. If a specific measure matches note-for-note, there is no point to re-entering it. By using the method of entering a string variable for an entire measure, you can merely re-play that string. And line 90 plays all of the strings in the sequence that you want. This means that you have to do a little searching when preparing a piece for entry.

If you are familiar and confident entering a single line of music, you can try your hand at entering up to three lines of notes. The primary problem that faces you is that all notes must be placed in the order that they occur in each measure, regardless of which voice they are to be played by.

Thus, the measure that follows would be entered as follows:

```
"V2O4H#CV1O4Q#GV3O2IBO3I#CO2IBV1O4I#GIAV3O2IAV1O4QBV2O4QDV3O2I#GIAV1O4
Q#GV2O3QBV3O2IBO3I#C"
```

Note that the first note of V2 is a half note, and has priority over the quarter note in V1 and the eighth note in V3.

CP/M Update *by Todd Madson*

Anyone who has used a Commodore 1571 disk drive can readily attest to its increased speed over its predecessor, the 1541. In fact, the 1571 is so fast, some neophytes to the Commodore world often wonder what the fuss about slow disk drives was all about. The increased rate of speed for the new drive is not due to any incredible, new discovery. Rather, it could be called utilization of existing technology. The C-128 and 1571 both use a new method of data transfer, appropriately dubbed by Commodore as "Fast Serial Mode". Fast Serial uses a previously unused wire on the serial bus cable known as "Service Request". The "Bus" can be thought of as a cable carrying a group of wires, each carrying various signals to route information to and from the computer. This line has been put to work by transferring a high speed clock signal along with the data.

The first CIA (Complex Interface Adapter) chip in the C-128 has assumed more responsibility for communications, which allows a whole byte to be sent at a time, instead of one bit at a time with the old "Slow Serial" protocol. The speed now increases from a lowly 300CPS to a swift 3000 CPS. CP/M mode is slightly faster at this point, with transfer speeds around 3500 CPS. Small to medium sized BASIC programs load in 3-10 seconds, and productivity packages load in 15-25 seconds, not 4 minutes as on the 1541. The speeds we are discussing are fairly fast now, between 16000-28000 baud. This is really quite fast, and probably why Commodore set these speeds as default within the machine. Of course, FAST mode speeds things up even more! Within the operating system of the machine, there is a set of commands that are referred to in hushed whispers as BURST mode. Conventional serial disk access will operate one byte at a time while burst mode sends whole BLOCKS (256 Bytes) at a time! Commodore sources have indicated transfer speeds in BURST mode at 15000 CPS or greater. This is 120,000 baud, FIFTY times as fast as a 1541 drive! The possibilities for use of the 1571 with BURST are especially appealing as published specs have indicated top speed of the 1571 at 41,360 CPS. This is approaching (or exceeding!) hard drive speed. This would make BURST ideal for loading in records of a massive database, as Superbase by Precision Software does. When in CP/M mode, and using disk formats other than Commodore's, the drive automatically goes into burst mode and uses the sector sizing option to select whether 128.

CP/M Update Continued

256, 512, or 1024 byte sectors should be used for data transfers. Since the drive is completely programmable, any disk format could be used, as long as it is compatible with CP/M. MS-DOS files could be read and written to, but a special controlling program would have to be written to accomplish this. More on BURST mode in a future article.

Von Ertwine, the person responsible for writing most of the CP/M operating system for the C-128 has released a utility into the public domain that is as useful as it is exciting. The utility, C1571.COM, disables sector verification on disk writes. What this all means is that whenever you save a file, regardless of the mode you're in, the drive will write a sector to disk, and immediately read it back in again to make sure the sector was written correctly. Unfortunately, this verification is not always needed, and often this process actually increases the time to save a file. C1571.COM to the rescue! I tried experimenting saving various files to disk with C1571 disabled, then enabled it and saved the same files to another disk. The chart below may surprise you, as the speed increases on the larger files were substantial.

Filename	Size	Save Time	Save Time (With C1571.COM)
MEX.HLP	50K	1:55	1:05
C1571.COM	2K	0:20	0:14
CONF.COM	4K	0:20	0:15
CONF.HLP	6K	0:24	0:16
NEWSYS.COM	28K	1:02	0:36

As you can see, C1571.COM is an impressive bit of coding, and a very useful tool, especially for those who transfer software via modem since the time for downloading software is decreased somewhat.

Speaking of Von Ertwine, his new version of CONF.COM has been released, along with new commands, and an online help utility, much like the one the MEX terminal emulator uses. The new commands make the utility very versatile, and some annoying bugs have been fixed. CONF now correctly passes parameters to MEX, for instance. Some of the new commands are CURSOR, which allows you to specify the type of cursor you'd like to use - from an solid underline, to a blinking block - the choice is yours. Also incorporated is DATE, which allows you to set the date and time from within CONF. Now you can eliminate DATE.COM from your CP/M disks and have some extra free space on them. The MAP command allows you to print both character sets currently in memory. This would be especially useful if you redefined the character set using a character editor and wanted to examine both sets. Also included is a PARITY command, that allows you to set Parity, Word length, and stop bits for communications purposes. POKE allows you to poke various locations in memory, and REPEAT allows you to increase the rate of key repeat, or eliminate it completely. This is another nice utility that makes using CP/M a breeze. Best of all, it's free.

Fleet System 3 *by Avonelle Lovhaug*

The shift from standard typewriting to word processing has achieved one main goal: improved and facilitated the drudgery of writing. As word processing becomes more and more common, new features will be added to even further aid the writer in his task. One new word processor for the C-128 which has taken the goal of improving writing one step further than most is Fleet System 3, by Professional Software Incorporated.

Fleet System 3 is the updated C-128 version of Fleet System 2 for the C-64. In addition to most of the standard features generally offered by word processors, Fleet System 3 also offers built-in mail merge capabilities, a 90,000 word spell checking dictionary, and an integrated thesaurus. The last ability, the thesaurus, makes Fleet System 3 a pioneer (hopefully) in a new wave of software which helps the writer improve on his own work.

Fleet System 3 is command driven; that is, commands rather than menus are used to manipulate text and move within the program. Formatting text (left and right margins, indentation, justification, spacing, etc.) is also accessed through the use of commands, such as "lm10", for left margin set at the 10th character from the left, etc. Most of these commands are identical to commands used in other popular word processors. Fleet System 3 performs all of the standard word processing features, such as the ability to insert and delete text; move, copy, and delete ranges of text; and underline, subscripts and superscripts, and boldface text.

When using the program, a status line appears at the top of the screen at all times. This line gives the user information about what area of the program is being utilized at the time. The five different modes that the status line relates information about are: numeric mode, shift mode, insert mode, function mode, and extra text mode. Numeric mode allows the user to set up numeric tabs for easy column and number manipulation, and will also help make

Fleet System 3 Continued

addition and subtraction calculations. Shift mode is a left over from Fleet System 2's method of creating a "CAP" lock on the C-64, and is obsolete on the C-128. Insert mode allows the user to insert text as you type, without typing over what was previously there. Function mode lets the user access the disk drive, set ranges, search and replace words, as well as other features. The last mode, extra text mode, allows the user to view and edit other files without losing the text in the main text area. Fleet System 3 also uses this space to display its help files, and disk directories.

The most important features of Fleet System 3 are its 90,000 word spell checker and its integrated thesaurus. Although many C-128 word processors offer a spell checker, none have offered as large a dictionary as this program does. Furthermore, to this reviewer's knowledge not one company has even tried to offer a thesaurus, a must for any person who frequently writes. Both the spell checker and the thesaurus are impressive, using little time to scan words and check them against their own independent files. Although after the spell scan has finished, the user must still go back and edit or ignore unrecognized words by the spell checker, since the dictionary is so large, there is very little besides typos that the spell checker doesn't recognize. Both sections of the program are easy to use, and a real aid to the writer.

As excited as I was about the spell checker and the thesaurus, I cannot whole-heartedly recommend this program for several reasons. First, some of the command language is either too cute or obtuse that it often makes understanding the purpose of the command complex. For instance, although the standard for many word processors (as well as other programs) is to use the control key and the commodore key to access program features, Fleet System 3 uses what they call the "FCN" key, either f1 or "Run/Stop". Another idiosyncrasy of this program is the use of the word "recall" for "load" and "store" for "save". This may seem slightly picky, but it becomes a real pain when you want to know quickly how to load a program from disk, and "load" isn't in the index. Finally, probably the biggest flaw in this software is the incredibly simple way it allows the user to mess up his text just by reading a disk directory. If the user has text in the main text area, he must be in the extra text area to pull up a disk directory, or the directory will read in right over his work, and destroy all of it. Why the software developers didn't make it impossible for this to happen is certainly beyond me, and I'm sure that many new users of this program will agree.

Since Fleet System 3 is an update to Fleet System 2, the manual reflects this, adding only a few free pages of inserts to the manual entitled "Fleet System 2". These inserts cover printer files and the thesaurus, as well as some hype about added memory, etc., available on the newer version. The manual isn't bad, but it is 5 1/2 inches high and 8 1/2 inches wide, making it an odd shape which rarely sits politely without falling off of a stand, or anything else. The one advantage to the size and shape is that is does fit on top of the C-128 above the keyboard rather well.

Fleet System 3 is a pretty good idea, that almost makes it as a great program. At $79.99, it is probably a little high priced for the power, especially considering some of its drawbacks. However, if you absolutely have got to have a spell checker and thesaurus, and you have the money, Fleet System 3 is not only a fairly good choice, its your only one.

Paperback Filer 128 *by Avonelle Lovhaug*

When software developers produce an excellent program, they are often under pressure to repeat their success. Digital Solutions superior word processor, Paperback Writer, placed a great deal of pressure on them to make the spreadsheet, Paperback Planner, equally successful. And now that the "Planner" has met such favorable reviews, the big question is "Will Paperback Filer, Digital Solutions recently released database program, meet the standards of the first two software pieces?"

Paperback Filer is easily recognized as part of the "Paperback Family" by its similar screens and control codes for handling disk accessing and other routines. The main menu offers the user four choices: Create/Modify a layout, Enter/Edit/Sort a database, Report/Print, and File Utilities. The user can also switch between 40 and 80 columns in this menu by just pressing the "40/80" key on the C-128.

In the create/modify module, the user can design his own database, or change an existing one. This is simply accomplished by moving the cursor to where the first field should be and pressing "f1", and a "U" shaped marker will appear on the screen marking the place where the field exists. To continue, merely press "f1" until the field is the length desired. Paperback Filer will allow 5 different field types: alphanumeric, numeric, logical, date or time. To choose a kind of data for a particular field, press "ESC", at which point the user will be prompted with several questions he should answer about each field. Default settings are also available. In addition to choosing the field type, the user can also pick right or left alignment of data within the field, a choice of null entries in certain fields, if certain data can be disallowed or allowed, and if all of the data should be converted to upper case. All

Paperback Filer Review Continued

these options give the user a substantial amount of flexibility and control over what kind of information is entered into the database. To return to the main menu from this and other modules simply press the commodore key, then x. The prompt will appear "Save layout?Y" If the user wishes to save the current layout to disk, press return; if not, press "n" and then return.

The enter/edit mode is where the database information is entered, and modified. First, the user must load a current file, which is automatically prompted. Type in the name of the file you wish to add to (if a database file has recently been accessed, that name appears on the screen now) and press return. Paperback Filer allows the user several methods to move between fields and between records, as well as ways to search for specific records. In addition, an added feature of Paperback Filer is the use of a memory buffer holding previously accessed records in memory for quicker access than waiting for a disk operation. The disadvantage of this is that if you have a power surge, you are more likely to lose more data, since the information isn't always being saved to disk immediately. However, it does save a significant amount of time and disk operations, but if you prefer to load and save records immediately, the memory buffer can be turned off. This module also offers two additional features: password protection and data encoding. The password protection available not only requires that the user must know a certain password before using the database, but also can allow only certain restrictions to continue. For instance, if you want only some people to be able to enter data, but not view it or delete it, then this is possible through the use of password privileges. The seven different privileges provided are view, add, change, delete, sort, modify, and print. Another super feature is the ability to encode your files so that only Paperback Filer can read the data. One final option in the enter and edit module is the ability to sort the records of each file. The sort option offers several criteria by which to sort and also offers one interesting side light: the option to physically sort the records to another permanent order. This is the first time this reviewer has seen this option in a database, and for the person who uses his data sorted the same way all of the time, this feature could save considerable time and headache.

In the report module, the user can create a format for his database which will print a useful report of his data. The report section allows the user to arrange the different fields he will be using in his report, and to add certain textual information to the report. The user can also add headers and footers to his report, and make calculations and decisions with several math statements also included. These math statements serve as a kind of mini-programming section to help determine which information should be printed in the report. For instance, conditional statements are used to determine the values of field data, to decide whether it should be a part of the report. These conditional statements use an if/else format which is explained in the manual. Also, formatting symbols allow the user to choose such things as if information will be centered, left or right aligned, or duplicated, if commas will be inserted in numerical data every three digits, if dollar signs are added, and the displaying of negative numbers. All of these items give the user maximum flexibility in choosing what kind of format is right for him. Finally, the report module offers several printing options, such as the number of copies, pitch size, pauses at the end of a page, number of columns, printing only certain records, whether or not to print subtotals, and creating a disk file.

The fourth module which offers file utilities can allow the user to back up or restructure his files. The file back up routines will allow more than one drive, and allow the option to copy all or only some of the files. Restructuring the database gives the user the ability to lengthen and shorten fields, as well as adding more fields or eliminating them. To the database user who has ever realized half-way through entering his data that he needed more than 8 characters for last name entries, this utility is irreplaceable.

Paperback Filer is not, unfortunately, the perfect database. This reviewer was unhappy with the fact that every time we switched between modules, we had to reload the database. This is probably the most significant flaw of the program. The manual, also left a lot to be desired as far as its explanations of some features go. It seemed as though when I needed a brief explanation, so that I didn't have to waste time on a lot of text, the manual took forever explaining a simple function. On the other hand, when it gave me table, the table was too simple and didn't cover enough. Since the program offers help screens, and is basically easy to use, most of this can be excused.

Paperback Filer is an excellent program for the C-128. It has reached the standards of its predecessor programs by being easy to use, and adding features unheard of in other programs, such as the memory buffer and the privilege password system. Paperback Filer is a worthwhile addition to your software collection.

1670 Modem *by Bill Nicholson*

This 300/1200 baud modem was the first peripheral that Commodore released with the C-128 computer (aside from the obvious 1902 monitor and 1571 disk drive). I have to admit it is my favorite and most used item. With this modem I connect myself to large nation-wide databases, as well as computer networks, such as Quantum Link and Viewtron, and talk about everything under the sun with locals on myriad Bulletin Boards in the area.

1670 Modem Review Continued

The modem is Hayes-compatible, which is among the most-used business protocols, and is extremely flexible and easy to use. Since it is a smart modem, all dialing and modem commands are not software or CPU driven, but are residence in the hardware. It also does not have to be physically set for baud rate, but automatically detects the speed and transfers data at that rate.

The negative side of all the advanced hardware is one major "flaw" resident: the modem has a default setting of auto-answer, which is irritating if you are using the computer while the modem is still in the user port, and your phone rings.

Another possible flaw I have found with my modem is the permanent setting of the internal speaker. It does say in the manual that the speaker may be switched off for the entire comm session, or left on for the same. Instead, the speaker is active through the dialing and until the carrier is detected and responded to. This feature is not appreciated by sleeping spouses at 3:00 a.m.

Design problems are very few. The modem is designed to be placed into the user port at the back of the machine, but there is some torquing as it hangs at an angle to the rest of the machine. Also, carrier and detect lights would have been convenient, and a better quality speaker as well. It is quite tinny.

In its favor, I can only say it is the easiest modem I have ever encountered for use with the C-128 (or the C-64, for that matter). Once the few Hayes commands are learned, it will auto-dial, auto-connect, and auto-answer. The $179 price tag (available nearly everywhere at some discount) might seem a bit much, since 300-only modems are being offered for free by Commodore and Quantum Link, but file transfers at 1200 pay for the modem in a very short time.

Mousetrap! A Rat Falls For A Mouse *by Loren Lovhaug*

I have never had a great fondness for mice. I have always felt I could go farther faster with a good keyboard then with an inverted trackball. But the 1350 mouse for the C-128 has changed my mind "somewhat" about mice, and their role in the microsphere.

My bias against mice stems from using a mouse on the Macintosh computer for a month long project I was doing last year. The mouse forced me to lift my hands from the keyboard (the place where I was doing the majority of my data entry and where I was quite accustomed to leaving my hands while working with the computer) to fumble with the mouse to access various menus and command sequences. In addition it seemed that no matter how I arranged my work area I never had the space necessary to manipulate the mouse effectively. I found myself constantly running into books, coke cans, disk boxes and whatever else would find its way onto my desk. In frustration I was forced either to shove half of my stuff onto the floor or lift the mouse off the desk and make many tiny "skirting" movements to get to my on screen destination. Lastly I found myself having to make too many mouse strokes to do things I could do on other computers with only a few quick commands which I could tap out almost instantly.

While it is true that mouse-driven applications are much easier for the beginner to learn and use because they are "user-friendly" and very seldom involve intense memorization of commands or concepts. I have found that the term "user-friendly" used in conjunction with "mouse-driven is often synonymous with slow, clumsy, inadequate, and too cute. For instance, Commodore's Jane 128 (by way of Arktronics) is all of the above, and, thankfully as of this writing is the only non-graphics program designed for specifically for the C-128 which utilizes the mouse.

But in spite of all of the above I really like the "Commie-rat" (my affectionate term for the 1350 mouse). I like the mouse for graphics, in fact (choke) I LOVE the mouse for graphics, and I dare say it is the best graphics tool ever invented for your C-128. The mouse makes programs like Doodle and Blazing paddles perform like professional design tools, giving you an immense amount of control and precision never before experienced with a mere joystick. After using the mouse for graphics you will never settle for anything less. I didn't, every drawing in this issue of Twin Cities 128 was done with the mouse, and I think they are better because of it.

The mouse itself is styled to match the exterior of the C-128 (though it works just fine with the C-64) and is slanted downward to back to front. I found this slant design to be much more natural and comfortable to work with than the flat design of the Macintosh's mouse. It also sports two buttons which gives it more flexibility than the traditional one-button design. The cord is a generous 4 feet (121.6 cm) and seems to be long enough even for left handers who have to drape the cord over the front edge of the keyboard since the game ports where the mouse connects to the C-128 is on the right edge of the machine. The mouse's roller mechanism rolls well and generates a very true signal, though I would recommend the purchase of a "mouse pad" to increase the traction and sensitivity of the mouse. The mouse will work with any software designed to work with a joystick in either port (though you can increase efficiency via ML by

1350 Mouse Review Continued

treating the mouse a little differently) and is very easy to program for using BASIC 7.0 (see Sparrow's Slick Tips in this issue). Generally this product is well designed and seems extremely durable.

There are some drawbacks to the mouse which I feel obliged to report. First the manual is practically non-existent. It is a folded 8 1/2" by 11" sheet of card stock, extolling the virtues of cleaning your mouse and one rather scant pin-out diagram and the phrase of the year: "Additional programming and technical information may be found in the Commodore 128 Programmer's Reference Guide." (You know, the second biggest vaporware product of the year next to the 1572 dual disk drive, snicker). And second, at around $50.00 it is a tad steep, but in my opinion well worth it if use Doodle, Print Shop, or any other Graphics program more than for the occasional greeting card.

Sparrow's Slick Tips *by Sparrow James*

Slick Tips: The ART of taking your C-128 farther, faster. Each month we try to present techniques which will save you time, enhance your programs or applications, or simply amaze you and your friends! If you have a trick or special technique you have pioneered or just heard of, why not share it with the rest of the universe? Send your SLICK TIPS to our mailing address.

00010100: SUBSTRING INSERTION

It is not mentioned in the system guide, but the MID$ function in BASIC 7.0 can be used as an argument in an assignment statement allowing you to insert strings into other strings quickly and efficiently. The following example shows how the string constant stored in the variable A$ can be inserted into the constant stored in B$.

```
10 A$="COMES"
20 B$="HERE       THE SUN"
30 MID$(B$,6,5)=A$
40 PRINT B$
```

SOURCE: Sparrow's desire to break all of BASIC's rules.

00010101: WINDOWS FROM ML

Several people on our BBS have asked how to simulate the Basic 7.0 Window command from assembly language. This can be done by passing the proper values to zero page locations 228-231 ($E4-$E7). These locations control the current text window as follows:

```
228    ($E4)    Value for bottom screen line (0-24)
229    ($E5)    Value for top screen line    (0-24)
230    ($E6)    Value for left corner        (0-39 or 0-79)
231    ($E7)    Value for right corner       (0-39 or 0-79)
```

SOURCE: A Sysop's desire to satisfy his users

00010110: TRAP YOUR SUBROUTINES!

BASIC 7.0's nifty error-trapping commands have an extra added feature that can be extremely useful, the RUN/STOP key generates an error that can be trapped and be used as an extra command in a menu or to escape from a graphics screen. The RUN/STOP key generates error number 30 in the reserved system variable ER.

SOURCE: Quantum Link

00010111: DISABLE DLOAD & DSAVE (As well as LOAD and SAVE)

You can disable the DSAVE and DLOAD commands from BASIC 7.0 using the following:

```
Disable DSAVE : POKE 818,180
Disable DLOAD : POKE 816,0
```

You can re-enable them using the following:

```
Re-enable DSAVE : POKE 818,78
Re-enable DLOAD : POKE 818,108
```

Rumor/Opinion/Mayhem *by Loren Lovhaug*

Over the past two months in this column I have reported on various rumors that have been floating around the Commodore world. This month we are featuring information sources for your C-128 so I am going to turn away from the rumor mill and log my official protests and various information-related atrocities in the Commodore 128 world. As for the rumors, we will just have to wait and see.

Complaint #1 - Commodore 128 Programmer's Reference Guide

The Commodore 128 personal computer has been on sale on the retail level for an entire nine months now and there is still no Programmer's Reference Guide. There is no acceptable excuse for this. My sources from Commodore have told me that the C-128 design and engineering staff proofed the final draft copies of this manual long ago, and that it is Bantam Books, the publisher's problem. Frankly, no matter who is ultimately to blame, this is Commodore's problem, because the longer it takes for this book to come out, the more credibility Commodore loses in the eyes of retailers and disgruntled C-128 programmers. Commodore ought to pressure Bantam to get the book out as soon as possible, if for no other reason that it is bound to spark a new round of peripheral sales and software development, both commercial and public domain.

Complaint #2 - RUN magazine's "unethical policies"

Strike One: First, in an effort to lure new C-128 owners into purchasing their magazine they have attached nice glossy pictures of the C-128 and its peripherals all over its front cover and articles, even on issues which have little or nothing to do with the C-128 per se. For instance, on their April 1986 cover they featured a C-128 pitched atop a house as a leader for a home security article (the picture was repeated on page 23 at the beginning of the article) which featured C-64 mode software only (by the way, in that entire issue there was only ONE feature article on the C-128). The same kind of thing was repeated in the March issue when they featured a C-128 on a leader picture for RUN Script 64, a purely C-64 mode word processor. And most recently in their May issue they pulled the same kind of bait and switch tactic by putting a picture of a brand new MPS-1000 printer (a C-128 peripheral which has sparked a great deal of curiosity from C-128 owners) on their cover while touting a "special printer review feature" that mentions absolutely nothing about the MPS-1000! This policy of placing articles which have nothing to do with C-128 (except for the coincidence of C-64 mode) next to pictures of C-128's and their peripherals is deceptive and insulting to their readers because without a doubt it is designed to deceive and take unfair advantage of naive C-128 owners who are starved for information about their new computers.

Strike Two: On page 26 of RUN's April issue Tim Walsh, one of RUN's editors, advises beginners to do the following: "...go to a computer store that sells Commodore equipment and ask a lot of questions. Ask for demonstrations of printers in conjunction with a variety of software products: and get prices. If the printer of your dreams is sold and serviced by the dealer you will probably be best off buying your printer there -- unless, of course, you can get the same model at a mass merchandiser at a better price." It seems incredible to me that an editor of the largest Commodore computer magazine in the country would advise beginners to take advantage of a good-hearted computer retailer by spending an hour or two playing information parasite while the sales person breaks their back to earn an almost non-existent mark-up, while the "customer" fully intends to take his/her business to the mass merchandiser who most certainly will have a lower price, but whose sales people probably won't know the difference between a computer printer and a food processor. I wonder how Mr. Walsh likes having his time wasted.

Strike Three: RUN announced on page 6 of its May issue that it would be devoting only a maximum of 25% of its editorial space per month to the C-128. I think adopting and announcing such a policy in light of the excellent sales record of the C-128 and in light of Run's desire to court C-128 users (see strike one above) is ludicrous. But at least we know now where NOT to turn for C-128 information.

Your Commodore 128 *by Bill Nicholson*

A quick look through the computer sections of local book stores would make one assume that there were a plethora of books available for all Commodore products, with the Amiga certainly taking first prize for the flashiest and most expensive. There are, however, quite a few available for your computer and mine, the C-128. We have all been waiting anxiously for the Programmer's Reference Guide, which Commodore seems to have relegated to the back burner, but some others have come out that make the waiting easier.

"Your Commodore 128", which surfaced about 2 months ago, was at first glance, a good replacement for the System Guide. However, upon getting it home and spending some serious time with it, I found that it seemed to have the same failings of earlier books: it seemed to use the prototype version that includes some features that the production model didn't have, while missing others that it did.

Your Commodore 128 Review Continued

My biggest complaint is the importance this book places on outdated peripherals. Namely, the Datasette recorder and the MPS 803 printer. Granted, both are still being manufactured and both are supported by the C-128, but no one uses them. (Actually, I shouldn't include the printer, as there may well be many out there with them). If there are any C-128 users that are using the datasette as their principle storage medium, they are keeping awfully quiet. The book does mention the 1571 drive as the drive designed to work with the 128, but the book leaves out the command set for a 1541 drive, or even using the 1571 in 1541 emulate mode. Obviously, the writer of this work spent most of his time getting old VIC-20 programs to run from his datasette and printing out graphics to his MPS 803.

The command set for the BASIC 7.0 is fairly complete, but it doesn't address my favorite feature of 7.0: the music and sid commands. The section devoted to music and graphics uses BASIC 2.0 commands almost completely, which shows again a lack of interest by the author. Some people must enjoy PEEKing and POKEing into all of those addresses, but I am not one of them.

This book does have a pretty extensive section on CP/M mode, which is interesting, but unfortunately I am not capable of reviewing this area. I followed most of the commands and sequences, and it gives a fairly good glossary of commands for CP/M, so I will give it the benefit of the doubt, and say it is adequate.

To make it as brief as possible, "Your Commodore 128", despite its art nouveau cover, lacks most of the things I want to go beyond the System Guide that comes with the machine. I want hints that make the machine work better for me, with the hardware I have. This book is too little, for too much money. Wait for the Programmer's Reference Guide.

C-128 Internals *by Jim Butterfield*

This book contains a great deal of information about the Commodore 128 that is not yet generally available. It contains a rich fund of technical data on the 128, together with comments on the C-64 mode as it fits together with the machine. The information isn't complete (perhaps it's too early to hope for that yet). There are some inaccuracies. And the material is heavily technical...it's not suitable for a beginning level programmer or user. Considerable machine language experience is called for in working through the book.

Hacker's heaven

The book has a heavy hardware/hacker/machine language content. You know that if a chip is mentioned, you'll get detailed pinout diagrams and register descriptions. If an interface is described, you will get a picture of the edge connector and possibly information on voltage levels. Good for service departments, but probably overkill for the programmer and user who would never think of diving in with a soldering iron. This kind of detail is featured prominently, and takes up a fair amount of space in the book; the reader who doesn't need it will end up browsing through to areas of more interest.

The book also spends a good deal of time and space on 'known' hardware and techniques. Even though the 40-column chip has only two new registers (at $D02F and $D030), the whole chip is discussed at length in Chapter 2. You'll get the story on sprites, split screens, character sets, screen colors, high-resolution, multicolor and extended color modes. None of this is new, and in this area the book cautions, "all of the following programs must be entered in the 64 mode". The two new registers are not discussed at this point. You'll find similar detailed discussions of 'known' chips in Chapters 3 and 4, which deal with the CIAs and with the SID chip.

If you're moving up from the Commodore 64, you might feel that you know all this stuff, and that you'd rather see the book focus specifically on new material. On the other hand, you might be pleased to see the book attempting to cover the entire range of the architecture.

Chapters 5 and 6 do break considerable new ground. The VDC - the Video Display Chip that drives the 80-column display - is outlined in detail and several of its idiosyncrasies are discussed. And the MMU - the Memory Management Unit that controls the architecture of the computer - is also outlined. Until the Commodore Programmer's Reference Guide becomes available, this book may be one of the best sources of information on these two chips.

Chapter 7 is titled 'Assembly Language Programming'. It largely discusses the Kernal routines, and how to create a 'boot' or autostart disk. It is the last 'narrative' chapter of the book... and yet the book is less than half finished.

C-128 Internals Review Continued

Chapter 8 is the bulkiest part of the book. It contains a detailed disassembly of Kernal ROM, with each line commented. I miss the BASIC ROMs, but they are 32K in size, and including them would have more than doubled the size of this book, which is already massive. To read such ROM listings, a user needs to have some expertise. The pieces of code fit together jigsaw fashion, and to understand one part the reader will need to know how other parts interrelate. The disassembly code contains some 'European' ROM programming that is not found in North American models of the Commodore 128.

The verdict
The book is relatively good, but is not completely free of errors. Part of the disassembly on page 387 is wrongly performed (instructions at F934, F936 and F94B are not shown correctly). The memory configuration table on page 145 has copied a typographical error from some early Commodore documentation: configuration 13 should reference RAM 0, not RAM 1 as shown.

This book is not an easy read. There's a lot of good information here, but you'll have to dig for it. There's an index, but it's of limited use. The writing style is often stuffy ("Those who think it would be boring to take a close look at this chip would be deceiving themselves.")

But there's good stuff in here. The book is out early, and contains a great deal of data not easily available from other sources. For the serious student - and you'd better be a serious student - it's a goldmine of good information.

C-128 Tricks and Tips *by Loren Lovhaug*

One indication of the popularity of a given computer is how much third party literature there is being published on the various aspects of a machine. By this gauge the C-128 is apparently doing very well. There are several books now in print and available for you to utilize with your C-128. The vast majority of these books are designed for the beginner and often encompass a BASIC 7.0 programming tutorial section and material outlining various aspects of the available hardware and software. Commodore 128 Tricks & Tips is a unique book which introduces and explains various programming techniques for enhancing BASIC 7.0 and machine language programs. This book however is definitely not for the beginner, nor is it for anyone who does not enjoy or have a general interest in advanced programming techniques. This book assumes from the outset that you have a thorough understanding of the BASIC language and at least a fundamental grasp of the internal architecture of your machine.

Some of the topics discussed include: Character set redefinition in both the 40 as well as the 80 column display modes, manipulation of the 8563 VDC registers and 80 column bit mapped screens, advanced screen windowing, real time clock design, techniques for copy protection, manipulation of the keyboard buffer and dynamic keyboard techniques, changing keyboard assignments and utilization of the function keys and other special keys, creating BASIC extensions and wedges, BANK switching and manipulation of the MMU registers, autostart routines and boot sector creating, and important zero page addresses and BASIC tokens. I found that all of these topics as well as those I did not mention were thoroughly covered in a professional manner.

Perhaps the only complaints that I have about C-128 Tricks & Tips is that there were a few obvious typos in some of the program listings and some programs which did not work exactly as printed, but in general the techniques discussed and example programs listed to amplify them were very good. I thought this book was very interesting and worthwhile to any programmer with a fair amount of experience who wants to dive a little deeper into his/her C-128.

Sparrow's Slick Tips *by Sparrow James*

00011000: PRINT AT on your C-128: While the C-128's rich BASIC 7.0 does not have a PRINT AT command in and of itself, many will be relieved to know that the CHAR command (page 242-243 in your System Guide) will allow you to place text at a particular position on a either a 40/80 column text screen or a 40 column bit-mapped graphics screen. However, there is a slight quirk when using the CHAR command with the C-128's unique split-screen/graphics and text window mixtures (GRAPHIC 2 and GRAPHIC 4 on the 40 column screen). The CHAR command when using GRAPHIC 2 or GRAPHIC 4 will only place text on the graphics bit-map portion of the screen. If you try to place characters at a certain position on the text portion of the screen with the CHAR command, the text will appear on the GRAPHICS screen in the specified position even if that portion of the graphics screen is not visible at the current time. (Of course you may adjust the the size of the graphics screen upward to allow the viewing of the text, but in the process you will decrease the size of your text window.) Here is a simple method to allow the placement of text on the text portion of a split graphics/text screen:

```
SYS 49176,0,Row,Column:PRINT"Your text"
```

Sparrow's Slick Tips Continued

This method will also work on a normal text screen without a graphics screen invoked, however, this method WILL NOT place text on a graphics screen. Note that the zero separating the memory address and your row values simply passes a zero to the .A register which is unused by this Kernal routine.

00011001: Getting a little more HELP: The HELP key on the top of the C-128's keyboard is a nice addition when debugging BASIC programs, but often it is nice to utilize the HELP key inside your own programs. In reality the HELP key is a really just another function key which can be programmed in the following manner, by POKEing the Ascii value of the text you wish to reprogram the HELP key with into memory locations 4168 - 4172.

```
10 FOR I= 4168 TO 4172
20 READ Q$:POKE I,ASC(Q$)
30 NEXT I
40 DATA T,C,1,2,8
```

Unfortunately, there is only five bytes available for HELP key definition. But often this is enough to create some kind queue for program direction. You may place an automatic return on the key by POKEing location 4172 with a value of 13, of course this will cost you one byte of your five.

00011010: Quick Character Set Copy: When designing custom character sets for your C-128 it is often helpful to copy the character definition table from BANK 14 ROM locations 53248 to 55295 into some portion of random access memory for modification (this prevents you from having to start from scratch by allowing you to retain the design of characters you may wish to keep the same as the built-in character set and modify only those you wish to change in your new character set). Most individuals implement this copying procedure by using a FOR..NEXT loop to first read (PEEK) the character definitions and then write the definitions to the desired RAM (POKE). This method works just fine and is fairly easy to understand, but it is extremely slow (It takes about 23 seconds even in FAST mode!). An easier and MUCH faster method for copying the character definition table is through the use of the BASIC 7.0 BLOAD and BSAVE commands especially if you own a 1571 disk drive. The following example copies the entire character set to locations 12992 to 15038 (a handy location for Ultra Hi-res character sets) in a few seconds!

```
10 BSAVE "CHARACTER ROM",B14,P53248 TO 55295
20 BLOAD "CHARACTER ROM",B0,P12992
```

To place a copy of the character table at any other RAM location simply change the address in line 20 to your desired location.

00011011: Italics on your C-128! By simply modifying your C-128's existing character set you can have italics displayed on the screen! The trick involves shifting the upper four rows of each character to the right giving them an italics-type slant. The following example is for the forty-column screen, but the same trick will work with the 80 column screen by applying the technique in lines 40 - 90 to the 8563 video RAM via the VDC registers. (Note: This trick also works in conjunction with Ultra Hi-res character sets as well!) In this example printing reversed characters will produce italic characters. (Note the cursor will be invisible.)

```
 10 FAST:POKE 2604,30:217,4:REM TELL C-128 TO LOOK FOR CHARACTERS AT 12288
 20 BSAVE "CHARACTER ROM",B14,P53248 TO 55295:REM SAVE NORMAL CHARACTERS TO DISK
 30 BLOAD "CHARACTER ROM",B0,P12288:REM COPY NORMAL CHARACTERS TO 12288
 40 A1=12288:A2=A1+1024:REM DEFINE START OF CHARACTERS AND REVERSE CHARACTERS
 50 FOR I= 0 TO 1023
 60 N=PEEK(A1+I):REM READ VALUE FOR CHARACTER ROW
 70 IF (I AND 4)=0 THEN N=N+2:ELSE 90:REM CHECK IF LINE IS ROWS 1-4
 80 POKE A2+I,N:REM SHIFT ROW
 90 NEXT I
100 SLOW:GRAPHIC 0:REM SELECT FORTY COLUMN SCREEN
110 PRINT "NORMAL CHARACTERS"
120 PRINT CHR$(9);"ITALIC CHARACTERS":REM TURN REVERSE ON TO PRINT ITALICS
```

00011100: Get those manuals & magazines in order! If you are like me you buy or subscribe to many computer publications because there is a wealth of information that can be found in them. Unfortunately, I have a nasty habit losing and forgetting things. Because of this, I spent hours searching for information I was unable to recall which manual or which magazine or where that publication was the information I required could be found. But no more, I have developed a system where each month I highlight all articles of interest to me and I list on mailing labels on

Sparrow's Slick Tips Continued

the front covers of the publication the article's name and its page number. I have also begun to do the same with my software and hardware manuals so that I can find important reference sections. Of course all of this may seem simple and perhaps even obvious, but the real slick tip here is: Take the time to do this, and you really will save time in the long run! (by the way putting this kind of information onto a database would not be a bad idea either!)

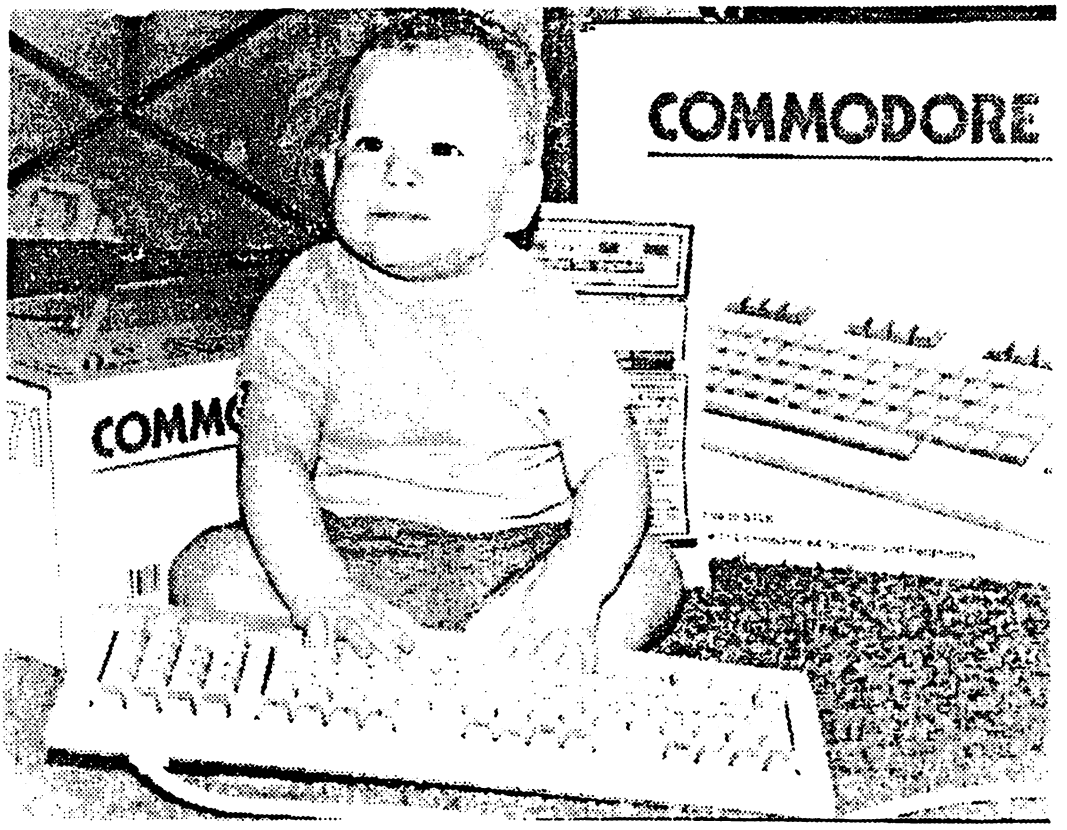
00011101: 80 Columns on a standard television set: As hard as it may seem to believe it is possible to display 80 column text and graphics on a standard television set, if you own a Video Cassette Recorder (VCR). First, what you need is some kind of 80 column monochrome which plugs into the 9 pin RGBI video connection on the back of you C-128 and plugs into your VCR via the Video-IN RCA plug connection. The VCR will take the monochrome composite signal and convert it to a T.V. compatible signal. This type of video output will not be nearly as sharp as that produced by a green or amber monochrome monitor or an RGBI color monitor, but it will be readable and very useful in a pinch or until you can afford a more suitable monitor for 80 column work.

00011110: SYSter C-128! It has never been so easy to pass values back and forth between a BASIC program and a machine language routine. The C-128 let's you pass values directly to the A, X, Y, and P registers through the use of the SYS command in BASIC 7.0 as follows: SYS address,A,X,Y,P The RREG command will do opposite by allowing you to pass the register values back to BASIC variables as follows: RREG A,X,Y,P For some reason the RREG command was omitted from the system guide, but it is working BASIC 7.0 command which can really come in handy when testing CPU status just after an ML routine.

00011111: C-128 Input prompt minus the "?": When using BASIC 7.0's INPUT command it is often desirable for aesthetic purposes to suppress the question mark that is displayed as part of the on-screen prompt. This can be done on the C-128 as follows:

```
10 POKE 21,64:INPUT "This is the prompt:";A$:POKE 21,0
```

When the above is executed the text inside the quotation marks after the INPUT command is displayed without a question mark. Note: The POKE 21,0 is crucial following each INPUT you use with this technique; failure to do so will cause your programs to run erratically.



Rumor/Opinion/Mayhem *by Loren Lovhaug*

The other day I was standing in a local bookstore browsing at the computer magazines. Standing next to me was a rather portly fellow in a three piece suit (I was wearing my usual T-shirt, baseball cap, jeans, and Hi-Tops) who was also gazing at the computer rags. As I paged through some of my favorites the rather portly gentleman spoke, "I can't believe how many magazines there are for those toy computers! I wish this bookstore would carry something for the serious user. Being somewhat angered by the insinuation that I was not a "serious" computer user I asked the gentleman what kind of computer he owned. He responded sharply, "An IBM PC, of course!" and then he added smugly, "If you ask me, there is nothing else for business!" I then proceeded to inquire what business he was in and he told me that he owned a small stationary store and gift shop and that he had gotten his computer to help him keep track of his inventory and for business correspondence. He also mentioned that the only thing he really regretted about his purchase was how much it cost, he added that it required a rather hefty loan. He said he had bought his system two months ago at the recommendation of a business associate. I asked him how much research he had put into the purchase and he replied, "Not much, that is why I bought IBM, because I figured that I could not go wrong with IBM." Well we talked some more and I began telling him about the C-128, at first he seemed very skeptical, but as I went on he seemed to become more accommodating. After about forty-five minutes I convinced him come over and look at my C-128, after all I said, "What do you have to lose, certainly not another $3500!" When he came over I showed him some the fine productivity software available for the C-128, such as Paperback Planner 128, Paperback Writer 128, Superbase 128, and he was impressed and dismayed at the same time. He realized that he could have satisfied his small business computing needs with the C-128 without having to shell out one-third of what he had spent on his IBM system. I told Richard that it may not be too late. After all, he had owned his IBM for only 30 days and it was quite possible that since it was still under the dealer's warranty it was likely he could return the system and get his money back. He said that he would consider it and get back to me. Now you may be wondering why I bothered to tell you this story. Well, the truth is I think there are a great deal of potential C-128 business users out there who through misinformation, sales hype, and lack of marketing on Commodore's or anyone else's part, are being lured into purchasing expensive computers that they do not really need. I am not trying to say that the C-128 is the ultimate computer for every business, because it does have some limitations, but for the vast majority of business computer users the C-128 could be an excellent low-cost alternative that should at least be considered. For many small business tasks the C-128 offers more than enough computer power and in comparison to the software and hardware prices associated with the Apple and IBM world, may for many make the difference as to whether to own a computer for business or simply be left wishing they could afford one. So, if you know of someone who is considering purchasing a computer for business, do them and you a favor by mentioning the C-128 as an alternative they ought to consider (if you like, you can mention this article and have them contact me through Twin Cities 128). After all, you more than likely will save them a great deal of money, and each and every C-128 that is sold only strengthens Commodore and the C-128 world.

Commodore's future: Over the past few months we have all been bombarded by speculation about what will and will not happen at Commodore. Here are a few are a few things that seem apparent to me:

1. Commodore will be marketing too many products. It is a good bet that Commodore will be marketing the following in the near future:

- A 3.5" 800k disk drive for the C-128 & C-64
- The 64c (the C-64 facelift I mentioned in April)
- The PC-10 and PC-20 IBM PC clones they are currently marketing in Europe
- Assorted Amiga attachments (memory expansions, GEN-LOCK interfaces, the IBM gadgets etc).

What this means is that Commodore could be marketing as many as seven machines: The 64c (C-64 with a facelift), the Plus/4, (Yes, Virginia they are still making them), the C-128 (Our hero), the PC-10 and the PC-20 (My clone sleeps alone), and the Amiga (Do you know somebody who owns one?), plus a ton of peripherals. Is this not the same company who was on the "brink of disaster" just a few months ago? They are really going to have to do some effective marketing if they are going to pull this off without dropping or cancelling a product. Of course, just because something is shown at CES does not mean it will necessarily be marketed (remember the 1572 and the LCD?).

2. Commodore has to market to specific groups of potential customers, and not just the mass market in general. For instance, Commodore might focus its advertising on promoting the C-128 as a small business computer or the Amiga as a desktop publishing machine. To do this effectively, Commodore has to much more than simply say these things in magazine ads or television spots. They have to cultivate third party software producers more effectively and develop much better local support networks through user groups, retailers, and VARs.

3. Strategies for Survival - If I were running the joint, what I would do:

Rumor/Opinion/Mayhem Continued
A. Dump the C-64 (or 64c) and the Plus/4 on the European markets and slowly remove them from the North American market. Europeans tend to have less expendable income than Americans which make the Plus/4 and C-64 much more attractive there. In addition, the market in Europe is far less competitive and the Europeans are much more accustomed to the idea of lower-end micros. In the United States, the C-128 could become the flagship of the consumer line at just a slightly lower retail price (say $200) and compete much better in the American market.

B. Abandon the PC-10 and PC-20 in the United States. The PC clone market is already far to crowded and with the violent price slashing that is taking place in that market it seems to be a losing a proposition for a new entry in the market.

C. The Amiga market has to be much better defined. So far nobody is really sure who the Amiga is designed to serve. I think Commodore ought to launch a campaign to popularize the machine, to bring it into the mainstream by flooding the television networks with Amigas being used for everything from rock videos to accounting to home security, and perhaps demonstrate multi-tasking on air, showing why this is a useful function of the Amiga Finally, continue the price cuts gradually until it makes no sense to buy anything else.

Superbase: The Book *by Avonelle Lovhaug*
If you are like me, you are a little awed by the manual for Superbase 128 by Precision Software. Although the manual is very organized, the power and flexibility of Superbase make it difficult to use for the person new to databases, and frustrating to the person who knows little or nothing about programming. However, Precision Software has now released a book to supplement the manual entitled: Superbase The Book. This book offers aid to both the new Superbase owner as well as the seasoned Superbase user.

The first four chapters guide the user through setting up a Superbase file, different kinds of fields and how to use them, methods to make your files more attractive and practical, options for data entry, and database searching, sorting and printing. The book assumes that the reader has read his manual, but it still provides ample information so that the reader must not constantly refer to the manual to find out what is going on. These chapters, although very useful to the beginner, are also of invaluable aid to the user who wasn't happy with his previous Superbase databases. Important information that was sketchy in the manual is made very clear in the book; for instance, although the manual states "there are times when duplicate keys are NOT a good idea" the book emphatically cries "do not go for duplicate keys -- this creates more problems than it solves." These chapters make some great suggestions on creating good databases, what kinds of fields to use for certain applications, and how to import and export records to and from sequential files.

Chapters 5 through 8 are titled the Automated Database, and they focus on helping the user write simple Superbase Programs, making more use of Superbase's menus, and using the report functions that it provides. Chapter 6 in particular emphasizes the need to plan and analyze your needs before jumping in and writing application programs. Chapter 7 then shows the reader how to use those plans to create programs that will significantly automate your database to make it easier on the Superbase user. The entire eighth chapter is devoted to helping the reader produce high quality reports. In addition to several tips on report generating, some sample programs are also included, along with an analysis of the methods used in the samples, and their strengths and weaknesses.

The last chapters of the book are mainly designed to aid the well-seasoned Superbase programmer put the finishing touches on programs to make them entirely automated, and also to suggest solutions to problems that often occur when using the program. Included in this is a complete troubleshooting list divided by type, with possible reasons for the error (sometimes the error name just isn't enough). This has proven invaluable in my experience with Superbase, and I find that it is the first place I look when a problem arises. The last chapters give some great ideas on creating menus to add to your Superbase programs to give them a very professional look: these also help the non-Superbase user to move around the program without causing any trouble.

I highly recommend this book to anyone who owns Superbase 128. Although not a replacement for the manual, this book offers such a wealth of information, and clarifies many of the Superbase functions and commands that it will make a significant addition to anyone's library. Also, this book may offer an opportunity for you to retrieve lost data from a corrupted file. For a mere $15.95 this book is worth every penny.

Blitz 128 *by Harold Shore*
If you like working with BASIC 7.0 but you want a product that runs faster and protects your source file, compilers offer a good solution. You can do all the difficult constructing and testing of your program using the rapid

Blitz 128 Review Continued

development cycle which an interpretive language makes possible - editing and almost instantly testing your program until it seems perfect, and then, when you are confident that you have achieved the program you desired, you can compile it so that it runs faster.

One of the first compilers to appear for BASIC 7.0 was Blitz 128 from Skyles Electric Works. I had already begun work on a major BASIC 7.0 program, and had been searching for such a product. When it was released around Thanksgiving of 1985, I was fortunate enough to obtain one of the very first copies of this compiler.

Unlike the earlier Blitz 64 compiler, Blitz 128 requires a plug-in key, or "dongle", to be used. This means that the disk itself is easily backed up, a major advantage for anyone preparing to use a product in a serious application.

Blitz 128 supports the entire BASIC 7.0 language. One's first uses of this program are likely to reveal some syntax errors that regular testing may have missed - the compiler reads the whole program file - not just the part it is trying to execute. So the very first attempts at compilation are likely to require that you fix the errors the compiler detects before you can run your program. But once this has been passed compilation is usually quite easy.

To begin a compile, you must boot the Blitz disk, and then answer some simple questions. First, attesting to the international scope of the Commodore community, it wants an indication of which of four languages it should use for its prompts and error messages. You are then given a menu through which you indicate the number of drives you are using and how certain BASIC 7.0 constructions are used in your program. After that, you should put a disk with your BASIC 7.0 program on it in the drive. BLITZ 128 will ask you for your program's name. Reply to this question and compilation will begin.

Compilation takes two passes over the text of you program. During the first pass, your program is checked for syntax and a symbol table is written to disk. On the second pass, a compiled form of your program is produced. The compiler's progress can be seen by the line number window near the top of the screen as it proceeds.

At the end, if all goes well, you will have two new files on the disk. The first is a file of linenumber/location references for you to use in the event of runtime errors in your program. By using this file, you can get some indication of where an error occurred. This file has the same name as your original program, preceded by 'z/'. The second new file on the disk is the compiled program itself. It has the same name as your original preceded by a 'c/'. It is this program that you may now load and run.

The compiled program will probably run quite a bit faster than your original BASIC program. How much faster depends upon the relative amounts of mathematical and string handling operations. But in any case, GOTOS, GOSUBS, and DO ... LOOP structures will execute much faster. The larger the program, the greater the relative speed increase.

You will not be able to list the compiled program. So keep the original BASIC program around in case you want to make any changes! But, by the same token, no one else will be able to discover or change the way you did things either.

How long does all this take? Depends upon the size of the original file. A 3 block source file will compile in under 4 minutes. A 225 block source file will take close to an hour to compile.

How large is the compiled file? Again, this reflects the size of the original file. A 1 block source file will grow by almost 43 blocks in compilation. A 70 block file will compile to code of about the same size. And our big 225 block friend will compile to a program of about 140 blocks. These compiled programs are stand-alone - they need no other program on the disk to support them.

As I indicated before, this compiler handles the entire language, and can deal with quite complicated loops and BEGIN: .. :BEND blocks. The program may contain GOTOs or GOSUBS to REM statements, a factor which aids in program organization.

Of course, there are certain restrictions. LOAD,SAVE,LIST, NEW and MONITOR commands are not allowed - but then these are inherently 'direct mode' commands. The same effect may be achieved by a SYS to the correct kernal address if it is absolutely necessary. The CHAR statement requires that a graphic buffer has been allocated, so positioning text must be done another way. (See Slick Tips in the May issue of TC-128 to find out how.)

One problem which I had in using this compiler is that it requires that the drive be in 1541 mode during compilation. It seems to lose its way during its second pass if this is not true. Since running all my compilations in SLOW mode on the 1571, I have avoided a repetition of this problem.

Blitz 128 Review Continued

Skyles sent me an upgrade disk in early January with some fixes for the compiler. This indicates a level of user support that should become more common in this industry.

But it is the MERCEDES of Commodore compilers in any case. It really works, even with very large programs. It is reliable, and produces code that is compact and fast. Although it is priced above other compilers on the market, it's performance justifies every penny of the difference.

Multiplan *by Harold Shore*

The new version of Multiplan by Epyx for both the C-128 provides the 128 owner with a classic spreadsheet whose performance at last lives up to its promise. This review will concentrate on the version for the 128 exclusively, as it is this environment that is required for Multiplan to function most effectively. The 80-column mode allows a good number of cells to be viewed at one time, and the FAST mode of operation allowed with that screen permit the program to execute with reasonable efficiency.

The basic unit of information in a spreadsheet is the cell. A cell may hold a value, some text, or a formula containing other values. Multiplan allows the user to build sheets of up to 63 by 255 cells. Motion around the sheet is accomplished by using the cursor keys, the "goto" command, or the "home" command. The monitor screen acts as a window which slides over the data as the user moves around. In addition, Multiplan allows the user to define up to 8 windows so that widely separated parts of the screen may be viewed simultaneously. It is especially easy to define a window which just includes data cells, thus leaving the row and column titles always visible. This makes data entry and interpretation much easier.

Despite the large size of the Multiplan sheet, the user may occasionally need to handle larger arrays of data. Multiplan provides for that need by allowing the user to define other spreadsheets which support or depend on the current sheet. This allows the user to link sheets so that a result derived in one may be used in another without the necessity of having to repeat the supporting data from which that result was derived. In this way very powerful computational matrices can be constructed.

Multiplan allows the user full control over how his data is to be presented. Individual columns or rows may be sized and formatted as the user needs.

But the heart of a spreadsheet is its ability to calculate values repeatedly based on changing inputs. It is in the area of calculation that Multiplan really shines.

First, Multiplan offers a very wide range of numeric, textual, and logical functions by which data can be related. Since textual data may be sorted and "located", as well as selected by these functions, Multiplan provides the rudiments of data handling within the spreadsheet environment. Logical relationships can control how the spreadsheet is recalculated, and even vary the data's presentation accordingly. Numeric calculations include functions appropriate to both financial and scientific work. In this way, Multiplan becomes a tool for a much wider range of purposes than the financial planning environments with which spreadsheets are associated.

Second, information may be entered into formulas as constants, absolute cell references, relative cell references, or by name. Although this might appear to the reader as almost too many options, in reality the variety of data entry modes makes using the sheet quite easy.

A final point that should be made about the operation of Multiplan concerns how it recalculates results. Normally, every time a value is entered, every value affected by it is recalculated. When I began to work with this program on small spreadsheets, I enjoyed this feature a lot. But as my sheets grew larger and larger, the time all this recalculation took began to be noticeable. Fortunately, Multiplan allows the user to set an option which controls whether recalculation is to take place with every cell that is changed. By turning recalculation off when I was entering large amounts of data, the whole process is greatly speeded up. After everything is in place, turning recalculation mode back on returns the analytical mode which makes such tools so valuable.

Certain types of calculation repeat themselves, using results from their last go around as the modified input to their next iteration. Interest calculations in business, or the classic Newton-Raphson method of finding square roots are examples of these. Typically, these calculation loops stop when a result has been found that does not differ significantly from the previous result. The result is said to converge upon a final value.

Multiplan Review Continued

Now Multiplan allows you to solve these sorts of problems without any algebraic manipulations. All that is necessary is to specify the order of calculations, (which will be circular), the final limit of significant change and perhaps an upper limit of the number of iterations. Functions are provided that make all of this rather simple. This greatly enlarges the range of problems that can be solved with this spreadsheet.

All this power takes time to learn, but Multiplan offers the novice user reasonable help in doing so. The manual which accompanies the program is divided into a guided hands-on tutorial section, and a reference section. The tutorial section is quite extensive and detailed -it's worth the time to work through it. The reference section is somewhat terse by comparison, the assumption seems to be that its reader is quite familiar with spreadsheet programming. But the gap between these two parts of the manual is made up, in part, by the inclusion, on the back of the distribution disk, of a number of sample spreadsheets. By loading, studying and modifying these, the user can be productive rapidly.

In addition, Multiplan is a well-known tool to users of other computers. It is available on the Apple, the IBM-PC, the Macintosh as well as others. For this reason, a small library of books has emerged dealing with using Multiplan for various purposes - a library which can be used effectively by the C-128 user when the manual provided fails to provide the needed assistance. The fact that the models developed on one machine may be used on another ensures that the work and data you have set up will continue to be useful even if you should change hardware. In addition, Multiplan provides a means of writing files suitable for transmission over modems, thus making the promise of machine migration a reality now.

I am familiar with many of the other spreadsheets offered for the C-128. Some provide capabilities not available with Multiplan, such as graphics or sideways printing. These would be handy, but I have learned to do without them. But Multiplan is powerful enough to help in planning a small business with all its variables. I know this, because I'm doing it now.

Mouse Modification *by Avonelle Lovhaug*

For many of us who purchased the Commodore 1350 mouse, it was a great disappointment to find that our two-button mouse was really a one-button mouse with two buttons. What do I mean by that? For those who don't own one, the 1350 mouse has two buttons, but only the circuitry on the left button is connected. Why, you may ask, did Commodore see fit to supply us with a mouse that was paralyzed on the right side? I wish I knew. It wasn't a cost saving measure, because the part needed to make the connection costs a whopping 7 cents. Perhaps they assumed that since no software exists in either C-128 mode or C-64 mode which would utilize such a feature, it wasn't worth the effort to finish the connection. However, Twin Cities 128, in its ongoing struggle to keep its readers in touch with the most up-to-date information, now presents to you: Mouse Surgery 101, or how to add a right brain to your mouse.

Now before you panic let me reassure you that this is not at all painful to your mouse, that it is not dangerous, and is a very simple procedure to perform. If you are not very good with your hands, you probably won't need to be, but to be on the safe side you may want to find a friend with a steady hand. And if you are worried about your warranty, you may not want to open your mouse until it has expired. However, these warnings should not ward off any but the most nervous, because the modification is very easy, and should not be any problem to perform.

First, locate the two screws at the base of the mouse where the connecting cable is attached to the mouse. Find yourself a small screwdriver and unscrew these two, then tip it over, lift the top up and push forward. This will remove the cover of the mouse to reveal the inside circuitry. Though most of the inside is fairly colorless, you will notice that there are several different colored wires located at the upper right side of the mouse (assume that the mouse cable is coming out of the top). If you look very closely at the black base that all of these wires are coming out of, you will see that each of the sockets the wires are plugged into are numbered 1 through 8 (the numbers are very tiny, etched in the plastic, but they are there). The wires that we are concerned with are located at number 6 and 8, or the blue and black wires. These are the two that must be connected in order for the right mouse button to work. To make the connection, buy a 330 ohm resistor. You can purchase them at Radio Shack, part number 271-1315. They come in a package of 5 and cost around $0.39. You only need one (or two in case you accidentally bend one wrong) so if you have a friend who has a mouse, you could go in on it together, if you think the price is too steep for you pocketbook. The ends of the resistor are quite a bit longer than you need them, and will need to be cut, but I used an ordinary scissors to cut mine, so don't worry about wire cutters, or anything. All you need to do to complete this modification is to bend the end wires of the resistor to create a U-shape, and then insert the ends into the small rectangular notches above the wire connections. No soldering is necessary, just push down until it is firmly in place. After you are comfortable with this, take the resistor back out again, and cut the ends so that the resistor still fits into the notches, but so that the cover of the mouse will fit over the top without bending the resistor.

Mouse Modification Continued

Before closing up your mouse, you will probably want to test it, so plug the mouse into joystick (mouse) port two (make sure the computer off). Then turn on your computer and type in this loop:

```
DO:PRINT (POT(3) AND 255):LOOP [Return]
```

This will create an endless stream of "0's"on your screen. To test if the right button is working, press the right contact button and hold it down. The numbers on your screen should change to 255, if all is well. When you release, the numbers should again be 0. You will also want to press the other button to make sure that nothing is amiss, and that the left button does not affect the program loop. If all is well, then your modification is complete. No soldering, and nothing very complicated. The only thing left to do is replace the cover on the mouse, and reinstall the screws. Make sure when you replace the cover that the resistor is not affected by the cover, or your connection may be lost. Congratulations, you now have a 2 button mouse.

CP/M User's Guide *by Randy Margolis*

Those of you who are ready to dig into CP/M on your C-128 now have a new alternative. Abacus Software has finally issued its long promised CP/M User's Guide. At a price tag of close to $20 the book carries the same cost as the supplementary package that is offered by Commodore. That volume is many hundred pages long and comes with two disks full of extra CP/M programs. You should know, however, that these disks contain programs strictly for creating Z-80 machine language files, and other very technical information. Also, the book, while excellent, is from Digital Research and is not really Commodore specific. With this in mind, should you forsake the Commodore offering and purchase this C-128 specific CP/M instead of Commodore's? There is much to be said for this argument. If you don't plan to get into Z-80 machine language for a while, I think the Abacus book is actually better.

It is fairly complete and begins on square one. It goes through each of the resident and transient commands that come with your CP/M system, and fully explains any C-128 specific features. An entire chapter is devoted to the nuances of the SET command, dealing with passwords and date stamping of files, which can be quite confusing. Chapter six is dedicated to explaining all the things you can do with the PIP program (I always rename it to COPY on my disks). After reading this narrative, I gained a new appreciation for this often maligned program.

Chapter seven is the real meat of the book, explaining all the little extras that Von Ertwine implemented when he configured CP/M Plus for the C-128. Virtual drive E:, 1571 disk formats, and the redefinable keyboard are covered in a forthright manner, as is changing screen colors and several other neat features.

The remainder of the book is an introduction to Z-80 assembly language. This section is not a complete tutorial, but just an introduction. If you want to learn to develop your own CP/M utilities, you'll need a more extensive volume than this. Next is 75 pages of Z-80 ROM listings. I fail to see why Abacus insists on wasting pages on this type of stuff. I'd like to find out if anyone ever uses it. Also, in my opinion there is far too much white space in this book. What's there is good; I wish they had filled the pages with more of it.

There is much of value in this volume. Is it worth the $20? If just for the Commodore specific information, and its clear beginner's introduction, I think it is.

Forgotten BASIC *by Loren Lovhaug*

Lately I have been examining much of the C-128 public domain software that I have downloaded from across the country, and I have noticed a disturbing trend. This is a lack of full exploitation of BASIC 7.0, or perhaps to be more accurate a lack of understanding of what BASIC 7.0 has to offer the programmer. When most programmers think of the advantages of programming in the C-128's BASIC 7.0 over the Commodore 64's BASIC 2.0 they almost automatically think of the advanced graphics and sound commands which allow very easy access to the machine's powerful capabilities. Unfortunately most programmer's forget that BASIC 7.0 also incorporates many other commands which really make programming the C-128 a joy! The goal of this article and the next articles in this series is to make you more aware of BASIC 7.0's forgotten gems, and present a few concepts and techniques which may prove valuable in your own programs.

DO look INSTR it!

Many television commercials ask the immortal question "Why pay more?" when trying to hype an upcoming sale. This question in an odd sort of way applies to programming on the C-128. Many of BASIC 7.0's structures will allow you to do more with a lot less code (and usually quite a bit faster and easier as well). For example, one of the most

Forgotten BASIC Continued

common situations that arises in application programming is the need for a user to select an item from a specific list of choices. In the "menu" scheme I have just described the key is to allow the user to easily select his/her choice while prohibiting the choice of an option that is not allowed. Most programmers who are used to working with BASIC 2.0 would probably solve this problem by displaying a list of choices on the screen using PRINT commands and use multiple conditional statements (IF..THENs) to to determine if the user's choice was valid. However, the solution using BASIC 7.0 is much more elegant. Consider the following example:

```
 10 DATA (P)izza,(S)teak,(F)ish,(O)ranges
 20 FOR I= 1 TO 4
 30    READ C$(I)
 40    CHAR 1,10,(10+I),C$(I)
 50 NEXT I
 60 CHAR 1,10,17,"Please enter your choice by"
 70 CHAR 1,10,18,"pressing the first letter of"
 80 CHAR 1,10,19,"choice."
 90 DO UNTIL C>0
100    GETKEY A$
110    C=INSTR("PSFO",A$)
120 LOOP
130 CHAR 1,10,21,"You chose:"
140 CHAR 1,31,21,A$
150 CHAR 1,10,22,"which is for:"
160 CHAR 1,35,22,C$(C)
```

The key to this particular example lies in the DO loop in lines 90 - 120. This loop runs until the numeric variable "C" becomes greater than zero. The numeric variable "C" is defined in this loop by the INSTR function which searches the first string quantity (the first argument in the parentheses) for the second string quantity (the second argument) and returns the starting position of the second string if it lies within the first (there is also an optional third argument which specifies the starting position of the search inside of the first string). This particular technique of combining a DO loop which terminates upon finding the position of a particular substring suits our menu/control problem very well because this technique effectively locks out any but the specified choices while making entry for the user almost effortless and easily understandable. Another thing to note is how the position of the particular substring can be combined with the string array variable "C$( )" to allow easy referencing of the data based upon the user's choice (of course the menu characters in the first argument of the INSTR function had to be placed in the proper sequence so that they would correspond with the menu data). Another aspect of the above example worth noting is the use of CHAR commands to place text on the screen. The CHAR command functions like a PRINT AT command in other BASICs and allows the programmer to place text at the exact position on the screen he/she desires. Those of you that are veteran C-64 programmers probably can recall the hours spent padding PRINT statements or writing roundabout formatting routines to output text exactly where you want it on the screen. Overall, I think you can appreciate the savings in both time and space BASIC 7.0 commands like INSTR, CHAR, and DO loops can offer. But the one other thing to note about the use of BASIC 7.0 structures over the more "primitive" structures of BASIC 2.0 is the style. The use of structures like the DO loop in this particular example make the program easier to read and understand. My alternative in this particular situation would have been a terminated loop involving the GOTO command and an IF..THEN statement which would have made this example harder to read and less "structured". (The current trend in BASIC programming is to move toward more structured, non-line number dependent code (ala Pascal, C, Comal, Modula2 etc.) and away from routines which involve the use of commands like GOTO which encourage programmers to write unorganized, inefficient, "spaghetti" code.

Got to GETKEY some SLEEP

There are many instances when it is necessary to pause or slow down the execution of a program. Two most often used BASIC 2.0 techniques for accomplishing this are using an empty programmed loop (FOR..NEXT) or halting a program until a key is pressed (10 GET A$:IF A$="" THEN 10). BASIC 7.0 makes this much easier to do. To delay program execution for a specific amount of time you can use the SLEEP command. The SLEEP command will halt program execution for a specified number of seconds, for instance SLEEP 10 will cause normal program flow to be halted for 10 seconds. If you want to halt program flow until a key is pressed, that too can be accomplished much easier from BASIC 7.0 through the use of the GETKEY command which halts program execution until a key is pressed and will define the specified variable according to the key that was depressed. The GETKEY command can also be used in combination with other commands to serve in a very powerful input routine as you may have noticed in the above example.

Sparrow's Slick Tips *by Sparrow James*

00100000: The Master has roasted Sparrow again! Our first Slick Tip this month comes from the Master. The Master delights in finding Sparrow's snafus and correcting them, for which we are thankful: There is an incorrect statement in the second of last month's Slick Tips concerning reprogramming the HELP key. Contrary to the Sparrow's tweet, you may have more than five bytes assigned to this key. The string length is stored at location 4105. The following little program should do the job automatically:

```
10 INPUT"New text for help key";Q$
15 PRINT"add a <return> ? (y/n) "
16 GETKEY A$:IF A$="y" then Q$=Q$+chr$(13):X=LEN(Q$):GOTO 20
17 IF A$="n" THEN X=LEN(Q$):ELSE 15
20 POKE 4105,x:i=0
30 DO UNTIL I=X
40 POKE 4168+I,ASC(MID$(Q$,I+1,1))
45 I=I+1
50 LOOP
```

This program will only work if you have not redefined any of the other keys, because the location of the actual string for the HELP key will be pushed out of location 4168 if you change any of the other key string lengths.

00100001: C-128 Crash Helmet: This one comes from Harold Shore of Bouncing Dog Software: It seems there is a strange quirk about the C-128 that allows you to magically recover from most deadly BASIC or machine language crashes. Try this, type in a short program in BASIC or machine language, then hold the RUN/STOP key down and press the reset button on the right side of C-128 and then let go of both. You will note that when the computer is finished resetting itself you will be inside the machine language monitor. From this point type "X" to return to BASIC or simply examine your program from the monitor. Theoretically your program ought to be history, but surprise, you find it is there safe and sound! This technique is very helpful when developing routines which may alter many crucial ML vectors, since it will allow you to reset from a crash with out losing your code.

00100010: Superbase Programming Tip From Superbase Sam: During Superbase program development the programmer may find him/herself continuously having to jump from Superbase's main menu (where you end up after a program error) and Superbase's program editor. This can be tedious especially when the bugs are attacking in droves! But like just about everything with Superbase even this can automated. Simply place the PROG command on the last line of your program and then GOTO it from the command line or simply type PROG from the command line. To automate this process even further include a line number with the PROG command such as: PROG 500, and your Superbase program will be listed beginning with that line number upon entry of the program editor.

00100011: Superbase Programming Tip II, Another one from Superbase Sam: The left arrow key (the one just above the CONTROL key on your keyboard) will allow you to jump backwards to the ends of Superbase program lines inside of the program editor. Try it, it will save you hours of scrolling with the cursor keys!

00100100: Unresolved Reference Referral From Morgan P. Spendthrift: BASIC 7.0's RENUMBER command is certainly wondrous, and at times a real lifesaver. But there are times when its strict adherence to the rules can really cramp your style. For instance while writing BASIC code you might construct a ON GOTO or ON GOSUB branch with line numbers that do not as of yet exist. If you try to RENUMBER that ON GOSUB or ON GOTO line you will get an Unresolved Reference error because the system could not reference a line that does not exist for renumbering. One way to get around this, is to simply place a REM in front of the offending line and the RENUMBER facility will ignore it.

00100101: Destroying The Dimensions From Darren: From time to time it is handy to be able to clear out all array variables (perhaps even redimension them) while retaining the values of all non-array variables. The CLR command will not do the trick because while it will clear out all array variables and allow redimensioning it will also destroy all non-array variables as well. Fortunately this is possible on your C-128; examine this example, especially line 100:

```
  1 DIM A$(50)
  2 A$(50)="HELLO":B$="MELVIN"
100 POKE 51,PEEK(49):POKE 52,PEEK(50)
101 DIM A$(60):A$(60)="WOW"
102 PRINT A$(50),A$(60),B$
```

Rumor/Opinion/Mayhem *by Avonelle Lovhaug*

Computer Widows - if your husband, boyfriend, or any friend is a computer enthusiast, you are very familiar with this term. Signs of being a computer widow include: your partner numbering in hexadecimal or binary, his voice sounding like a modem carrier signal on a phone line, or worse yet, conversations held between the two of you only if you leave messages for him on his favorite board. One thing that really exacerbates the problem of computer widowdom is that many of us married to computer addicts cannot possibly see what draws our partners to their machines. "What does it have that I don't?" we cry. Jealousy is very common for we computer widows. We threaten to fold up their diskettes, bury their disk drives, watch "All My Children" on their monitors, and hide their computers in the garage, hoping the suckers will disappear with someone else's husband at your next garage sale. The problem is especially pronounced in the spouses of C-128 owners, because the old "we can't afford that new..." just doesn't work all of the time, since C-128 software and hardware is often very affordable. Even if they aren't spending any more money on the computer (how come the computer gets expensive presents and we don't?), it doesn't cost much once someone has a modem to get on every local board and stare at the screen for hours, downloading as if there would soon be a public domain shortage, and free programs would be a thing of the past. These first two stages of computer widowdom (buying up and downloading) are followed by a third stage: invasion of the computer lovers. They will infiltrate your house, and brainwash your children. Keep a particularly close eye on your junk food, your peripherals, and especially disks and manuals. These people are not to be trusted. You may think me exaggerating, but consider this story: Mrs. Johnson was a normal wife with a normal family until her husband bought a C-128. She survived the first two stages, despite the fact that they had to take a second mortgage out on the house, and their phone bills were so high she had to hire out her oldest son to sell telephone books to people without phones. However, when stage 3 hit, her life fell apart. The entire house became the work station for ten computers and their owners, who's own families had already kicked them out due to the extreme nature of their condition. She tried to communicate with them, but unfortunately, they had lost all communication abilities except for screen to screen contact. She finally gave up on her husband when she saw he and another gentleman trying to connect the food processor and the computer together. He evidently was muttering something like, "I know all we need is the right interface to make this work." She didn't even look back as she drove away. (The names have been changed to protect the addicted). How can you prevent this from happening to you? Well, there are a few theories on what can be done, although little research has been conducted. First, prevention is the best medicine. Before it is too late, remove any technical electronic materials from the house. Censor articles in magazines and newspapers on personal computers. Try to make him think about the good old days, before everything got so automated. If you receive a incorrect bill due to a computer error, you have received a blessing from God. Frame it, and make a big deal about all the problems that computers cause. If your spouse already has computer equipment, you should know some important tips. First, lightning can kill a computer and peripherals if they are plugged in when it strikes. Make sure to leave his equipment plugged in during an electrical storm. Also, hiding disks is a good way to increase his frustration at the entire computer notion. If things have gone too far, there is another alternative, but this is only as a last resort. If you can't beat 'em, join 'em. That's what I ended up doing. Hey, honey, did you get that new piece of software I wanted?

Some other thoughts:

If you are like me, occasionally it gets a little tiring of constantly being considered inferior in the computer world because you are a woman. This woman was actually told by a rather blunt man that he thought my husband wrote my articles for me, and that he just used my name. Can you believe the nerve of some people? I think we women are going to have to get more involved in these silly computers, and more vocal, too! Anyone interested in a C-128's for Women User Group?

WHO ARE WE???

Recently, Twin Cities 128 has received a lot of mail from people and user groups who have decided to rename the magazine to "The Commodore 128 Journal" or "Twin Cities". In addition, Loren has received some mail addressed to Ms. Loren Lovhaug. If you think that the magazine needs a new name, let us know. However, I can't do anything about Loren's male gender (nor do I want to), so I guess you'll just have to live with that one.

Kids, Summer and C-128 *by Loren Lovhaug*

The summer months are traditionally a very slow time in the computer industry, especially here in the northern climes where pleasant weather is a luxury that fades all too fast. But now is a very important time for a certain segment of the population to think about computers, namely: PARENTS, especially parents of school aged children. Many educators warn parents not to let children's minds go "un-educated over the summer". They point out that a child needs to be constantly challenged creatively and intellectually throughout the summer months so the learning process is not retarded. Personal computers, and the C-128 in particular offer a unique opportunity for children to exercise their minds while still preserving freedom and flexibility (the very essence of summer).

Kids, Summer and C-128 Continued

YOU CAN'T BANG THEM OVER THE HEAD

Of course, as most parents can attest, children have an annoying tendency to reject those things which parents label as "intellectually stimulating", "educationally rewarding", and "conducive for growth and development" for those things that are FUN! (don't we all). Fortunately, computers are flexible enough to allow your child to be entertained and enriched at the same. However, just having the computer or even having a computer with entertaining educational software is not enough. This is because computers like toys are simply tools which extend the child's mind, and like most toys, computers will soon become boring to the child unless the right environment is provided for the child to interact with the computer. The following are the key elements in providing an environment conducive to an effective learning experience with the computer for kids.

1. Interesting subject matter: The old cliche about leading a horse to water holds true with children and computers as well. You can plop the child down in front of the computer, provide him/her with software, but you cannot make the child learn unless he/she wants to. Central to sparking the interest and imagination of a child, and as a result kindling the desire to learn is an interesting course of study or inquiry. Often this can be accomplished simply by choosing a subject that the child is already interested in. This is easy if the child is interested in something that is directly related to the computer, such as programming, however this may not always be the case. In instances where there is no direct correlation between the subject of choice and computers in general or there is no software designed to aid in learning about the subject, then the computer must take the role of an outside information processor. Through the used of online telecommunication services such as Quantum Link, CompuServe, Genie, Delphi, and the Electronic University you and your child can gather information on just about any subject without even leaving your home. The computer can also be an invaluable tool for expression of your child's thoughts on any subject. Encourage your child to write about the subject of their interest using a word processor (like Paperback Writer or Paperclip) or draw pictures about the subject using a graphics editor (like Geopaint or Doodle). Encouraging children to express themselves either through writing or drawing on the computer will enhance the child's familiarity with the computer in addition to improving the child's writing and spatial skills while still enjoying the freedom of pursuing the subject of their choice.

2. Encouragement and attention: Computers are not and never should be mistaken as electronic babysitters. As I stated before, plopping a child down in front of computer and commanding him/her to learn probably will not be very effective especially if you substitute quality interaction between you and your child for time on the computer. Computers don't make good conversation, smile, encourage, love or comfort, things which your child needs to foster an effective learning environment. So it is crucial that you as a parent do not "make yourself scarce" every time your child sits down at the computer, but instead show an interest in what they are doing and involve yourself in the actual learning process. In this way the computer can serve as a beneficial tool for bring you and your child closer together instead of isolating you from your child.

3. Avoid over-expectations: Perhaps the worst thing a parent can do with their child is place excessive demands on their child, this is especially true with computers. Recently I have met several parents who felt compelled to purchase a computer for their kids because they felt that if they did not, their kids would suffer an educational disadvantage. When I inquired what they were going to do if their child did not have an equal level of ambition for using computers as their parents did for purchasing one, one parent replied, "Well I guess I will just have to force him to use the computer for his own good." Reiterating what I have said before, it is extremely difficult to compel a child to use a computer or learn by using a computer unless he or she has a direct interest in doing so. This is especially true when you add the increased economic pressure placed on both you and the child when you rationalize the purchase of a computer as being a tool for your kids to USE! I recommend that parents do not purchase a computer specifically because they feel that it would be good for the child to have exposure to computers at home, unless the child demonstrates a genuine interest in computers prior to the purchase. Of course many of us already have computers for our own purposes, but even so it is amazingly easy for us as parents to place undue pressure on our children to perform on their computers. Perhaps the best thing to keep in mind when encouraging kids to use the computer is the realization that not everyone is cut out to be a "computer whiz" and that the development of good computer skills takes time and experimentation. So let your children experiment and progress at their own rate and avoid heaping too many expectations upon them.

4. Explain and establish good habits: Of course there are certain areas where you should expect your kids to perform. Perhaps the most important area is care and maintenance of the computer. Let's face it: computers are expensive investments and they are not necessarily child-proof. You should explain and enforce certain rules about behavior around the computer. Here are some suggestions:

A. No "rough-housing" around the computer, because it is very easy to get tangled in or trip over the myriad of cables and connections that are essential to the computer's operation. (you as a parent probably ought to "bury" these

Kids, Summer and C-128 Continued

connections as best as possible to avoid accidents which could damage you computer as well as injure your child.)

B. Be strict about eating and drinking around the computer. Nothing can be more frustrating than a shorted-out or gummed-up keyboard instigated by the wanton flow of kool-aid which just happens to escape from the glass. So be careful to establish a strict policy about eating and drinking around the computer and stick to it (nothing is more confusing to a child than a rule that's bent to often)

C. Make specific rules about the care and handling of delicate materials such as floppy disks. Teach your child how to handle and use items like floppy disks, ROM cartridges, dongle keys etc. Make sure they are also aware that these items should be put away after each time they are used.

D. Teach your children to let you know if something is not working or if they are unsure of what they are doing.

5. Too much of even a good thing...: Computers can be extremely addictive, especially for kids. And even if a child is learning and enjoying himself with the computer, it is unhealthy for them to spend an excessive amount of time with the computer. It is difficult to gauge how much time is "excessive" but keep in mind that the computer should not preclude or become a substitute for interaction with other children or for physical activity. Make sure your child seeks other activities outside of computers or shares his/her computer experiences with others.

To help get you started thinking how your C-128 can be used to help educate your child, here are some ideas:

1) Write a story building program for younger children in which randomly chosen sentences (by the computer) are strung together to form short stories. Children could then be asked questions on their comprehension and grammar use.

2) Music education on the C-128. This can be anything from testing basic musical knowledge about the musical staff and notes to music composition with BASIC 7.0's SOUND and PLAY commands.

3) Mathematical drilling on addition, subtraction, multiplication, and division skills, metric conversion, the ving of polynomial equations randomly queried by the computer, with the student typing the correct response. These are good practice for the student to improve his skills.

4) High school students often take probability and statistic classes during the regular school year. A good way to practice these skills is to help your child develop a program that will compute these kinds of statistics for a hobby they are interested in, like a sports team. Plus the writing of programs by students for the computation of complex statistics such as standard deviation, normal distribution, Chi square distributions, and linear regression often give a student special insight and understanding into the computation and uses of these statistics which is not normally gained from within the classroom.

5) Spelling and proof-reading and grammatical syntax can often be improved greatly through the use of a word processor. For those students who are particularly lacking in these skills, try typing up several documents purposely making spelling, grammatical, and punctuation errors and have your child correct them for you. You will find that this kind of practice will really spill over to improve your child's writing and syntax.

BobsTerm Pro 128 *by Margio Trellfar*

[We have recently learned that the C-128 is gaining a great deal of popularity with interstellar travellers. It seems races from beyond our planet actually prefer the C-128 more to computers of their worlds because of its flexibility and ease of use. Apparently, extraterrestrials have been stealing large quantities of C-128's from Commodore plants worldwide, and are in fact the root of Commodore's financial bind, and the shortage of C-128s on dealers shelves in some regions. When questioned about this recently, one Commodore executive stated emphatically, "NO COMMENT!" However, in our search for the truth, we intercepted the following communique:

In Search of the Perfect Telecommunications Software - Stardate 9875.3: The search ends. First Officer Margio Trellfar reporting...While mapping the vast C-128 software racks on Earth we made an important discovery, one which will have far reaching effects on those who use their C-128s as telecommunications devices. Sensors indicated that the product's name was BobsTerm Pro, a disk based terminal emulation package for the C-128 by Bob Ictini from gressive Peripherals. Being an avid telecommunicator of Sysop rank I acquired this piece of software for further ...estigation.

BobsTerm Pro Review Continued

Upon my return to starbase delta seti I immediately retired to my quarters for a session with my venerable C-128 and this alien program. I carefully opened the box and found a floppy disk, a good sized instruction manual and a small grey plastic "dongle key". I was especially pleased to see the dongle key. Dongle keys are the latest and perhaps most sensible form of software copy protection because it protects the rights of the software producer against illegal distribution of his/her product while allowing the user to freely and easily make backup copies of their software. The dongle for BobsTerm Pro plugs into the C-128's second joystick/mouse port and must be inserted before the program is executed.

Upon examining the manual I found that BobsTerm Pro to be an extremely sophisticated telecommunications program that was well documented. The manual has twenty-six sections which cover step by step every aspect of telecommunications with BobsTerm. I found the sections to be extremely thorough and so comprehensive that the manual for BobsTerm Pro could conceivably be used as a very good telecommunications primer. The only short-coming of the manual I could find was that while it had an extensive table of contents, the manual lacked an index.

On the box in which BobsTerm Pro is packaged it says, "The last communications program you will ever need", and that claim may in fact be valid. BobsTerm Pro is without a doubt the most complete telecommunications package I have ever seen on any microcomputer, not just the C-128.

Just for starters, BobsTerm Pro has a 60,000 character capture buffer. In my opinion, this is perhaps the most important feature of any terminal emulation software. This means you can collect an immense amount of information (equivalent to over 235 disk blocks!) for later viewing, printing or editing. BobsTerm Pro's capture buffer includes a built in editor which encompasses the best features of a good word processor such as block editing, search and replace, and output formatting. Buffer contents can be saved to disk, sent to a printer, or sent to another computer via the modem. For some individuals these capabilities alone would be enough to justify the cost of this program but BobsTerm Pro features so much more.

Of course another important consideration for any terminal emulation software is the modem compatibility. BobsTerm Pro has no problem here. BobsTerm Pro provides a variety of preset parameters for almost any modem that has come to market, plus Bobsterm Pro affords the user the ability to define their own parameters, thereby further insuring compatibility in the future. Bobsterm Pro also supports ten transmission speeds from 50 to 2400 baud. Another nice feature is Bobsterm Pro's ability to emulate some of the most popular dumb terminals such as Digital Research's VT100 & VT52, and of course the 80ADM31.

Bobsterm Pro also affords the user a great deal of flexibility as far as file transfers. In addition to the traditional Xmodem and Punter10 protocols, Bobsterm Pro also supports XON/XOFF, straight ASCII, straight binary, sequential line with prompt wait, and DC1/DC2 capture. With this kind of flexibility I would be surprised if there exists a bulletin board or a network anywhere in this galaxy that this piece of software will not be able to transfer files with. I also like Bobsterm Pro's view transfer option which allows you to see the incoming or out going data as it is being transferred.

Bobsterm Pro can also be fully automated with an extensive, but fairly easy to use macro programming language. There are provisions for autostart and autodial macros as well as the ability to automate any function or option on the menus. This means that the user can program complicated log-on sequences for getting onto local or national systems onto a single keystroke. In fact, anything that can be done from the keyboard can be programmed into a macro for automated execution. Macros can even be linked together to form very intricate sequences that are capable of conditional evaluation. Bobsterm Pro can also be programmed for timed execution of commands, or can be used as a remote BBS so that your C-128 could telecommunicate without you.

I could go on and on about the features of this package, but I realize that the Galactic High command is pressed for time so I will just add that Bobsterm Pro also incorporates: a word wrap toggle, two system clocks, a screen echo toggle, a programmable phone book and autodialer, and seven fonts styles plus the ability to add up to three more user font styles.

I am obliged to report, that this program while operating smoothly and completely "bug-free" it does have two drawbacks, first, the price. 1400 - 2000 galactic credits ($60.00 - 70.00) is a hefty sum for this program, but it is worth every penny for the SERIOUS telecommunicator. However, the casual user might be better to continue the search in the public domain. My only other gripe is with the character sets that are provided. I feel they could be a little larger and clearer, but this is only my opinion and this easily corrected by adding the standard set to the options (See Sparrow's Slick Tips this issue) although this option could have been provided by the author. Overall this is a superior product, to which I give my highest recommendation. --Margio Trellfar out.

Vizastar 128 *by Randy Margolis*

Vizastar 128 is a long awaited spreadsheet/database/graphics program for the C-128. Vizastar comes in an impressive package within which is a program disk, ROM cartridge, and the documentation. At a price tag of right around $100, it is pretty expensive for a Commodore product, especially when similar programs can be had for half its cost. Vizastar performs some extraordinary feats, however, and I think it should be seriously considered for the user who can appreciate its sophistication.

When first loaded you are presented with a blank spreadsheet with about the most putrid color combination imaginable. Fortunately, one of the first things discussed in the manual is how to change these awful colors. Also, each time you save a spreadsheet the color combination in use at the time is saved along with it. This product is designed to closely emulate Lotus 1-2-3 and, considering the memory "limitations" of the C-128, it succeeds quite nicely. Like Lotus, the program is menu driven, with a tap on the CBM key bringing up the menus, from which you can cursor to the desired choice or press the initial letter of the commands.

There are many facilities to aid in spreadsheet input, which is among the most tedious of tasks. For instance, the program can be set to automatically skip to the next cell to the right after typing in an entry and pressing return. In this way, you can just use the numeric keypad and not worry about cursor movement.

The spreadsheet contains 64 columns by 1000 rows which should be sufficient for the largest applications. Numbers can be formatted in several different ways including a true Date function. You can set the current date at the beginning of your session and include it in your work with the @today function. Very nice for date stamping your spreadsheet.

Vizastar is the only Commodore spreadsheet program I've seen with a true point mode. Anyone familiar with Lotus will be ecstatic over this feature. Suppose you want to write a formula which adds a large column, and then multiplies the result by a constant value stored in the worksheet. With Vizastar you can begin writing your formula, and when you get to the point when you have to start typing in the addresses, you may enter the "point" mode and move around the spreadsheet until you arrive at the cell you want to reference. Vizastar remembers where you were entering the formula and jumps back to it when you are through pointing. Unfortunately, you can't name cells or ranges and refer to them like you can with Lotus.

Formulas are standard, with cell addresses being a combination of a letter and a number, which in my opinion is far preferable to "Row 11, Col 24" syntax. The built-in functions (proceeded by the "@" sign) include SUM, AVG, MAX, MIN, a full set of date functions, rounding, and two very useful commands for table lookup. Most of these are identical to 1-2-3. Also, there is an @IF function which provides conditional control over cell entries. One thing I cannot understand, though, is why no financial functions were provided. Things like Net Present Value, Payment and the like. That's the kind of thing people use spreadsheets for, right?

Spreadsheet recalculation can be set to automatic or manual, rowwise or columnwise. Cell protection can be invoked to prevent accidental altering of key information. One other nice feature is that column widths can be set individually, allowing you to really dress up your work. True relative referencing is implemented with the COPY and MOVE commands. Vizastar automatically adjusts cell addresses when you copy formulas unless you specifically indicate otherwise. The windowing facility is quite impressive, permitting nine (!) windows to be open simultaneously. The program also boasts a lightning fast sort routine by which you can sort lists of information alphabetically or numerically.

The database function is truly impressive and co-exists with the spreadsheet, thus is available at all times. Values can be passed back and forth between these two sections, allowing you to update either one with the other. This is a true disk based database system, with a file structure surprisingly similar to Superbase. Supported are key field searches, importing and exporting of sequential files, and a special Report function. This really is like a subset of Superbase, without some of the fancier commands, but with the ability to interact with the spreadsheet you can turn some nice tricks with it. Unfortunately, the program crashed a couple of times when I was fooling around with the database section. I hope that this was just an isolated occurrence, since these crashes were unrecoverable.

Perhaps the most impressive feature of Vizastar is the "Exec" facility. You can automatically input any one of Vizastar's many command functions, database handling commands, etc. This is an actual programming language with conditionals, user prompts and looping. With a properly written Exec program you can totally automate your spreadsheet or database application! There are some truly awesome examples on the program disk which are fully documented for tutorial purposes, such as user menus, spreadsheet file merges, and so on. In fact, the entire disk tutorial is one huge Exec program!

Be prepared to keep the manual close at hand for a while, as the program author chose not to waste a lot of memory on help screens which, after you get to know the program, become extraneous anyway. Instead, there is a full 60

Vizastar Review Continued

kilobytes for your spreadsheets. The manual contains a wealth of information on all of Vizastar 128's many features, although I feel it could have been organized a little more effectively.

Printer support is very good, including the ability to send "setup strings", or escape codes, to your printer at the beginning of a print run. Also, on Commodore or Epson compatible dot matrix printers you can print out high resolution graphs from Vizastar's impressive graph section. Included are bar graphs, full color exploded pie charts, and, perhaps most outstanding of all, full color three dimensional multibar charts.

Is all this worth 100 bucks? Is the full database capability and macro language worth twice the price of similar C-128 application software? Consider also that with the need for a ROM cartridge you are precluded from using memory expansion units. This, I think is the program's most serious flaw. It is likely that the high powered business user is just the one that will be using the memory expanders. If your computing needs require the kind of power that Vizastar provides, by all means ask for a demonstration and compare it with other spreadsheet programs. You may find that Vizastar is just the product that you were hoping for.

Super C *by Harold Shore*

At last, another serious development language for the Commodore 128! Super C-128 from Abacus Software presents a great opportunity for programmers who prefer portable, structured languages to develop products for the C-128 market.

A basic feature of Super C-128 is the presence of a command line processor to control the development and execution of programs. This provides an environment closer to CP/M or MS/DOS than to the usual Commodore DOS. All operations are initiated from this command line. At the same time, Bank 1 of the C-128 is configured as a RAM disk, capable of holding function libraries, programs, and data.

The use of Bank 1 as a RAM disk has many advantages. First, the entire editing, compilation and linking process can occur with the system disk in Drive 0, using the RAM disk to hold temporary files. This makes the whole process quite rapid. Loading and saving programs to the RAM disk is virtually instantaneous. It is so fast, that at first one doubts that anything has happened. Multiple repetitions of the development cycle can be executed without having to swap disks, load files, etc. Of course, since RAM disk data disappears when the computer is turned off, it is important to save source and object files to a data diskette at the end of a session, or whenever you feel that some significant change should be permanently recorded.

Second, the use of Bank 1 as a RAM disk restricts programs to Bank 0: in practice, this limits programs to about 51k of compiled code. Since this code includes space allocated for data storage, it might seem that programs of only limited size may be produced with this system. But the RAM disk is always there. Programs can use it just as well as the development system. Thus data may be stored in RAM disk files - able to be accessed quite rapidly, and which remain even if program chaining is performed. Thus fairly powerful programs may be developed with this system.

Super C-128 comes with function libraries for math and graphics as well as an expanded standard library. It also has a macro definition file for the usual character testing and transforming operations. Unfortunately, there is no library of sound functions. And the graphics library is limited to supporting multicolor bit-mapped graphics. But it shouldn't be too hard to put together functions for hi-res graphics and sound. It would have been helpful if Abacus had supplied the source for some of their function libraries, even if this would have meant adding another diskette to the package.

The manual that comes with Super C-128 has been much improved over the earlier SUPER C 64. Abacus has even tightened up their spelling on this one. (Not entirely though!) The manual gives a fairly good hands-on introduction to using the system as well as an introduction to C itself. Thus the package is useful to programmers who are just getting into C as well as more experienced users. I was able to get the package running in minutes, editing, compiling and running a simple program within an hour of opening it.

I mentioned that program chaining is possible between programs produced by this compiler. It would be nice if overlays were implemented as well. This would allow large modular programs to share data more easily and rapidly. Of course, chaining to programs already loaded into the RAM disk should be fairly quick by current standards.

Super C produces some very fast code. In a simple benchmark that I wrote to compare execution of a simple loop, BASIC 7.0 in slow mode came out at 54 seconds, in FAST mode at 27 seconds, in a BLITZ'd version in 17 seconds. Super C-128 ran the same algorithm in 11 seconds, and with the computer in FAST mode in 5.7 seconds.

Super C Review Continued

As usual with Abacus products, Super C comes on a heavily copy-protected diskette. To have a backup, the company wants an extra $10 along with the registration card. It seems to me that anyone using a compiler will be using their diskette fairly frequently, with a greater chance of having the disk sustain some damage. Since it is annoying for to go without your tools waiting for a backup, this product would have been more useful to software authors if it had been protected by a 'dongle' or hardware device. Of course, Abacus might eventually follow the industry leaders by not copy protecting their programs. After all, not many hobbyists can really use a heavy-duty compiler such as this one, and anyone involved in commercial program production is likely to want the end-user support that having a legitimate copy makes possible.

More importantly, the copy protection scheme might damage a drive. The second time I attempted to load this compiler, my 1571 (only 5 months old) began to sound as if the read/write head was seeking a track located south of Rio. I quickly shut down the system. Prior to this I had experienced no problems with this drive. But from that time forward, the drive was seriously out of alignment, and I was forced to replace it. In Abacus's favor, I must report that the compiler is loading without incident on the replacement 1571. But the original drive should never have been damaged.

An additional problem arose after I had developed a 600 line source file within the Super C editor. As I made corrections to various places within the file, it was natural for me to scroll back and forth between the functions I was correcting. I began to discover stray digits scattered through my source text! At first I suspected that the video chip in my computer was bad - but I discovered that these random digits had actually become part of the source file, generating compiler errors when I moved into compilation mode. After some experimentation I found that these random digits were only produced when I scrolled rapidly through the file - movement using the GOTO LINE # command in the editor didn't produce this problem. However this really reduces the function of the built-in editor to a line editor, not the full-screen editor I had hoped to use. Super C stores text files as "usr" type files - a format not readily produced by other, more reliable, full screen editors. Although it is possible to import files produced by other editors, the tedium of doing so is hardly worth the bother. It negates all the ease of use achieved through use of the RAM disk.

This compiler is thoughtfully designed and would be of benefit to anyone who is looking for an alternative high-level language in which to program. But until Abacus fixes their editor, and ceases to threaten the functionality of our disk drives with their protection schemes, I cannot recommend that you purchase it.

1001 Things To Do With Your Commodore 128 *by Avonelle Lovhaug*

As I was perusing my local bookstore looking for new C-128 literature, I happened to come across a new book by Tab Books Inc. called "1001 Things To Do With Your Commodore 128." Typically, when I run across something with a title like this, I toss it aside as worthless sap. To me, a book that is going tell me how to use my machine is a waste of my time, because I already know what I want to do with my C-128, and if I didn't know, I shouldn't have one. However, since the book was newly released, and since it was C-128 specific, I thought I'd pick it up and warn others about it.

Much to my surprise, this book really charmed me. "1001 Things To Do With Your Commodore 128", by Mark Sawusch and Dave Prochnow, is a worthwhile book for anyone who owns a C-128 or who is considering buying one. It is NOT a list of software titles for the C-128, nor is it 200 pages of word processing and databasing hype; instead, it is a book filled with some creative and interesting ideas of ways to use your C-128.

The book is divided up according to broad applications divisions. For example, the first section is titled "Applications For Everyone." In this section, many of the usual suggestions for home computing appear, such as balancing the checkbook, indexing recipes, storing greeting card lists. However, what makes this book wonderful is that it also gives some ideas that most people never would think of. For instance, had you ever considered using your computer as a shorthand translator by developing a method of shorthand that the computer could decode while you wait? How about using your C-128 to figure out how much wallpaper you'll need for your entire kitchen? Do you keep track of your car maintenance information on the computer? Information such as when and where your last tune up was, etc. could prove useful and easy to locate if it is stored on a disk somewhere. These were all suggestions in this book. Not type-in programs, although there are a few short ones scattered in the book. Instead of programs, a short description of what the program would do, and sometimes an outline or a formula are provided. This book is especially useful for the programmer who has trouble of thinking of new programs to write.

The second section is devoted to business and financial applications. Investment theory is a significant part of this; the book becomes a sort of mini-primer for investment and stock theory. Several different stock analyses are provided, such as fundamental analysis, moving averages, price/volume trace, change distribution, price indexes, and

1001 Things To Do With Your Commodore 128 Review Continued

trend line analysis. The list goes on for several pages. Mortgage theory is also covered, as well as some formulas for depreciation. A few pages are included to get the readers started in more organized personal finances, and using spreadsheets to calculate figures. My favorite part of this chapter was the section called "Money-making Applications." Besides making stock predictions, the authors suggest using the C-128 to start of number of service-oriented businesses. Some of the more standard ideas were a word processing business, a typesetting and indexing business, and a mailing list business. However, the authors also included some that had never occurred to me, like starting a collection business, a newspaper clipping service, a computer dating service, a supermarket comparison service, or selling personalized children's books (writing and printing done on the C-128). This certainly isn't an all inclusive list, but it did offer some fresh notions for those of us who had been considering this idea for a few years now. Next, the authors suggest that business decision making can be facilitated on the C-128, helping to compute reorder timing of inventory, demand forecasting, market research, analysis of vendors for purchasing, and bid preparation and job pricing.

The next two chapters involve mathematical, statistical, and scientific applications the C-128 can be used for. Formulas for some basic statistics are included, as well as some examples of where those statistics might be important. As a former math student, I was impressed by the kinds of mathematical problems I could solve using my C-128; if only I had known then what I know now! Two important sample programs are offered, a trigonometric calculator and a trigonometric plotter program. Both programs are fairly short, and although they aren't spectacular, they do meet their respective objectives. Also, I think the person who types them in will learn a few things about BASIC. Probabilities are also covered, as well as some mathematical games that can be played via the C-128. Scientific applications, according to this book, include weather forecasting, energy efficiency computations, and medical, chemistry, and physics calculations. This kind of number crunching is something I always knew could be done on my C-128, but the authors present such interesting questions and problems to be solved, that I was intrigued and wanted to try out some of their suggestions.

The next few sections deal with educational and entertaining uses for the C-128. Besides the obvious educational use of drill and practice that the computer provides to students, some other ideas are given. Simulations are a large part of this, and such simulations as world dynamics, astronomical theories, and the growth of an amoeba colony are a few examples of the kinds of things the book suggests. Some other educational program ideas given include math flashcards, logic reasoning, metric calculations, spelling drills, and speed reading tests. It was unfortunate that the educational section was so short; it is my belief that the educational uses of the C-128 are underplayed and need to be exploited more fully. However, there were several original ideas given, so it wasn't a total waste. Probably my favorite section of this book was the hobby applications chapter; I really enjoyed some of their ideas. For the visual arts, the authors suggest random art patterns generated by the C-128, or kaleidoscope patterns. For the musical arts, programs that compose music are an interesting concept, as well as using the C-128 to educate students and quiz them on musical notation and important works. Sports statistics, garden planning, amateur radio assistance and model building are all ways the C-128 can be used to assist people with their hobbies. And of course, an entire chapter is devoted to the games that people can play and write on their C-128 for entertainment and education.

Some of the last chapters of "1001 Things" deal with the C-128 applications that are more complicated, and are also many times more expensive. A big section of this is the notion that the C-128 can be rigged to control household appliances, lights, alarms, and security apparatus. With the right connections, and the right equipment, a robot can even be hooked up to be controlled via your C-128. (According to the authors, HEROjr, by Heath/Zenith Corp., is one such robot that is already available to be connected to the C-128.) The book also spends some time describing national (such as CompuServe) and local BBS systems, and the advantages of gaining entry to such services. (The book tends to focus on CompuServe, barely mentions Quantum Link, and doesn't mention GEnie.)

The very last chapter of the book gives us few more way out ideas of things to do with our machines. Language translators, dictionary compilers, literature analysts, and grammar checkers are just a few. Aid to the handicapped is becoming a more frequent use of the C-128; exploiting such areas as voice synthesizers, conversion of text into braille, telephone dialers, and quick information management are some brief suggestions in a new field of applications opening up to the C-128 world. The last few pages of the chapter list 50 miscellaneous applications which didn't always fit into the other categories.

My only complaint about this book, and it is a minor one, is that occasionally I would have appreciated a little more guidance from the authors on where to look for software that would perform these kinds of functions, or at least where I could buy a book to teach myself to write the software. Related to this problem, sometimes the book seemed to general for me, and I wished for specific C-128 examples. These, however, were infrequent occurrences, and most of the time I was just plain awed at the ability of these writers to come up with so many unique thoughts, and ones that were useful too!

1001 Things To Do With Your Commodore 128 Review Continued

I was really impressed by this book. It was very evident that a lot of creative thought had gone into the development of it, and by the time I was past the first chapter, I was excited to get started on some of these new ideas. For the person who isn't interested in taking their machine farther, this book is worthless. But if you are the type of person who wants to get the most out of your C-128, I suggest that you give this book a chance.

Ram Expansion Overview *by Loren Lovhaug*

Technolust: The overwhelming desire, bordering on obsession, to acquire electronic gadgetry, such as personal computers and peripherals, stereos, video equipment etc.

Okay, I'll admit it. I suffer from an acute case of technolust. I like gadgets, and gadgets to improve my gadgets. It probably started long ago with my first digital watch and has progressed uncontrolled ever since. I have often told my wife, it could be worse, I could lust after other things, that are far more destructive (drugs, gambling, illicit sex etc.) Computers can very easily breed technolust. For me, my computer technolust began with the my first personal computer, a Vic-20. The Vic-20 was a very good inducer of technolust because, while it offered a lot at the time for the money, it lacked an awful lot. For a few weeks I was satisfied with the Vic's "whopping" 5k of random access memory, but as I progressed I soon found technolust setting in and I longed for more, so first came a disk drive, then an added 16k of random access memory, and then another 8k and an eighty column board, and then a modem and a second disk drive, and so on...through the C-64 all the way up to our present day C-128. I must admit the C-128 has kept me content in its own right for a lot longer then most of my other "technotoys" probably because the C-128 comes with so much (80 column capability, 128k of RAM, an excellent implementation of BASIC etc.) But now technolust has set in again and my eyes and my many times depleted wallet have turned to yet another piece of hardware, the 1700/1750 RAM Expansion for the C-128.

Now the frugal amongst you might demand some kind of logical reasoning for the purchase of such a device (my wife sure did) and to be honest there is none. And as my wife so eloquently pointed out there is no "PRACTICAL" reason to acquire ram expansion for the C-128. In fact, she presented me with a rather large laundry list of reasons why I should not buy a RAM expansion. Her reasoning went something like this:

First, the memory is not Direct Access memory. This means that when you add your Ram expansion to your C-128 you are not really increasing the amount of internal memory to the system, that is memory that is DIRECTLY available for my BASIC or machine language programs. This has to do with the fact that the C-128 is still an eight-bit micro which in reality can only address 64k at any given instant. But then you ask, how can my C-128 be a true 128k machine? Well rest assured it is, but not in the conventional sense. The whiz kids at Commodore thought up a rather brilliant scheme that allows the C-128's operating system to switch between two 64k banks of random access memory plus the system's read only memory very quickly, allowing the C-128 to truly incorporate 128k of RAM though technically not all at once (this does mean that there are a few peculiar quirks about programming the C-128, but in practice it adds a great deal of power and flexibility to your system). Because of speed and certain other limitations inherent in the C-128's 8510 microprocessor the C-128 does not use the same internal bank switching procedure for addressing the memory within the RAM expansion module, but instead relies on an external switching technique (more on this later) to "trade" and "replace" certain areas of the C-128's internal RAM.

Second, my wife stressed the fact that I have yet to write a program which as she put it, "totally exhausts" the capabilities of my C-128 under its current configuration. Here again she is correct, although I have done a great deal of programming on the C-128 I have yet to really tap its absolute "full" potential (although, to be truthful, I believe that I nor anyone else in reality has tapped the full potential of any micro, even my old Vic or my trusty C-64).

Last, she quite fervently points out that all of the excellent productivity software for the C-128 (like Superbase, Pocket Writer, Multiplan, et al.) function like a champ in my C-128's current configuration and as of this writing none of them have the capability to address the ram expansion (though I have been assured by several sources that there are revisions in the works...stay tuned).

"Okay, Okay already", I said picking up a white shirt and waving it symbolically as a gesture of surrender. So why then, you might ask did I risk divorce and financial ruin and pick up a ram expansion? Well I justified it to my wife (fully expecting to be shown the door) by suggesting that it was a must for Twin Cities 128 (I was not lying), and that it might help save Commodore (stretching a bit), and that it is very important for me to retain my pioneer spirit in the world of computers (really stretching it), and most of all I told her that my SICK mind saw a great deal of potential for the device (the absolute truth, technolust strikes again!). She grumbled a few obscenities and let me hide in the computer room to play with my new toy.

Ram Expansion Overview Continued

It turns out that both my wife and I were correct about the Ram expansion My wife was correct in her assertion that at this time the Ram expansion does not really have a great deal of practical value especially given the lack of software that is currently available to take advantage of the Ram expansion's potential. But I also was correct in my belief that the Ram expansion does indeed hold a great deal of potential, especially for those willing to take the time to design applications which exploit its talents.

programming the Ram expansion is not really difficult if you keep in mind that the Ram expansion is not a direct memory access device (as stated earlier) but instead is really a large external storage buffer. The Ram expansion uses a special chip designed by Commodore called the Ram Expansion Controller (REC) to transfer or trade the memory contents between portions of C-128's internal Ram memory and the external memory in the Ram expansion module. These manipulations are accomplished in Basic 7.0 through the use of the STASH, FETCH, and SWAP commands or from assembly language through the DMA CALL Kernal routine and manipulation of the MMU's configuration register. The results are breathtaking, the REC is capable of transferring memory at an incredible 1 megabyte per second rate (however, the maximum you are allowed to transfer is 64k at any given instant). This kind of speed opens up a whole new world for the C-128, for example, an entire graphic screen (8k) can be animated at speed up to 128 fRames per second (with the 512k expander you could store up to 64 separate screens, with the 128k expander you could store 16 screens) allowing real time animation of the sort previously only available on machines like the Amiga. Combine these capabilities with C-128's sprite facilities (consider sprites moving across rapidly changing backgrounds) and I think we may very well see a whole new generation of game and graphics software for our little 8 bit wonders. Of course there are other areas where the Ram expansion's speed and flexibility create revolutionary potential for the C-128, consider creating Basic programs with program and/or variable overlays (through the use of the C-128's pointer command) that are appended immediately into the executable code. Or consider a disk copying utility which copies entire 1571 formatted disks in ONE pass through the use of the 512k expander. However, at this time, all of the programming for such applications would be left up to you, but with this kind of power at hand, it more than likely will not be very long before you see commercial packages which feature Ram expander options (we know of a few already which are in the works: GEOS 128, Paperclip II, Pocketwriter 128 (upgrade)). Plus there is always CP/M which handles the Ram expansion like a super fast disk drive (just PIP the files you want access frequently to drive M: at the beginning of your session).

So what is my conclusion about this product? I like it a lot! But then again I suffer from a severe case technolust. Should you buy one? Well that depends upon you. If you are a programmer/hacker/potential developer interested in exploring new areas to exploit your C-128 then by all means, go out and get one of these babies, you will not be disappointed (at least not until the bill comes through or your spouse files for divorce). If you are a person who is strictly a non-programmer/applications user I recommend you hold off on the Ram expansion until we start seeing commercial packages designed to take advantage of this product, but do keep it in mind, because this product really makes the C-128 zing!

Forgotten BASIC *by Loren Lovhaug*

As I said last month, there is a great deal of C-128 public domain software that is starting to surface around the country, and that is great! Unfortunately, however it seems that many BASIC programmers are ignoring a much of the power of BASIC 7.0 in favor of the tried and true, "blood and guts" methods they previously used when writing programs for the Commodore 64 and other machines with limited BASICs. This is truly a tragedy, because as we observed last month often the new commands and structures of BASIC 7.0 can really save memory and execution time, and make your code much easier to understand and work with.

Things are really SSHAPEing up

Most programmers are aware of the C-128's bit mapped graphics commands like DRAW, CIRCLE, BOX, PAINT etc. These commands are used to draw figures on the C-128's 320 x 200 pixel graphics screen. Most programmers are also aware of the C-128's fantastic sprite graphics abilities. However a vast majority of C-128 programmers ignore perhaps the most powerful graphics commands available in BASIC 7.0. These "forgotten gems" are the GSHAPE, SSHAPE, and the SPRSAV commands. What makes these commands so special is that they allow you to store and manipulate graphic data in string variables. This gives the BASIC programmer a very fast, convenient, and efficient means for "editing" sections or even whole graphic screens. Consider the following program (typing it in may help your study of it):

Forgotten BASIC Continued

```
100 DEF FNr(x)=INT(RND(1)*x)
110 COLOR 0,1:COLOR 1,7:COLOR 4,1
120 GRAPHIC 1,1
130 CHAR 1,1,1,"watch this space!"
140 BOX 1,5,5,143,16
150 FOR i= 0 to 5
160  SSHAPE a$(i),i*24,0,(i*24)+23,21
170  SPRSAV a$(i),i+1
180 NEXT i
190 FOR i = 1 to 20
200  GOSUB 520
210  FOR j = 0 to 5
220   GSHAPE a$(j),x+(j*24),y,m
230  NEXT j
240 NEXT i
250 FOR i = 0 to 5
260  SPRITE i+1,1,2
270  MOVSPR i+1,100+(i*24),100
280 NEXT i
290 FOR j = 1 to 20
300  SLEEP 1
310  FOR i= 6 to 1 step -1
320   SPRSAV i,a$
330   SPRSAV a$,i+1
340  NEXT i
350  SPRSAV 7,a$
360  SPRSAV a$,1
370  GOSUB 460
380  FOR i =1 to 6
390   MOVSPR i,x+(i*24),y
400  NEXT i
410 NEXT j
420 FOR i=1 to 6
430  MOVSPR i,FNr(360)#(FNr(8)+1)
440 NEXT i
450 SLEEP 10
460 FOR i=1 to 6
470  MOVSPR i,0#0
480  SPRITE i,0
490 NEXT i
500 GRAPHIC 0
510 END
520 x=FNr(200):y=FNr(200):m=FNr(4)
530 RETURN
```

The key to this little demonstration is the ability of BASIC 7.0 to take bit mapped data and store that data into string variables. First the program places the message "watch this space!" with a box around it onto the graphics screen (lines 130-140). Then the message and the surrounding box are stored into a string array via a FOR..NEXT loop and the SSHAPE command (the SSHAPE command stores a representation of an area on the graphics screen into a specified string variable) and six sprites are defined from the newly formed array via the SPRSAV command. (The SPRSAV command performs "double duty"; it is used to transfer graphic data from string variables to sprites OR transfer the graphic data from a sprite to a string variable.) Note, when using the SSHAPE command to define shapes for later use as sprites one must carefully choose the coordinate parameters for the SSHAPE command keeping in mind the 21 x 24 pixel size limit for sprites. (lines 150 - 180). The next section of the program uses two FOR..NEXT loops and the GSHAPE command places our shape data (stored in the array) on the screen at random positions on the screen. (The GSHAPE command places graphic data stored onto a graphics screen. It also has various placement options for altering how the graphic data being placed affects the graphic data already on the graphics screen.) You might want to note how the random positions are being chosen via the user defined function in the subroutine at line 520. Another thing to observe about the execution of this program is how the GSHAPE command's placement options effect how the message is place onto the screen especially in conjunction with already existing graphics data (the placement mode is randomly chosen, see the definition of the numeric variable "m" in line 520). These effects can be very useful when overlapping various images. (lines 190 - 240) The rest of the program demonstrates the power and flexibility of the C-128's sprite capabilities for animation. For a more detailed discussion of sprites and the use of the GSHAPE, SSHAPE, and SPSAV commands see pages 109 - 123 in the C-128 System Guide and Chapter 4 in the C-128's Programmer'S Reference Guide as well as the Basic Encyclopedias in the System Guide and the Programmer'S Reference Guide.

In the MID$dle of it all

Since we are discussing extraordinary methods of defining variables in this addition of Forgotten BASIC, I thought I would remind you of something Sparrow James brought up in one of his Slick Tips a few months back about the insertion of substrings using the MID$ function. Most BASIC programmer's are aware that the MID$ function is used to glean substrings from string quantities. However, BASIC 7.0's version of the MID$ command also allows you to INSERT substrings into string quantities. Sparrow's example demonstrates this:

```
10 A$="comes"
20 B$="Here       the sun"
30 MID$(A$,5,5)=B$
40 PRINT A$
```

This quirk can really be beneficial for writing string replacement and editing routines.

Sparrow's Slick Tips *by Sparrow James*

00100110: Bring Bob Back to Normal Being a Sysop and an avid telecommunicator, I really appreciate a full featured telecommunications package like Bob Letini's BobsTerm Pro. However, amidst all of the wonder bells and whistles of Bobsterm Pro there is one major irritation, namely, Mr. Letini has created his own character sets for displaying text and other information. I have found the display fonts somewhat difficult to read. Fortunately, BobsTerm Pro is so comprehensive that it allows you to add your own character set for use inside the program. You could use a standard character set editor to edit an existing font to your liking, but there are a few caveats you must be aware of. First the character sets in BobsTerm Pro are in standard ASCII order and not PET ASCII order, and second BobsTerm uses several specially defined characters which display control codes and graphic borders which really should remain intact. With this in mind, I have written the following program which creates a new character set on a BobsTerm disk which contains all of BobsTerm's special characters but replaces the alphabetic and numeric characters with ones from the C-128's normal character set. The new character set can be used directly within the program itself or can serve as a basis for further modification through the use of a character set editor. Programmers may want to note the use of location 215 (line 100) to check which display the user is using and the error trap in line 190. Also, if you run this program using a 40 column display you will get to see the actual character set translation being implemented since the translation takes place on the 40 column graphics screen.

```
100 IF PEEK(215)=128 THEN sc=5:ELSE sc=0
110 PRINT "place a bob's term pro disk in the drive and hit a key."
120 GETKEY a$:GRAPHIC 1,1:TRAP 190
130 BLOAD"character set.0",p11392:BSAVE"char rom2",b14,p55296 to p57344
140 BLOAD"char rom2",p8192
150 FOR i=8200 TO 8407:POKE i+3968,PEEK(i):NEXT i
160 FOR i=8712 TO 8919:POKE i+3200,PEEK(i):NEXT i
170 FOR i=8576 to 8664:POKE i+3200,PEEK(i):NEXT i
180 BSAVE"character set.7",p11392 to p13440:GRAPHIC sc,1:SCRATCH"char rom2":DIRECTORY:END
190 IF ds=62 THEN GRAPHIC sc,1:PRINT"w r o n g  d i s k !":SLEEP 3:GRAPHIC sc,1:GOTO 110
200 GRAPHIC sc,1:LIST:PRINT:PRINT"disk error:";ds$:DCLEAR
```

00100111: Calculated Restore As most of you who read Twin Cities 128 regularly are aware, there are numerous advantages to programming in the C-128's BASIC 7.0 over C-64's BASIC 2.0. Here is another advantage that many programmers are not aware of. The RESTORE command (the command which resets the available data pointer for the READ..DATA structure) has been enhanced in BASIC 7.0 to allow you to RESTORE from DATA in a specific line in addition to the blanket RESTORE of old. This means the cunning programmer can use actual program lines in a BASIC program more effectively for mass storage. In fact, by using some kind of dynamic keyboard routine to add/insert data into a BASIC program it is conceivable that one could write a fairly comprehensive "self-contained" database manager for all kinds of applications (remember with the C-128 you have almost 60k available for BASIC text). Consider the implications of this example (also note the use of the PRINT USING command in the last line):

```
  10 DATA Harold,Male,Programmer
  20 DATA Curt,Male,Software Retailer
  30 DATA Avonelle,Female,Magazine Editor
  40 DATA Gary,Male,All around good guy
1000 INPUT"Retrieve data on which individual (1-4)";dp
1010 RESTORE (dp*10):READ N$,S$,O$
1020 PRINT USING "Name:################Sex:##########Occupation:##########################";N$,S$,O$
```

00101000: Superbase Snafu As many of you Superbase 128 users who are manipulating large databases on the ever loving but sometimes "flaky" 1571 may have already learned there is a severe bug in the program which can cause a rather serious catastrophe when the database gets large enough to "spill" over onto the back side of the disk, especially when using an operation which involves the manipulation of a sequential file such as a FIND or a SORT. The problem (according to Precision Software) has to do with Commodore's deviation from the original design specifications released to Precision on the 1571. Commodore will soon be marketing upgrades of the 1571 ROMs (and new C-128 ROMs) that will correct this problem. In the interim it is probably best to avoid the second side of a 1571 disk.

00101001: Filling Your Bit Maps The C-128's built-in monitor can really come in handy even if you are not a machine language programmer. I have found there are times when developing a graphics screen (or importing one from a program like Doodle) it is necessary to change the background color of the bit map. This is easily accomplished by simply entering the monitor, then entering the following: F +7168 +8191 +01. Upon exiting the monitor by typing X and return and examining your bit map, you will find that the background color has been changed to white. By changing the last value you can specify which color you wish to change the background color to.

nor/Opinion/Mayhem *by Loren Lovhaug*

HAPPY ANNIVERSARY C-128

That's right, the C-128 has been on the retail market for one whole year now. For me it seems much longer, because it is very difficult for me to think of not having this fabulously versatile tool. A great deal has happened in the C-128's first year of commercial distribution, here are just a few of the first year highlights:

1. The C-128 boasted North American sales of over 600,000 units and around 1 million worldwide (counting C-128d sales in Europe) making it one of the most successful selling computers of all time in the first year it was commercially available.

2. Software development for the C-128 (in C-128 mode) has focused mainly on productivity applications [and contrary to popular belief has been extremely brisk, at last count we had over 100 titles] giving C-128 owners who use their machines for so called "serious applications" an incredible number of quality choices for extremely low prices. I personally know of many Apple and PC clone owners who wish they had our productivity selection at the kind of prices we enjoy.

3. There has been a tidal wave of publishing houses producing quality texts about our machine. Just go to the computer section of any B. Dalton Bookseller or Waldenbooks or any other bookstore and I think you will be amazed at how many titles there are on the C-128. The grand-daddy of them all, "The Official Commodore 128 Programmer's Reference Guide" published by Commodore and Bantam, was a little late, but is a 744 page wonder reference on all aspects of the C-128 which amazingly contained just about everything you could ever want to know about the C-128 with numerous examples and most thankfully, very few errors!

4. The 1571 disk drive proved to be exactly what it was designed to be, the most versatile disk drive ever hooked up to a microcomputer. Unfortunately, a few bugs have reared their ugly head (they always do) having to do mostly with complicated operations involving the second side of the disk. But, in a bold and refreshing move by Commodore, these bugs are being addressed in the form of upgrade/replacement ROMs which are going to be made available to the end user for a nominal fee (more on this in future issues, stay tuned!).

...now computer gossip mongers, here are a few developments I see on the C-128 horizon:

1. In the next few months we should see commercial software appear which utilizes the 1700/1750 RAM expansion units. We already have heard of several companies working on either upgrades of their current products or new products which will tap into the power these devices offer the C-128 world. Dropping a few names: Batteries Included, Digital Solutions, Solid State Software, Berkeley Softworks, Abacus...just to name a few!

2. Sometime in late October or early November Berkeley Softworks should be ready to release the C-128 version of GEOS [more on this in our monthly GEOS update column]. The C-128 version of GEOS could radically alter the C-128 world, and could very well give the C-128 the power and might it needs to remain a viable competitor in the shadow of the CLONES and the 16/32 bit glamour machines.

3. Yes Virginia, there is goint to be a 1581 3.5" disk drive after all (thank goodness)! Jim Gracely and Jeff Porter of Commodore have swore to me that this product IS A GO! This 880k slimline 3.5" 9000 cps disk drive will give C-128 owners a very nice mass storage alternative and make the C-128 even more competitive in the small business market. Unfortunately, RUN magazine's lofty claims that the consumer is going to see this drive in the fall of 1986 are totally unfounded. Both Gracely and Porter indicated that this drive should be on the retailer's shelves (though not very long, they will sell like hot cakes!) around February or March of 1987 (that is if the FCC cooperates!). So don't forego the mortgage payment this month, but file those tax returns early! [We have also heard that the major software developers should be receiving prototypes of this drive in Amiga 1010 cases sometime in the next month. We are going to beg, plead, grovel, cry, whine...etc. in an effort to get one of these babies so you can have the scoop, but don't hold your breath!]

4. Rumor has it that the retail price of the C-128 is going to increase sometime during the next few months. We certainly hope that this is not case, however, those sitting on the fence about purchasing their C-128 (or a second or third..for that matter) might consider doing it now. Around the country the going retail price for a C-128 is somewhere between $250 - $285, the 1571 seems to be stable at around $250. On the more pleasant side of pricing, we have heard the price on the 1700/1750 Ram expansions are scheduled to drop sharply in the very near future, the going rates at most retailers seems to in the neighborhood of $250 - $275 for the 512k 1750, and $130 - $160 for the 128k ... It may pay to keep your eyes on all of these prices! [Note these price ranges were the consensus as of the 15th of August 1986]

Software Thieves *by Curt Swanson & Avonelle Lovhaug*

One of the most misunderstood and controversial topics in the microcomputer world is that of duplicating copy-protected software. The vast majority of commercial ("for sale") software is copy protected. The purpose of copy protection is to preserve the rights of the commercial software developers from the illegal distribution of their software. However, most copy protection schemes prove ultimately futile because they can be circumvented or eliminated by persistent thieves using sophisticated disk backup utilities.

Software and the Law

The copyright laws of the United States are indeed complex. But, the intent of the law is clear. The law prohibits the unauthorized reproduction or duplication of copyrighted material in any form by any means. (Since 1980, this has included computer software and accompanying documentation.) The penalties for being convicted of software theft are specific and harsh. They include provisions for treble damages, punitive damages, and prison terms for those convicted of software theft.

Why do they do it?

Certainly these penalties are sufficient to deter an otherwise law-abiding citizen from engaging in such illegal activities. And yet the software theft problem is widespread and spans all social and economic classes. Why do some people risk the penalties? Here are some typical unjustifiable excuses that people use to rationalize BREAKING THE LAW:

Transparent excuse #1: Everyone does it (the popularist argument):

The RIGHT stuff: 1) If that were true, nobody would be in the business of developing commercial software since "everyone" would just steal it anyway. 2) Two (or, for that matter, two million) wrongs still don't make a right. 3) Thousands of murders were committed last year in the United States, but my conscience just won't permit me to "run with the crowd" on this one. What's wrong is wrong, period.

Transparent excuse #2: What's the difference? Nobody ever gets caught!

The RIGHT stuff: Don't believe it! An August 18, 1986 Infoworld article by Scott Mace reports that the operator of an alleged "pirate" bulletin board in New York is being sued by a software publisher for copyright violations. A number of fellow software publishers are providing monies for the prosecution of this federal-court case. Arrests and convictions are well documented. Such stories are usually carried in "trade journals" and therefore (unfortunately) are not seen by the public at large. But, make no mistake, software theft is <u>not</u> a "safe" activity.

Transparent excuse #3: Software is overpriced, so publishers deserve to have it stolen.

The RIGHT stuff: Agreed, some software is priced too high. But consider the economic law of supply and demand. If a piece of software is in fact overpriced, low sales will dictate that either the price will come down, or the product will be displaced in the marketplace by a more affordable alternative product. Meanwhile, pricing carries no inherent right to steal anything that one cannot afford to buy.

Transparent excuse #4: I do not distribute copies to others, in turn, so no one is really harmed.

The RIGHT stuff: Most commercial software authors work under a royalty arrangement. Therefore every illicit copy is money out of their pockets. You and I don't expect to work for free -- neither should they. Recently, two prominent software developers expressed their dismay at the rampant software theft, and said that they are having serious doubts about continuing in that profession considering the continual assaults on their pride and pocketbooks by software thieves.

Although software thieves (if pressed to do so) could probably add thousands more to this excuse list, the point should be clear. There is simply NO justifiable reason for "software mugging."

Finally, we would like to address the question of "backup utilities" -- those software utilities that enable the user to create duplicate copies of their software library. The computerist should and, in most cases can, make backups of their software. The same law which prohibits the illicit duplication of software allows for the owner of a program to make backups for their own use. Backups are generally available (to legitimate owners), for a charge, from the publisher of a program. But the ability to make your own copy allows you to use a new piece of software immediately, without the fear of something happening to our only copy. In addition, it is usually cheaper to make your own copy

Software Thieves Continued

then pay the company for a backup. And in some cases, the copy will bypass the protection completely, leaving software that is faster to load, and won't knock the drive out of alignment. However, we are fooling ourselves if we think that the producers of copy utilities believe that their software is only used for legal copies. These companies would not be in business if it weren't for software thieves, because these are generally the people that will buy commercial protection-breaking copy utilities.

Heavily protected software makes it difficult for the legitimate user to make backup copies, and leaves them worrying about their own copy every time their disk error light starts flashing. But software producers and developers must heavily protect their software, or have their paychecks ripped out of their hands every time a thief steals one of their programs. What should be done? It is clear thieves must be stopped in their tracks. Hopefully, stiffer penalties are on the way, as well as improved surveillance and increased litigation. This in turn, will serve as a warning to those already committing the crimes to stop what they are doing and think about the implications of their actions. Meanwhile, increased support should be given to those companies who have attempted to make copy protection as painless as possible in the form of protection that removes the focus from disk-based protection (that leave our disk drives crying) and move towards hard protection (such as dongles, ram-based, or optical key protection) that allows the legitimate user to make backups. However, the ultimate solution lies in making people aware of the damage software stealing is doing to the industry, to the developers, and to the users. Until people are aware of the damage that is created by thieves, the situation will only continue.

Record Master 128 *by Avonelle Lovhaug*

I don't know why people keep complaining about a lack of software for the C-128. I've had no trouble finding new programs for our favorite machine, and recently came across a new database program that utilizes the features of C-128 mode. Record Master, by WOODSoftware of Wichita, Kansas is a well thought-out database program for the C-128.

Record Master comes in an attractive, red on gray three-ring binder, with useful (but not overzealous) documentation. The program is not copy protected, which relieved me because in addition to being able to make a back up of this program, I didn't have to worry about my drive heads being knocked all over the place. The program also takes advantage of the autoboot features of the C-128, and is available in 40 or 80 columns. After a brief introduction and a glossary of terms that the manual will refer to (pretty standard stuff for the computer junkie, but very useful to the novice). the manual instructs you to make a back up copy of the program. I advise you to use a different back up utility than the one provided on the disk, I found it entirely too slow. Also provided on the disk is a "SETUP" program which will allow you to choose your default configurations for the disk drives, printer, and screen colors. After choosing your defaults, the information will be stored on your master disk for the program to use.

After self booting, the main menu offers 4 choices: open an existing database, create a new database, recover a corrupt database, or quit the program. Throughout the program, menus and prompts are clearly displayed, so that the program is easy to use and understand. To choose an item from a menu, you must type the first letter of the choice. Creating a new database is fairly simple. After choosing "C", the program will prompt, "Is this database like another database?" If you have similar databases (for instance, two similar databases that you keep separate for each different year) then you can use this option. To create a new database, Record Master will need to know approximately how many records the database will have, the number of fields per record, and how many characters per record. This is one of the unique features of Record Master. Instead of choosing the number of characters to be allowed in each field, the program requires you to do a little math on your own. Compute how many letters will be needed for each field (approximately), add them together, plus add 2 more for each field. This is the characters per record. Because of this method, if you have one record that has a very long first field, for instance, you won't have to restructure the entire database for that one record, or abbreviate the record to make it fit. Instead, Record Master adjusts the field size accordingly, only running into difficulties when the characters per record are too large. You shouldn't have a problem with this if you follow the directions carefully. (Incidentally, if you want to limit the size of a particular field, this is possible.) Next, you will choose a name for each field, and type them in at the appropriate prompts. The last thing required to create a database is choosing a key(s) for your database. Record Master will let you have as many keys as possible, and each key can include up to five fields. The keys are used to arrange the records in a specified order. I found that I appreciated the ability to choose more than one field for my key, as well as being able to give the program more than one key at a time. (When printing, the program will then prompt you as to which key to sort by for the hardcopy.) After all of this information has been entered, the program will tell you the size of your file when it is full, and the amount of Ram used. Finally, the program will let you save the database format on a disk.

Now the program will take you back to the main menu, so that you can open your new database. The open database submenu will allow you to access the records, print mail labels, print reports, switch the case of all the records in

Record Master Review Continued

your database (upper to lower and vice versa), use any disk commands, go back to choose another database, translate your database, or quit the program. Record access gives you many of the standard features, allowing you to add, change, delete, scan, search, or print a hardcopy of the current record. In addition, there is also an option to mass enter your data, to be merged with your database later on. This can save considerable time, especially if you don't like disk access after every record entry, or if you don't have information for all of your fields at the present time. Mass entry will let you choose which fields you wish to enter, and they will be stored in a temporary sequential file. This can then be merged with your database. You also have the option of reordering you records in any way that you choose, limiting the size of a particular field, or doing calculations to update certain fields. One final option, destroy database, is also available. Although the manual didn't mention this, the menu gives you the option to destroy a single record, or an entire database. However, when I tried to destroy a single record, the disk drive gave me an error, which eventually dropped me back to the main menu. This happened whenever I tried the "destroy single record" option, although destroying the entire database was not a problem. Personally, I would never use that option to delete a record anyway, when you have that choice in the "work on records" menu, where you can also see the record before you kill it. I suspect that this option was later eliminated since it is redundant, and was never entirely killed from my version of the program. Another anomaly in the program is the mention of a "user" option in the record access menu. This option supposedly gives the user the ability to alter the program. However, my version of the program did not offer this choice in the menu. Again, I didn't feel it was a necessary portion of the program, and I suspect that it has to do with a difference between upgrades. In fact, the "user" option is so rarely mentioned, it would be very easy to miss in the manual.

The mail labels and print report options from the open database submenu offer the user the two hardcopy options that are most needed. The mailing labels option will let you load, create, change, and save a layout that is suited to your labels and the fields they will contain. When creating a label layout, the prompts will ask questions regarding number of printed lines, number of lines between labels, number of labels across, and the field information that will be on the labels. The prompts make printing labels very easy, and these layouts can be saved to reuse. The print report option, again, will let you create, change, load and save layouts to disk. The layouts hold the following information: number and text for header lines, number of columns, information contained in columns, the selection process to choose which records will be printed, and the ability to calculate certain columns (for instance, a total at the end of each report). The manual explains this all clearly, giving examples throughout. The ability to select records (which is also available with the mail labels option) has an entire chapter devoted to it in the manual.

The method the program uses to guide the user through the selection statements is quite unique, and highly useful. When creating selection statements, the program will first ask if you wish to select certain records. After pressing <Y>es,, the screen will list all of the fields in your database, and begin the selection statement with: "If (..." and instruct you to enter the field number of your choice. The manual explains that the selection statement is made up of four parts with the form: "IF (field name) <equality> (argument) <connector>". After you enter the field, the statement will reappear, this time including your field name and your choice of 9 equity types. When you choose your equity type, the program again incorporates your choice into the statement, and requests your argument, which can be either alphabetical or numeric. Finally, you will be prompted for a connector, which is either "and", "or", or "then print". An example of a final selection statement might be: "IF Last Name equals "Smith" then print". I found this method of explaining and guiding the user through selection statements extremely helpful and valuable as a logic tool as well as a practical method to use the database. Several examples are also included in the manual.

Some other impressive features included are: complete instructions for creating sequential files to be read by a word processor, as well as easy to understand prompts to guide you through, complete information on translating a database to a different format, an option within the regular part of the program to help recover corrupt files, tables in the appendix to give you an indication of how many records will fit on certain disks under certain conditions, a command summary appendix, a full index, and technical support. The manual also suggests that data format disks will be made available for such tasks as payroll, accounting, inventory, check book balance, and other categories.

The manual is fairly well written, although my only complaint is that some of the spelling and grammar need to be cleaned up a bit. I always understood what the author was trying to say, and found that the information provided was adequate and useful.

WOODSoftware will take phone calls for troubleshooting if you are having difficulty with the software, if you have sent in your registration card. Backup copies and free updates for two years are available for $10.00. (Since the program is unprotected, this option may not be worthwhile, unless updates are very fine, although $10.00 is not a bad price for backups compared to some companies). Record Master for the 128, at $49.95, is a good buy for excellent productivity software. I would recommend this program to anyone looking for a flexible, easy to use database program. To get a copy of this program, write to WOODSoftware, P.O. Box 16193, Wichita, KS 67216, or call 316-529-1861.

Making Waves *by Bill Nicholson*

Music on the Commodore 128 is much easier than on any previous 8-bit microcomputer. The Sound Interface Device, usually referred to as the SID chip, plays three music sounds simultaneously, with a wide variety of tonal variations available. In fact, with the PLAY, SOUND, FILTER, and ENVELOPE commands readily available from BASIC 7.0, you can re-create any sound or noise that you can hear. Granted, the more difficult the sound (like a dog barking, a human voice, or the wind) takes great skill, and is much easier on many other micro computers and synthesizers. But, working with it can prove to be quite entertaining and rewarding.

A quick introduction to sound is necessary here. All sound, if you use a microphone to record it and play it through a sound processor or oscilloscope, has a characteristic shape and texture to it, each sound as unique as a fingerprint or iris pattern. The way the computer is able to describe these sounds is by using the ENVELOPE command. The ENVELOPE command is generally used to create tones that do not reside in the computer. You assign the sound an envelope number, which replaces the preset within the computer. The format for the ENVELOPE command is:
ENVELOPE n,[a][,d][,s][,r][,wf][,pw]
where:

```
 n = the envelope number (either a preset or a new setting; the presets I will explain later)
 a = the attack rate (the level of sharpness that a tone begins with [0-15])
 d = decay rate (the speed that a tone loses the sharpness of the attack [0-15])
 s = the sustain rate (the duration of the tone [0-15])
 r = the release rate (the sharpness that tone drops off [0-15])
wf = the description of the wave form:
     0=triangle  1=sawtooth  2=variable pulse (square)  3=noise (white and pink)  4=ring modulation
pw = the pulse width (how far the tone rises and falls from the base tone).
```

All of this is extremely confusing to beginning sound engineers, since a lot of the descriptions seem very similar to one another, and what in the world is a wave form? Let's take a look at the most common envelope (the preset I mentioned before), the piano. The piano preset, number 0, does not sound much like a real classical grand, or even the old upright you learned on. To my ear, it sounds amazingly like a pianoforte, which is the most recent ancestor of the modern piano.

The envelope of the piano is as follows: ENVELOPE 0,0,9,0,0,2,1536. The first "0" tells the computer to look for that envelope in its ROM, and the rest of the description is superfluous, but we leave it here to show how it is put together. The second "0" is the attack. This would be a very fast attack, a sharp strike. The "9" is the decay, and this is not a fast drop-off (0) or a slow one (15). The next "0" is the sustain, here very short (very good for staccato, or clear-cut, notes). The last "0" is the release rate, again fairly abrupt. The "2" describes the wave form, here a square wave. This is probably the most confusing description of sound to beginners. The word "square," "triangle," and "sawtooth" all describe how the tone looks when displayed on an oscilloscope, a tool used to diagnose electronic and physical movement. The square wave is generally a percussive string or wind instrument, and a sawtooth wave usually describes reeds and picked stringed instruments (guitar, double bass), and triangle wave is a purer, cleaner form of the sawtooth wave, that describes the clearer instruments like flute, calliope, and cello. The primary thing to remember when describing a tone with one of the terms above is that most instruments are combinations of the different wave forms. And, by using the ADSR wave form (the acronym for the Attack, Delay, Sustain, and Release rates) one can custom-tailor an existing parameter to create a similar, but radically different instrument (the wave forms of organs and violins are amazingly similar).

The resident sound envelopes in the C-128 are: 0=piano, 1=accordian, 2=calliope, 3=drum, 4=flute, 5=guitar, 6=harpsichord, 7=organ, 8=trumpet, and 9=xylophone. From these you can see a fairly wide variety of instruments, but they form the basis for many others. And, by changing the ENVELOPE parameters, you can redefine these to anything. After defining the parameter, you insert the envelope in the PLAY command.

To hear an example of the built-in tones of the C-128, enter the following short program:

```
10 FOR N=0 TO 9
20   A$=STR$(N)
30   A$="V104T"+A$+"U8X0"
40   PLAY A$
50   PLAY "V104QCDEFGAB05QC"
60 NEXT N
```

This will play a standard octave 9 times, each time with a different default envelope.

Partner 128 *by Avonelle Lovhaug*

As a computer addict who uses my computer for just about everything, I find that I am often annoyed by the problem that one piece of software simply will not do everything that I need it to do. This is always very apparent when I am in the middle of a large database, and I realize that I really need to schedule an appointment with my doctor. Rather than loading up my schedule program, I will probably wait until I'm finished working with the database; this generally means that I will forget to schedule the time. However, Timeworks evidently understands what it is like to be thinking about 10 or 15 things at once, and has developed Partner 128, software that will help ease this difficulty.

Partner 128 is a cartridge that supplies its owner with a built-in calculator, memo pad, appointment calendar, address/phone list, a screen dump function, and a method of temporarily freezing your keyboard. All of these functions are available at the touch of a button located on the top of the cartridge. When you press the Partner 128 button, your current software is frozen temporarily, and the Partner menu is displayed in a window. The menu offers the previous examples as choices, plus a place to set up your current drive and printer device numbers, as well as access to DOS commands.

The appointment calendar is an important part of the Partner package. The calendar offers monthly calendars up to December 1999. When a day is selected, the program leaves the user ample space to record his "to do" list, appointments for the day, and key tasks. Unfortunately, because screen space is limited (especially within the window) the only thing which appears on the monthly calendar is the title for the day, which can consist of only 10 characters. The appointment calendar can also be printed, and you are given a choice of printing a single day, a week, or the entire month. The edit features within the calendar include the ability to copy information from one day to another (within a single month, only) and also to clear the entire display at once. Also, a disk directory can be obtained here, as well as just about anywhere in the rest of the applications. When you are finished with the calendar, hitting ESC once will take you back to the Partner menu, and once more will take you back to your previous software. Your calendar information is stored in the cartridge RAM. However, before powering down, you should save this information to disk.

The memo pad option offers the ability to quickly write and print a brief note or letter. It operates very much like a rudimentary word processor, offering tabs, insert and delete, and word wrap. The Partner manual suggests that the user could place help screens for his favorite software in the memo pad, to be recalled later when needed from Partner. The memo pad area can be printed in two ways. First, when you have finished with your memo, you can use the print option by pressing F7. Partner also allows you to make your printer into a typewriter, which will print each line immediately after pressing <RETURN>.

Another interesting feature of this software is the address and phone list. This is a simple database routine which will save an address and phone list to disk, to be recalled quickly within Partner. Partner also features the ability to sort this list by any field (for instance, by zip code) and print this information or autodial your phone (if you have an autodial modem). When printing the information, you have a choice of printing the address records, printing labels or envelopes, or printing phone numbers. This is one of the best features of Partner 128, as it makes whipping out a quick address label or envelope a snap. Also, this information can be saved to disk. If you only need a simple database to maintain names and addresses of friends or relatives, this may be all that you ever need buy. Partner 128 can hold up to 60 addresses at a time in any given file, but you could save them to different disk files if you wished. One problem with this database is that it is designed for a United States address format, and foreign addresses will not generally fit well in a typical city, state, zip code format. This is especially true since you will probably want to include the name of the country the person resides in, and there is no feature within Partner to include this information. This may not be important to the home user, but for the business user this is of vital importance, unless it is a VERY localized business.

An additional option of this software is the calculator. When selected, the calculator option displays a calculator in the Partner window. Operations are performed in a manner similar to a real calculator. To add or subtract, you must use the "+" and "-" keys, respectively; to multiply or divide, use the "*" or "/" keys. The up-arrow key raises a number to a power, and a colon will change the sign of a number. This calculator also has the ability to store a number in temporary memory, to be recalled later. One final feature of the calculator option is that your printer can print out the calculator's information in "paper tape" fashion.

There are three other features that Partner 128 offers. For those persons who own a 1541 drive, Timeworks claims that the "Swiftload" feature of Partner will allow the 1541 to load programs as fast as a 1571 drive. Although I didn't have an opportunity to test this with any 1541 drives, I do know that I had no problem loading any software with this cartridge installed. However, if you are an extensive user of Paperclip, BobsTerm Pro, or any other program which requires a dongle in joystick port #2, I suggest that you find a "Y" connector, as Partner 128 draws some power from this joystick port by a cable which runs from the cartridge into the port. (Timeworks is offering these "Y"

Partner 128 Review Continued

connectors for a nominal fee.") Partner 128 also has a built-in screen dump utility which will let you print your current screen (no graphics screens, however). If you have had trouble with people accidentally bumping your computer when you were away from the keyboard for a moment, thus losing valuable data, Partner 128 also has a keyboard locking option which will freeze your keyboard until you type in a password which you choose. This option may be worth the program's price itself!!

Despite the fact that I really enjoyed using Partner 128, I found I did have some problems with it. One of the most serious is the way Partner calls up disk directories. For most of the options, you may either get a disk directory one line at a time, or you may see it all at once. However, the more convenient method generally destroys what ever you may be working on currently in Partner, for instance, a memo in the Memo pad. Another difficulty I had was that on one occasion, my cartridge worked its way loose from the computer, and when I pressed the cartridge button, it froze my software, but refused to call up the Partner menu. This meant I lost all of my data. One final quirk I found with Partner was the way it used the <INST/DEL> key. Generally, this key will erase characters to the left of the cursor. Instead, Partner uses this key to delete the character that the cursor is on, moving everything that is right of the cursor one to the left. I found this occasionally annoying.

The manual for Partner is concise and well written. Warnings are blazened everywhere with notes like: "DO NOT call up Partner if the disk drive is running", as well as some suggestions about when not to use Partner with certain software. Apparently database software tends to be the easiest to destroy, since this type of software tends to have heavy disk access. This is unfortunate, as I believe this software is definitely more useful for the business user, not the home user, and database software is an important part of business computing. Also, the manual clearly states that Partner will not work with Vizawrite or any other cartridge based software (forget using your RAM expansion), nor will it work with any 40 column programs or when there are high-resolution graphics on the display.

Timeworks seems to be very dedicated to this product. In addition to the fact that Partner 128 comes with a demonstration disk that gives the user some examples of how the program can be utilized, the manual also suggests that some of Partner's features can be re-programmed to create other useful utilities. This is only available through machine language, but if you are interested, Timeworks is offering instructions for this option, complete with a disk, documentation, sample files and a sample utility. This also will register the interested party as a user of the utility documentation, which will give them the option to receive help with utility writing from Timeworks.

If you are looking for software that can be coresident with your other programs, and can offer the kind of features I've been describing, I suggest you look into buying Partner 128. I make this recommendation with one qualification, though: please take a look at your most used software, and insure that it is not cartridge-based, or calls up high resolution screens, before slapping down around $50.00 to purchase it. If it is compatible with your usual software, this could be a real deal.

Ingredients of the Best C-128 Software *by Loren Lovhaug*

One of the reoccurring themes in Twin Cities 128 has been the role that quality commercial software has played in the C-128's success story. So far we have been generally very pleased with the C-128 software that has appeared during the C-128's first year in the market place. This is not to say that there haven't been a few dogs, but fortunately the vast majority of C-128 software has been very good. Reflecting upon the numerous winners and the handful of losers there arises a definite pattern or criteria which defines the attributes of the superior products and the pattern of failure inferior ones.

The pattern of success seems to be defined in three distinct areas: 1) Hardware and Software flexibility, 2) Utilization of the C-128 specific features, and 3) Usability and cost.

Hardware flexibility is crucial. The program must be able to perform effectively under a variety of system configurations. This means that the program should have the ability to adapt itself to the equipment I own and not vice-versa. For instance, most good word processors for the C-128 give the user a fairly large choice of printer configuration options to suit the huge array of different kinds of printers on the market today. Obviously, there is always going to be a off brand or newcomer appearing in the market so the better packages also incorporate an option for tailoring configuration drivers in addition to any presets that may be provided. This allows true flexibility because no matter what equipment the user owns it will work and insures that the program will not become obsolete directly as a result of the evolution of new peripherals.

Also extremely important is software flexibility. Flexible software allows the user to make his/her own choices according to his/her preferences. This means you can have your choice of colors, data entry templates, display modes,

Ingredients of the Best C-128 Software Continued

input devices etc. Flexible software also gives you a variety of choices as far as operation and automation goes. A superior example of this concept lies with Precision software's excellent database, Superbase 128. Superbase is perhaps the most flexible product available for the C-128 today. It allows you to completely design your own multi-page input screens and is capable of total automation and customization through its own built in programming language which is very similar to BASIC 7.0. So powerful is this user program facility that Superbase itself can be totally masked from the user, replaced by a customized application program which might be tailored to suit a specific purpose.

Another aspect of flexibility is software data exchange compatibility. Packages like Superbase and Superscript as well as Digital Solution's Pocket Series (Pocket Writer, Pocket Planner, Pocket Filer) allow the user to very easily exchange data between the integrated programs in their "family" as well as with other outside programs. This allows the user to choose the best tools for certain aspects of a task and then port the data over to another program which is better suited for another particular task. There is one rule, however, that one ought to remember about software flexibility, namely: the more flexible a program is, the more complicated package tends to be, meaning that it may take a little longer to learn how to use these packages effectively.

Another factor that makes or breaks a piece of C-128 commercial software is its utilization of the features of the C-128. Perhaps foremost in this category is memory usage. Those of us who have been around computing for a long time realize that 128k of random access memory need not be considered confining. Certainly those programmers who wrote software to fit into the smaller environment of the C-64 should be able to crank out superior code now that the memory constraints have been loosened. But there is always a temptation when you have more memory to write less efficient code, thereby taking precious random access memory away from the user. Over the last year or so we have observed that the most successful software companies in the C-128 world have adhered to the axiom that "less is more" when writing code. For instance, the code for Pocket Writer and Pocket Planner, two efficient yet extremely powerful productivity packages fit entirely in one 64k bank of memory, leaving the user a whopping 64k of RAM for his/her data. Another excellent example of how efficient code benefits the user is Bobsterm Pro which also utilizes an entire 64k of RAM for data (buffer) storage, while offering the user an incredible amount of power and flexibility.

Of course, efficient coding can have some other benefits besides leaving the user with a large amount of memory free for applications. For instance, Superbase and Superscript are written in such a manner that they can both be loaded into memory at the same time as co-resident applications which can share data.

Superior C-128 software designs make use of is the greater speed and storage capabilities of the 1571 disk drive. The challenge here lies in the fact that a software company must of course produce software which is compatible with the 1541 as well to maintain user flexibility (see above). The best programs allow the disk drive to decide whether we are working with a single or double sided disk and whether or not we can activate such wonderful features such as fast serial and burst mode.

The keyboard enhancements of the C-128 offer additional opportunities for the C-128 software developers. Use of the functions, the special keys (i.e. HELP, ALT, ESC, etc.), and the numeric keypad for special purposes can really enhance the sophistication and the ease of use of a product.

The final category which defines the excellence of a C-128 product may in many ways be the most important to the end user, namely usability and cost. Of course a good piece for commercial software should do what it claims to do, but there is more to it than just "working". So much of what makes a good program good, lies under the somewhat nebulous moniker of "style". A good piece of C-128 software should be able to meet the user's requirements for functionality while still being a pleasure to use. This means that the software should incorporate tasteful screen displays and outputs and a well conceived (read logical) command structure or menu system. The documentation should be clear and provide examples as well as command summaries and reference sections and must include an index as well as a table of contents.

And of course price plays a very important role in the success and failure of a C-128 software product. We have found that the majority of superior C-128 pieces go anywhere from $30 - $75. One other consideration that ought to be made concerning price is software protection backup/upgrade policies. The most successful software companies in the C-128 world either do not copy protect their software or offer non disk based protection (dongle/key) or offer reasonable backup/upgrade policies.

In the end, of course, the key ingredient to successful C-128 software is the end user, whose personal tastes and pocketbooks ultimately choose the winners and the losers.

CP/M Expeditions *by Todd Madson*

ADD 40K TO YOUR DISKS!

The Commodore 1571 disk drive is a very versatile device, capable of reading and writing to many different types of MFM (Modified Frequency Modulation) disk formats. These formats, such as Kaypro (double and single sided), Osborn (single sided only), Epson (single and double sided), and IBM system 34 CP/M can be used under CP/M on the C-128. All of this is accomplished simply by reprogramming the disk controller. Up until recently, there was no way to format a diskette in one of these alternate formats - you were restricted to using a diskette that had been formatted on one of these other machines. That is until now. Miklos Garamszeghy, a Toronto programmer and journalist has released a program into the public domain that allows the formatting of all the above formats except the Epson. The program, "128 MFM FORMATTER" runs in the native mode of the C-128, and is beginning to show up on bulletin boards all over the country.

Why use any of these formats at all when the C-128 double sided disks holds a hefty 336K? The obvious reason is storage! The Kaypro DSDD format gives you 376K of formatted storage in CP/M mode! Now you can have an extra 40K for programs, utilities, text files, or whatever. While the most obvious use of MFM formatting on GCR (Group Code Recording - the standard Commodore disk format) disks is heavy copy protection, the benefits of extra storage can be seen by everyone. Naturally, there is a drawback: You cannot boot CP/M off one of these diskettes - you must boot CP/M off of a standard Commodore formatted CP/M disk and then switch to the new format once CP/M is up and running. When you place one of these MFM formatted disks in the drive, it will attempt to read it. The system will then sense the type of disk via the number of bytes per sector and sectors per track. If the format is not unique, a highlighted box is shown in the lower-left corner of the screen. You are then required to scroll through the choices via the upper set of cursor keys. To read this disk once, simply press return.

I have noticed that this feature sometimes behaves erratically, as swapping disks will sometimes cause the drive to lock up when reading an MFM disk. At this point, I remove the disk from the drive, turn the drive off, and turn it on again. By this time, CP/M has usually returned me to the system with a BDOS error and it will be safe to attempt reading the disk again. I really like the idea that I can have extra storage on the 1571. Though it may be a bit more work to get this storage, the end result is certainly worth it, especially for a BBS System Operator like me. For more information on reading MFM disk formats in CP/M mode, see page(s) 705-708 in the Commodore 128 Programmer's Reference Guide. Mr. Garamszeghy's BASIC 7.0 program is listed below:

```
10 PRINT "MFM DISK FORMATTER BY M. GARAMSZEGHY"
20 PRINT "OPTIONS:":FOR I=1 TO 5:READ A,A,A,A$:
   PRINT I":";A$:NEXT
30 INPUT "SELECT A FORMAT";F:RESTORE (F*10+90):
   READB6,BS,SD,F$
40 PRINT "FORMAT >> ";F$:PRINT "PRESS <RETURN TO
   CONTINUE>"
50 PRINT "OR PRESS ANY OTHER KEY TO ABORT":
   GETKEYA$:IFA$<>CHR$(13)THENRUN
60 B5=39:S(0)=128:S(1)=256:S(2)=512:S(3)=1024:
   SI=0 :IFSD=2THEN SI=32
70 B1=700RSI:FOR I=0 TO 3:IF BS=S(I)THENB4=I:
   ELSENEXT
80 OPEN15,8,15,"UO"+CHR$(B1)+CHR$(129)+CHR$(0)
   +CHR$(B 4)+CHR$(39)+CHR$(B6)
90 INPUT#15,A:DCLOSE:INPUT"FORMAT ANOTHER <Y/N>";
   FA$:IFFA$="Y" THEN40
100 DATA 8,512,1,"IBM CP/M-86 SINGLE SIDED"
110 DATA 8,512,2,"IBM CP/M-86 DOUBLE SIDED"
120 DATA 10,512,2,"KAYPRO IV  DOUBLE SIDED"
130 DATA 10,512,1,"KAYPRO II  SINGLE SIDED"
140 DATA 5,1024,1,"OSBORNE    SINGLE SIDED"
```

RAM EXPANSIONS AND CP/M

One of my recent acquisitions for my C-128 has been the 1700 Ram expansion. It is a very useful device, and the fact that CP/M supports it as a high-speed Ramdisk (drive M:) is an even better reason to buy one. I've found several ways to increase productivity and increase the speed of things that previously have taken much longer without it. Using the MEX terminal program, I have downloaded programs and files directly to the Ram expansion, and then later dumped the contents of the expansion to my 1571 when off line. This really saves time, since the Ram expansion transfers data at 1 megabyte per second. The 1571, even with sector verification disabled isn't quite this fast!

Another nice application I've been using the expansion Ram for is backing up diskettes. Instead of using virtual drive L: with PIP on my single drive system, which results in many disk swaps, I simply place all the files on the diskette I wish to backup onto the Ramdisk. Then, when I have a blank formatted disk ready, I dump the contents of the expansion Ram onto the blank disk. Poof! Instant backup. Disk intensive programs work really nice with the 1700, though I recommend either buying a 1750 or upgrading your 1700 to a 1750 (see the July/August issue of Twin Cities 128 on the upgrade procedure) since 128K is really too small for serious applications such as dBase II and the like. Another really neat use for the expansion Ram, while of limited interest to most folks, is the storage of images from a BBS system. While working on the Citadel BBS, the saving of messages is nearly instantaneous! When the board is to be taken down for maintenance, all you need to do is copy the message files to a blank disk for later use. Although the Ram expansion isn't supported as a Ram disk in native mode, the Basic 7.0 commands FETCH,

CP/M Expeditions Continued

STASH and SWAP make it easy for you to "roll your own" routines.

Another little-mentioned trick of the Ram expansion is that unless the computer has been shut off or the Ram memory emptied, the files you've placed there will stay there. Even after a RESET, you can check the contents of the expansion, your files are still there. The expansion Ram unit comes with a 25 page manual that covers native and CP/M mode use of the device, plus a diskette with some mind-blowing demos on side a, and the newest release of the CP/M operating system on side two, along with some utility programs that allow for optomization of the CP/M side of the machine. You also get your standard Commodore 90 day warranty card. If you plan on making the most of your machine, the Ram expansion units come highly recommended.

A LITTLE HELP WITH PIP

PIP is one of those transient utilities you will find yourself using a lot under CP/M. PIP stands for "Peripheral Interchange Program", although many people find themselves renaming it "COPY", which is its basic function.

PIP can copy single and multiple files from one disk drive to another, or you can use a single disk and do swapping using virtual drive E: for this purpose. Virtual drive E is random access memory which is configured like an imaginary disk drive. PIP can copy the files from an entire disk to another, copy a text file and send it to the printer or other output device. PIP can, in the words of the Digital Research documentation, "transfer data from a logical input device to a logical output device, thus the name Peripheral Interchange Program".

PIP can even rename a file or files after the copying has been completed. But first, lets get down to basics with the program. To copy the files from one diskette to another using a single 1571, all you need do is the following (provided the PIP.COM program is on your diskette): PIP E:=A:*.*. Note the wildcard used for the file name and extension fields. You can even copy all of the COM files on one disk to another by doing PIP E:=A:*.COM. You will be prompted to swap diskettes by the system with highlighted inverse bars at the lower portion of the screen. You will note the files being copied will be printed on the screen as the process continues. This will be a somewhat slow and tedious process if you have a full CP/M disk in the drive.

For those lucky users with two drives, all you need do is the following: PIP B:=A:*.* This will then copy all of the files from the disk in drive A to drive B. But what if you want to copy just a single file from one diskette to another? Simply substitute the filename and its extension for the wildcard like this PIP B:=A:FILE.EXT.

Renaming a file after copying is another neat feature. You can copy it to the same drive/ usernumber, or a different drive/usernumber. But make sure you have enough storage on the disk for the file, as PIP has to have enough room for the new copy before it deletes the old one. This is accomplished like this: PIP B:newfile.txt= A:oldfile.txt. This copies the old text file from drive A: to drive B: with the name NEWFILE.TXT.

PIP even allows you to combine files by concatenating them into one massive file. One thing you should note is that PIP will stop when it reaches a control-z in a file, which is usually the default end-of-file character. You should use the O option for copying machine code files in which a control-z might be a valid data item. Here's an example that lets you concatenate three files together: PIP BIGFILE=FILE1.FILE2.FILE3. The three smaller files should be on the default drive A.

You can also copy files to and from devices in the system, but the file must contain printable characters. You can type a control-c to abort any PIP option. The logical devices you can use for PIP are CON:, which is your console input-output device (your keyboard), AUX:, which is any auxiliary input or output device, and LST:, which is the destination device assigned to the list output device (usually a printer). Here is an example of how to send everything you type to the printer (other than the usual control-p): PIP LST:=CON. You can terminate this by typing a control-z. You can also send whatever you like into a textfile like so: PIP A:TEXTFILE.TXT = CON: this will then send any text you type into a file. You end this with a control-z. Here is how you can send one of your files to the printer using pip: PIP LST:=A:BBSLIST.TXT[T8]. The T8 option expands any tab characters to the nearest column that is a multiple of eight.

Ram Packing *by Loren Lovhaug*

Last month, caught in the fury of technolust, I told you a little bit about my new 1750 Ram expansion. This month it is time to "get down to business" with the 1700/1750 Ram expansion by exploring how to use them in and with our Basic 7.0 programs.

Ram Packing Continued

First off let's try to better understand exactly how Basic 7.0 uses its own internal random access memory and accesses the memory in your Ram expansion. As I said last month the Ram expansions give you indirectly accessible random access memory. This means that the operating system of your computer does not control this random access memory in the same way it accesses and allocates the 128k of random access memory installed within your C-128. The operating system of the C-128 can only "see" (directly access) the C-128's internal random access memory. This memory is divided into two 64k sections called BANK 0 and BANK 1. The C-128's Basic programming language uses these 64k banks for two distinctly different purposes. BANK 0 is used by Basic for the storage of various system information and data and the text of your Basic programs (the vast majority of this space is reserved for the text of our Basic programs). BANK 1 on the other hand is used for the storage of variables in your Basic programs. We will discuss which sections of these banks are particularly useful later in this article.

Since the C-128's operating system can only "see" the contents of the memory in BANK 0 and BANK 1, the designers of the C-128 have given you the capability to copy or replace part of or the entire contents of either BANK to/from the external Ram in the expanders. This is accomplished through the use of three Basic 7.0 commands specifically to transfer data between the internal memory of your C-128 and the external memory of your Ram expansion.

The STASH command copies a specified section of the C-128's internal memory (BANK 0 or BANK 1) to a specified part of the external Ram expansion memory. This copying process does not disturb the contents of the internal memory. The FETCH command performs the exact opposite function of the STASH command, copying the contents of a specific part of the external memory to a specific section of BANK 0 or BANK 1. Like the STASH command the FETCH command does not disturb the source memory for the copy, in this case the specified section of the expansion memory. The SWAP command exchanges a section of internal memory with external memory. This means that with the SWAP command both internal and external memory ARE altered. The parameter list for each command follow exactly the same pattern, namely:

NUM BYTES, INTERNAL LOC, EXTERNAL LOC, EXT. BANK

Where: NUM BYTES - is the number of bytes involved in the operation, INTERNAL LOC - is the beginning location in BANK 0 or BANK 1 (BANK 0 is the default) involved in the operation, EXTERNAL LOC - is the beginning location inside the memory expansion involved in the operation, EXT. BANK - is the bank inside the Ram expansion where the operation is to take place.

After looking at the above parameter descriptions you may have wondered what all of that means. Let's consider a specific instance such as:

```
SWAP 1024,1024,0,0
```

This SWAP command tells the C-128 to exchange 1024 bytes (parameter 1) beginning at location 1024 in BANK 0 of the C-128's internal memory (parameter 2) with the expansion memory beginning at location 0 (parameter 3) in bank 0 in the expanders. The last parameter is necessary because the memory inside the Ram expansion units is organized into 64k sections, sometimes also referred to as banks (be careful not to confuse these external "banks" with the internal BANK 0 and BANK 1). The 1700 Ram expander is organized into two 64k banks numbered 0 and 1, while the 1750 Ram expander is organized into eight 64k banks numbered 0 through 8. That's all there is to it.

About this time, I envision many of you scratching your heads and saying, "Okay, I understand the principles behind utilizing the Ram expansion from Basic 7.0, but I am still not sure how to actually do the NEAT stuff?" The answer to that question lies in your understanding of the C-128's architecture or your ability to utilize various reference materials on the C-128's architecture. In order to use the Ram expanders effectively in your own programs you must be able to locate the areas in the C-128's internal memory which store or control various operations that you may wish to store and later recall from the Ram expander. This is best done by examining a detailed memory map of the Commodore 128 and studying various explanations of how things work on the C-128. Perhaps the best source for this kind of material is the Commodore 128 Programmer's Reference Guide published by Bantam. But here are some handy memory locations and ideas to get you started:

(memory locations are in BANK 0 unless specified differently)

Ram Packing Continued

40 column text screen (Location 1024 - 2023)
Transfer 1024 bytes beginning at 1024.
(example: STASH 1024,1024,0,0)
Idea: Multiple instantly accessible help screens

40 column graphics screen (Location 7168 - 16383)
GRAPHIC screen must be allocated.
Transfer 9216 bytes beginning at 7168.
(example: STASH 9216,7168,0,0)
Idea: Multiple graphic screen animation
Comments: GRAPHIC screen must be allocated.
Background color: 7168 - 8191 Bitmap: 8192 - 16383

Function keys (Location 4106 - 4351)
Transfer 245 bytes beginning at 4106.
(example: STASH 246,4106,0,0)
Idea: Instant function key overlays for menus
Comment: Transfers all function keys regardless of definition or lengths

Sprite definitions (Location 3584 - 4095)
Transfer 512 bytes beginning at 3584.
(example: STASH 512,3584,0,0)
Idea: Super animation with multiple definitions
Comment: Use of multiple backgrounds with multiple moving sprite sets would produce fantastic results.

Basic Program text (Normal Location 7168 - 65279)
(Location w/graphics 16384 - 65279)
Transfer number bytes necessary for Basic text.
The end of Basic text can be found as follows:
EB=(PEEK(4625)*256)+PEEK(4624)
So sample transfer might look like this:
STASH EB-7168,7168,0,0
Comment: This is the technique one would use for storing multiple Basic programs in the expansion. First you load the program into memory using the DLOAD command, then find the end of the Basic text in BANK 0 using the formula above and STASH it in the Ram expansion. Be sure to keep track of the location and bank where your programs are stored so they can be recalled.

Basic Variables storage in BANK 1
This is a little tricky because the STASH, FETCH, and SWAP commands are designed to operate on BANK 0. although the operating system can be convinced to let you access BANK 1 with them. The method involves manipulating bits 6 and 7 of the MMU's Ram control register. These bits are used to control the C-128's VIC Ram bank pointer as well as the C-128s DMA bank pointer. To find the location in BANK 1 of specific variable use the POINTER function. The following example sets the register so we can access BANK 1, performs a STASH and then returns the default bank for the STASH, FETCH, and SWAP commands back to BANK 0. It is important to return these bits back to their original values after performing a STASH, FETCH, or SWAP or your 40 column screen will display garbage.

```
10 POKE 54534,PEEK(54534) OR 64:REM set to bank 1
20 STASH 1024,21,0,0: REM perform stash
30 POKE 54534,XOR(54534,64):REM set to bank 0
```

Machine Language for Commodore Computers *by Loren Lovhaug*

It has been said that, "Real programmers do not code in high level languages, instead they gut it out on the level of the machine." Well I do not really believe that, but I must admit a certain admiration for those individuals who are proficient machine language programmers. There are many of these guys who know memory maps better than their home town street maps and can do binary, octal, and hexadecimal conversion in their head at a blink of an eye. But in the Commodore world, mention ML and one name comes to mind as standing head and shoulders above the rest, the Grandfatherly Guru, Jim Butterfield. I have met Mr. Butterfield a couple of times and attended his seminars while stumping Twin Cities 128 around the country, and I can personally attest that you will be hard pressed to find a better teacher or someone with more expertise. That is why I was overjoyed to learn that the GURU had recently

Machine Language for Commodore Computers Review Continued

completed a revision of his venerable "Machine Language for Commodore Computers" that included extensive coverage on C-128 ML programming. I own a copy of the original text, and for my money there was not a better text for beginners, until now. This book is a must have for anyone who serious about learning to program in ML.

What makes this book superior is the way in which Mr. Butterfield presents a very complex subject in a manner which is easily understandable and nonintimidating. This text is divided into 8 chapters which are organized in a logical sequence from first concepts to more advanced concepts. "Machine Language for the Commodore 64, 128, and other Commodore Computers" presents material on such topics as machine language arithmetic, addressing modes, and kernal routines that is not only informative and complete, but also in a manner that makes it ideal as reference guide long after the reader has mastered the concepts presented within. Each section is complimented with its own challenging, but worthwhile set of exercises. The answers to these exercises are also included in the text with extensive commentary on the solutions making this book ideal for review purposes. Mr. Butterfield has also included a rather hefty appendix section which contains a wealth of material on every Commodore computer that has been produced to date, including memory maps and technical overviews, and superior illustrations. This technical history of Commodore computing is very interesting and of great value to anyone who owns more than one make of Commodore computer.

I have found only one minor flaw in this text. This is the slight inconvenience of having to often consult the C-128 appendix in the back of the book when working through the main body of the text to get to the C-128 commentary and exercises. However, the book makes it very clear within the text when a C-128 owner should consult the appendix, and a simple bookmarker makes this aspect of an otherwise flawless text almost unnoticeable. 377 pages, $16.95, A Brady Book, published by Prentice Hall Press.

Forgotten BASIC *by Loren Lovhaug*

This month's installment of "Forgotten BASIC" explores another aspect of the BASIC 7.0 that while offering a great deal of power and potential to BASIC programmers is often ignored.

Many maids and housekeepers refuse to do windows but that is no reason why C-128 programmers should not do WINDOWs. The WINDOW command provides a very nice method for visually organizing output to the screen. The WINDOW command is used to specify the position and dimensions of the area in which output can be placed onto a text screen. The WINDOW command has five parameters which are as follows:

```
WINDOW X,Y,A,B,C
```

X represents the upper-leftmost character position of the area in which text is to be displayed. When the forty column screen is active X may range from 0 - 39; however, when the 80 column screen is active X may range from 0 - 79 (zero represents the absolute leftmost character position). Y represents the line ranging from 0 at the top of the screen to 24 at the bottom on which the upper-leftmost character of the display area may be placed. A and B are similar to X and Y respectively except that they represent the horizontal and vertical positions of the lower-rightmost position of the display area. The last parameter C, indicates whether or not the newly defined display area should be cleared or not. (1 clears, 0 does not clear).

The WINDOW command is destructive, so it cannot be used for true overlaying pop-down menus, but once a window is defined text will scroll within the defined display area. Another interesting aspect of the C-128's WINDOW facility is the fact that BASIC 7.0's CHAR command adapts its display coordination to the currently defined WINDOW. Type in and study the following example to get a better understanding of the WINDOW command and some of the abilities of the CHAR command:

```
100 ws=RWINDOW(2)-1
110 DEF FNr(x)=INT(RND(1)*x)+1
120 t$="Watch these windows get dirty!"
130 FOR i= 1 to 10
140  x=FNr(ws-1)
150  y=FNr(12)
160  a=x+FNr(ws-x)
170  b=y+FNr(12)
180   FOR j= 1 to 5
       WINDOW x,y,a,b,0
200     CHAR 1,FNr(RWINDOW(1)),FNr(RWINDOW(0)),t$,1
210     SLEEP 1
220     WINDOW 0,0,ws,0,1
230     a$=""
240      DO WHILE a$=""
250       GET a$
260       PRINT"press any key!      ";
270       LOOP
280    NEXT j
290 NEXT i
```

Forgotten BASIC Continued

This program upon execution will place the message: "Watch these windows get dirty!" in various places on the screen. This is how the program works:

Line 100 uses the function RWINDOW to decide whether you are executing this program on the 40 or 80 column screen. RWINDOW returns various information about the current display area depending upon the argument you provide. If you use an argument of zero with the RWINDOW function, the function will return the number of vertical positions in the current display area. If you use an argument of one, the function will return the number of horizontal positions in the current display area. And as we see in line 100 an argument of two returns the display mode we are in (40 or 80).

Line 110 creates a function which chooses a psuedo random number between 1 and whatever number is used as the argument of the newly created FNr function. Line 120 simply defines the string variable T$ to our message. Line 130 defines the primary loop in this program. This loop in reality keeps track of how many windows we have placed on the screen. Once we have placed ten windows and their messages on the screen this program terminates. Lines 140 through 170 are used to define randomly chosen dimensions of each new window. Line 180 defines a nested loop which will control how many messages are placed in each window. The WINDOW command in line 190 defines the current display area to the one chosen in lines 140 through 170. Interestingly enough this WINDOW command will also allows us to "jump between" windows as the "j-loop" executes. The CHAR command in line 200 places our message in a random position within the defined window. You might notice that the message is printed in reversed video, this is because the reverse video flag is set to one (note the last parameter of the CHAR command). You might also notice how the text occasionally scrolls. Line 210 defines a new window which will be used for a one line travelling message (this is the window we "jump between" as the "j-loop" is executed. Line 230 defines the string variable A$ as a null string so we may use it as a condition in the DO loop beginning in Line 240. This loop continues to display the "press any key message" in the one line display window until a key is pressed. When a key is pressed the program continues until all loop conditions are satisfied.

While the above program does not seem very significant, it does demonstrate some of the neat things you can do with the C-128's WINDOWing capabilities. Why not try them out in your programs?

XREF 128 *by Bruce Jaeger*

Cross-reference ("XREF") programs have probably been around since the first high-level computer language was written. An XREF program reads your source-code, then prints out a listing of the locations of the use of all variables, GOSUB's, etc. This can be a very useful debugging or documentation tool. Suppose, for example, your program keeps coming up with an answer of "X=101," a result that you know to be nonsense because X was only supposed to be 0 or 1. Using a program like XREF 128 might tell you that you also inadvertently used X in a counting loop ("FOR X=1 TO 100") somewhere else in the program--possibly saving hours of frustrating debugging, especially if your BASIC program is quite large.

Abacus Software's XREF 128 is a powerful cross-reference program. In fact, its power can be somewhat frustrating! In its unmodified, all-options selected form, it generated a lengthy 10-page printout of my 55 block long program, including such "useful" information as 3/4 of a page telling me wherever I used an "=" sign. Fortunately, programmer R.C. Wainwright included a program on disk to add to (or subtract from!) the BASIC keywords and symbols that XREF 128 looks for; I simply deleted such super-common keywords as "=," "+," "-," etc. The program also gives you the option of not listing all the BASIC keywords, and of not listing what it calls "the rest," which are line numbers and variables. You can specify line ranges for either option, further reducing the size of the printed output if desired.

XREF 128 lists all of the numeric constants ("X=37.5") used in a program, as well as all target line numbers for GOTO's and GOSUB's. However, it inexcusably makes no distinction between the two in the listing! It treats "X=100" and "GOTO 100" as the same!

Powerful as a good cross-reference program can be, what is generally more useful when debugging a program is a good "FIND" command, which you can use when actually working on the program. I've done a good bit of commercial programming, and I can count on the fingers of one hand the times I've needed a cross-reference program.

XREF 128 is heavily copy-protected, somewhat user-hostile and cryptic in its screen prompts. (The Commodore 128 has memory to burn for user assistance--why won't more programmers get a book of matches?) The disk is auto-booting, that latest abomination to hit the Commodore camp. And, you have to turn the computer off to get out of the program, surely a lot of nonsense for a simple cross-reference utility that you can get from the public domain for free.

Sparrow's Slick Tips *by Sparrow James*

00101010: Indented Listings: It is often desirable to indent your Basic program listings to make the program's structure more apparent. However, this cannot simply be done by adding blank spaces between a line number and the command on a particular line since the Basic parser will "throw out" your "unnecessary spaces". Fortunately there is a way to easily create indentations. Simply put a graphic character (the characters printed on the front edges of your keyboard) following the line number in your program and put as many blank spaces as you like and type your command and press return. When you list the program you find that the graphic character will have disappeared but your indentation will remain intact.

00101011: Basic Merger: As I have said many times in this column, the C-128's Basic 7.0 is one of the richest Basics available for any microcomputer. But there is one feature noticeably absent: the ability to merge program files from disk. With this capability you can write a set of often-used generic subroutines (sometimes referred to as "primitives"), save them to disk and later merge them into other programs, thereby saving the "reinvention of the wheel syndrome". Many authors have attempted to simulate a program merge on the C-128 by loading a program, then moving the start of Basic pointer to just beyond the end of that program in memory, which will convince the C-128 to load the second program in just after the first in BANK 0. Then, once the second program has been loaded into memory, the start of Basic pointer is moved back to the beginning of the first program thus convincing Basic that you now have a longer "merged" program in memory. This is all well and good, except that this type of merge is often incredibly inconvenient. The biggest drawback is that this type of merge requires that the subroutine you are merging be numbered with line numbers greater than those of the first program in memory. If you don't obey this rule, Basic will do some very strange things with your new pseudo-merged program. Fortunately there is a better method:

1. Save the program or subroutine you wish to merge to another program to disk as a sequential file like this:
   DOPEN#1,"filename",W:cmd1:list:print#1,"DCLOSE":print#1,"":DCLOSE
2. Load the program that you wish to attach the above program to.
3. Type the following and press return: DOPEN#1,"filename":SYS 65478,0,1:LIST

You will now have a perfectly merged program (you should see it on the screen). Don't believe me, LIST the program and see for yourself! Here is how it works: Step 1 saves your Basic program listing to disk as a sequential file, and then adds the command DCLOSE at the end of the file. By putting this entire sequence on one direct mode line we will avoid having the "READY." message placed in our file at the end of the list. Step 2 brings in the program you are about to append to. Step 3 opens the sequential file and calls the kernal routine CHKIN which in this case tells the computer to use logical file #1 as an input device (the default input device is the keyboard). Then the program is listed, but since the DOPEN'd file is the input device your listing is in effect "typed in" from disk, thereby merging it with the existing program in memory. The DCLOSE we placed at the end of the file in step 1 in turn closes the file automatically for you. This method for merging programs is much more convenient because it allows you to merge your routines freely without having to make sure that their line numbers are greater than those of the program you are merging to. (One caveat however, this merging technique will also replace lines in the program with lines for the "merge" file if they have the same line number.) Another side benefit of this merge technique is that it allows you (if you so choose) to use a word processor to edit Basic programs or subroutines. Consider altering programs with the powerful search and replace and macro capabilities of most C-128 word processors.

00101100: Creating a new default printer driver with GEOS: Many GEOS users have expressed to me sheer aggravation over the apparent inability to set up your own default printer driver. It is irritating to have to select your printer manually each time you begin a work session with GEOS. The GEOS manual suggests vaguely that GEOS will select automatically the first printer driver on the diskette, however we have found this only to be true with the BOOT disk and NOT your individual work disks. The solution is to rearrange your boot disk so that the printer driver of your choice is first on the BOOT disk, then your printer driver will be selected as the system's default printer driver.

00101101: Restoring your Function Keys: It is often desirable to redefine the C-128's programmable function keys for use in your Basic programs. However many programmers have a bad habit of not restoring the function keys back to their default values before the program terminates. This is unfortunate since it can very easily be done without even having to go through the drudgery of having to use the KEY command eight times. All you have to do is execute this simple loop: FOR X= 0 TO 76: POKE 4096+X,PEEK(52904+X):NEXT X

This loop reads the default definitions from the C-128's read only memory and POKEs those values in the function key buffer.

Rumor/Opinion/Mayhem *by Loren Lovhaug*

Stomp out DOOM and GLOOM

Lately I have observed that a large contingent of Commodore owners are suffering from a severe form of mental depression and paranoia. It seems every user group newsletter I read, every bulletin board system I log on to, every gathering of Commodore users I meet with, are dominated by lengthy discourses by people who are convinced that the golden age of Commodore computing has come and gone, and the end is most certainly near. For some bizarre reason they are compelled to feel bad about themselves, their computers, Commodore, and income tax reform. Well, I hope I can help you feel better about the first three items...as for income tax reform, BE CONCERNED.

Maybe I am living in a euphoric dream state, but it seems to me that Commodore computer owners, especially Commodore 128 owners, have never had it so good. Consider the following:

1. Commodore 128 owners are members of some of the largest microcomputer communities in the world. Consider the fact that Commodore has sold approximately 6 million C-64 compatible machines (C-64, SX-64, C-128), and approximately 1.1 million C-128s worldwide. Contrast those numbers with the recent announcement that IBM, after four years of production, has just passed the three million sold mark with the venerable PC. In addition, the 1.1 million C-128s out there are part of an immense group of over 20 million machines in operation which can run CP/M (counting IBM PCs/clones and Apple II series machines which can be made CP/M compatible). This means if the Commodore were to somehow disappear off the face of the earth tomorrow, there would be a lot of commercial and public domain software, as well as user support from several different sources to sustain the existing C-128 user base for years to come.

2. Contrary to popular belief software development in C-128 mode has been excellent. The current commercial software list totals more than 120 titles. With just a few exceptions, these packages have been quality products that offer a great deal of value as well as variety. To date, we have seen productivity packages, programming utilities and languages, graphics packages, and music programs, and telecommunications programs. In addition to the commercial C-128 mode development there has been a constant flow of very good public domain software surfacing on a variety of local bulletin boards and national telecommunications networks like GEnie.

3. We have recently received letters from people who are genuinely excited about their C-128s and are using them in: medicine, the practice of law, real estate, physics, the instruction of college level mathematics, and as sophisticated terminals to mainframe's for high level software development. This demonstrates my personal credo: It is not the machine that makes the man or woman great, but the man or woman who make the machine great. The people using C-128s are making this machine great, expanding the limits of 8 bit technology and proving that serious computing does not have to be limited to those who can afford to be disciples of BIG BLUE.

I am excited about this kind of "doing more with less (mostly less cost) philosophy" and I am proud to declare that we put together Twin Cities 128 each month with C-128s. In fact, we are to the point where we are almost totally C-128 mode, and when GEOS 128 is finally available we will be able to make the claim of not only "proudly produced on a C-128", but "proudly produced on a C-128 in C-128 mode!".

So, when you here people speaking ill of Commodore or predicting an end for the C-128, you just remind them of the above facts, show them Twin Cities 128 and remind them that there are plenty of users who are happy exploiting their tiny C-128s to the fullest and in so doing getting more for their computing dollar than most owners of those supposed "high-power" machines. After all, power is only worthwhile if you USE IT!

HOT RUMORS, GET THEM WHILE THEY ARE HOT

Here are some of the Hot Rumors floating around the C-128 world (listed in order of their viability).

Suer things: Progressive Peripherals (464 Kalamath Street, Denver CO 80204) is about to release a 10 megabyte hard drive called "Device Nine: The Vault" for the C-128 to retail for $895 with twice the speed as the 1571. Also from Progressive is a new multi-feature serial printer interface for the C-128 called "Device 1" which features a 16k buffer, downloadable fonts, and the ability to overlay graphic and text images over one another electronically. Louis Wallace of "C-128 Ultra-Hires" fame seems about ready to release a new BASIC extension wedge for the C-128, tentatively dubbed Basic 8.0, which will offer extraordinary 640 x 200 graphics support (80 column) including special support for 1700/1750 Ram expansions, and VDC RAM upgrades (see the article on upgrading VDC RAM in this issue). Apparently, various Commodore 128 engineers have been working with Louis to make the normal 40 column graphics commands completely compatible with some of Louis's "ultra-hires" enhancements on the 80 column screen.

Good bets: We will at last see the long awaited 1571 Rom upgrades (hopefully before Thanksgiving) rumor has it that they have now passed beta-testing with all of the commercial C-128 software producers who received prereleases and Commodore marketing is working out the final distribution and pricing details. Also from Commodore it appears that the long-awaited C-128 Macro-assembler/editor system written by Fred Bowen and some of the other C-128 Engineering crew will be available for purchase before Christmas.

CADPAK-128 *by Miklos Garamszeghy*

CadPak-128 is a fairly sophisticated drawing program for the C-128 which allows the user to construct detailed diagrams on a 40 column color monitor screen, and then either save the images to disk for later use or obtain a hard copy in several different sizes on a variety of dot matrix printers. The main CadPak drawing screen has a resolution of 640 pixels wide by 360 pixels high. The 40 column video screen is used as a 320 x 180 window on the larger drawing tablet. A view function is, however, provided to allow you to see the entire drawing at once, suitably shrunk to fit the screen. A secondary drawing screen with a resolution of 320 pixels wide by 200 pixels high can be drawn, edited, saved or printed independently of the main screen. It is used mainly for the construction of custom object templates or text fonts. Both screens are scaled with user defined units in either metric or imperial (feet and inches) format. The user enters dimensions in scaled units. Unfortunately, the vertical and horizontal scaling factors cannot be specified separately. CadPak-128 will accept input of drawing coordinates in one of three ways: by moving a drawing cursor around the screen with the normal cursor keys; by using a light pen; and by keying them directly in numeric form. The first two methods cannot be used simultaneously; turning on the light pen automatically disables the cursor control keys. Positioning is a two step process. Rough positioning is performed in 8 pixel increments. The second step gives you one pixel increment control.

The program provides a reasonable selection of drawing functions from an on-screen menu. Most of the usual CAD type functions are supported: LINE, BOX, CIRCLE, FILL, COPY, LETTER, ZOOM, etc. The functions are selected by keying in a two letter mnemonic (displayed on the menu part of the screen). Subfunctions are also selected by keying in one or two letter codes. Two drawing modes are provided: normal and extended. Although the functions provided under each are slightly different, the major difference is that drawing on the normal screen does not allow you go beyond the current screen window while Extended mode allows you to move over the entire 640 x 360 area. For example, if you specified a circle of a diameter that was too large to fit on the current window, normal mode would ignore or clip off the parts that would not fit on the screen, while extended mode would create the entire circle, even in areas that were not currently on-screen. Similar functions can be used for editing the secondary screen. The menu system relies on color combinations of the screen border color and the menu text to indicate what mode or stage of operation the command is in. This is fine, except that it assumes you are using a color monitor. With a monochrome monitor, it was impossible to distinguish between the various command modes.

Text can be entered in any one of four supplied fonts: normal, Tech, 3-D and Old English Script. The normal font (copied from the character set ROM's) can be readily displayed in four sizes: normal, double width, double height, and both double width and double height. The special fonts are loaded from disk files into the secondary screen area. Tech and 3-D are available in uppercase only, while Old English is supplied in upper and lower case. Although they can normally be used only in one size, other sizes can be created using the individual letters as predefined "objects" which can be adjusted in size as required. For anything more than a few characters, this procedure is somewhat tedious, however. The special fonts can be edited with the object/font editor to customize them or create new fonts. A library of other general purpose objects can be created and used as "letters", such as electrical symbols for drawing schematic diagrams or mechanical symbols for drawing process flow diagrams. All objects created this way can be placed anywhere on the main drawing and can be scaled, rotated and/or mirrored about any axis.

More complicated objects can be stored as "templates" consisting of a set of drawing codes and coordinates. This is somewhat similar to a keyboard macro, allowing you to define a shape as a series of lines, points, etc. A demo program is supplied for reading and creating your own template files. CadPak drawing files (both main screen and secondary screen) can be stored on disk as in a series bit image type files. The type of file (such as main screen, font, etc.) is denoted by the first two characters (m. for main screen) of the filename. These filename parameters are added automatically by the program. A main screen requires 8 different disk files with a total of about 130 blocks. The logic behind the 8 files (your specified filename with filename extensions from "a" to "h") totally escapes me. One file of the same total length should be sufficient. It should be noted that since the files are stored as a bit image, the same number and length of files are used regardless of the complexity of the drawing from a blank screen to a very complicated image. The file save function seemed to have at least one minor bug. I purposely tried to save a file without having a disk in the drive. An appropriate message, "drive not ready", was flashed on the screen. When I was ready to save the file, a message kept telling me "file open" error. This could only be corrected by loading in a font file then trying the save again. It was obvious that the error handling routine did not properly close the file before reporting the error.

As a professional engineer routinely involved with design work, I have used a number of computer assisted drafting (CAD) programs on both micro and mini/mainframe computers. I have even written several custom CAD programs for IBM-PC type computers. With my varied background in CAD systems I have mixed views about CadPak-128. On the good side, the program is quite flexible and powerful, once you become familiar with the commands. (I tend not to read software manuals. In the case of CadPak, it is absolutely essential that you do. The on-screen prompting is minimal, while a lot of the drawing functions require you to enter data in a specific order. This order is explained in the

CADPAK 128 Continued

manual. but is not readily apparent from the screen menus.) It supports a fairly wide range of printers (my Roland printer with a Cardco+G interface worked flawlessly with the Epson printer selection) with a reasonable selection of sizes both normal and sideways orientation. CadPak is perhaps the best general purpose drawing program I have seen on a Commodore computer (other than the Amiga).

Now for the down side. My biggest complaint is that the program is so slow. Copying images with the "point extract" option takes several minutes for reasonable sized objects. Mind you, another copying method, "block extract", is much faster, but you lose a certain amount of control over what you are copying. Printing a hard copy is also very slow. The program pauses for at least two seconds between printout lines while it composes the graphics sequence for the next line. The lack of speed seems to stem from the way the program was written. The length of the main program (almost 200 blocks) suggests that it was written in Basic and then compiled. Although compiling can increase the speed of a Basic program, it will never match the speed of a program written in machine language from scratch. My other major complaint is that it does not support a plotter. A graphics plotter is the backbone of all professional CAD systems. (The pixel oriented main drawing screen of CadPak is totally incompatible with the line and vector system used by plotters. Oddly enough, the drawing code system used in the template files could be quite easily adapted to support a plotter.) Other minor points include somewhat erratic operation with a light pen, and random patterns flashing on the screen, especially when working near the edges of the drawing. In addition, although the manual is thorough and includes several tutorials to familiarize yourself with the command structure, it lacks an index. Hunting through the manual to find a specific detail can become a time consuming task.

I would like to have seen CadPak take advantage of some of the advanced features of the C-128, such as the 80 column screen, either as the main drawing screen or to be used as a command screen. Many CAD packages on other computers which support multiple screens use one screen to display the drawing uncluttered by menus and use the second screen to display the menus only. While it is debatable how many C-128 owners would have two monitors (I do), the feature would be a nice professional touch. It would have also been nice to have support for a plotter to produce really high quality outputs.

All in all, while the program is far from perfect, it can be very useful and flexible if you are willing to live with its limitations. The program is suitable for most general purpose drawing, flowcharts, simple electrical schematics, text slides, room arrangement. The best feature of all is that it is perhaps the lowest cost CAD package on the market.

Rebel Assembler 128 *by Bruce Jaeger*

"Hooray!" I thought. "Finally a decent assembler for the C-128!" Commodore has inexcusably been dragging its corporate feet in releasing the C-128 version of their own workhorse Macro Assembler, and I was getting darned tired of switching to C-64 mode for my ML projects, and giving up fast disk access and 80-column capability. Along comes the Rebel Assembler/Editor, from Nu-Age Software. I was thrilled. Then I tried to use it. Now I'm back to waiting for Commodore.

THE ASSEMBLER

It's not that the Rebel Assembler doesn't do the job. I'm sure it probably does. But, try as I might, I couldn't get Rebel Assembler to assemble any of my programs. Oh sure, I could get it to assemble their samples--but not my own code.

With the Rebel Assembler, you can assemble either into Ram (lightning fast!) or onto disk. The "instructions" warned that you shouldn't, naturally, try and assemble into the Ram occupied by the assembler itself. Logically, you'd think that the next sentence should have detailed the Ram used by Rebel Assembler, and what free Ram was available It doesn't, and you're left to guess, and when you guess wrong you hear that aggravating bell tone as the program crashes into the monitor. (Sometimes. The rest of the time it just crashes into hyperspace.)

So, to play it safe, you assemble to disk. This is done by including your target file name after a :F(ilename) directive. Then the assembled code will be sent to a disk file, as well as assembled to Ram. (So then it, of course, crashes again!) To disable that dangerous Ram assembly, you use the :S(yntax) directive. (Although I have no idea what "syntax" has to do with suppressing Ram assembly.)

That's what I did, only my code still crashed the assembler. I was doing something terribly, and probably obviously, wrong. I went to the "instructions," which were produced on a dot-matrix printer by someone who couldn't spell. Most of the instructions concern the C-64 version of Rebel Assembler, which is also included on the disk. I did not care

Rebel Assembler Review Continued

about the C-64 version, but the C-128 information was scattered throughout, so I had to wade through it so I didn't miss anything. I'm still missing something, probably because the "instructions" are so hard on the eyes to read that my attention span was drastically reduced.

But enough. By the time this hits print, I may have figured out my problems, and they'll probably make me feel like an idiot. But my point is, obviously, that the documentation included with Rebel Assembler is miserable. I didn't exactly just get off the bus when it comes to machine code and assemblers, and I found myself frustrated and aggravated by the Rebel Assembler. Imagine a first-time user!

THE EDITOR

What's one of the reasons we like the C-128? The 80-column screen, right? So what does Rebel Assembler give us to write code in? Two 39-column windows, that's what! And with the bottom fourth of the screen filled with menu prompts! I wanted to pull out my hair and punch someone, only my hair has already gone the way of the buffalo, and there was no one around but my neighbor's dog, who is bigger and meaner than me. Fortunately, by experimentation and painful reading of the "instructions," I found that you could disable the dual-window "feature" of Rebel Assembler; this gives you a 78-column window, but you're still stuck with the space-wasting menu information on the bottom. (I've since found the ideal solution: hit Restore, then change the now-black background to a different color so that Rebel Assembler's default black type shows up again. Now you'll have the full 80x25 display that you paid for.) In fairness, the dual-window feature was designed to let you see and work on two widely-separated portions of code at the same time, something we all wish to do occasionally. But on the whole I found the REBEL Editor frustrating to use.

There's little point in going on. Judging by their packaging, Nu-Age Software is not a large company, and a release of something as powerful as an assembler is bound to have a mite or two in it. And a program/text editor is a real personal thing, like a word processor--we all have our pet likes and dislikes. The Rebel Assembler/Editor has the potential to be a real killer, especially at only $24.95. All it needs is a manual, and a bit more testing to see what features users want and/or dislike.

The Accountant *by Avonelle Lovhaug*

We've all heard the terms "serious computers" and "serious applications software". Many people will tell you that these terms have nothing to do with the C-128, but don't believe them. As more software is released for the C-128, we are finding a tremendous amount of "serious software" and very little games, etc. An example of this kind of software is "The Accountant" by KFS Software Inc.

The Accountant is a complex program designed to handle the business accounts of a small corporation or firm. The package comes complete with a master program disk, a master data disk, a spiral bound manual, a quick reference guide, and a dongle key for copy protection. (This dongle is the first wooden one I've seen.) Since the program uses dongle protection, you can (and are encouraged) to make back up copies of your disks. The program is designed to handle the four major accounting tasks of most businesses: general ledger information, payroll, accounts receivable and accounts payable.

The general ledger section of the program contains all of the accounts that you will post to during the month. This program uses a monthly cycle of accounting, which means that you need to update the file at the end of each month, and probably will start a new disk at this point to keep records on a month to month basis. The Accountant uses a self-balancing set of accounts, which makes it difficult to throw the general ledger out of balance. This is very important, since errors which create imbalances can be very hard to correct. You are allowed up to 200 different accounts. For the most part, you can choose how you want your account numbers set up, although there is a general pattern which must be followed in order for the program to function effectively. The software allows you to print several types of reports. One is a detailed general ledger report, which is a complete printout of all of the general ledger accounts, with their beginning balances, the month's postings, and ending balance. Another report is a series of financial statements, such as statement of income and retained earnings. The program does not offer a check printing routine. Another part of the G/L section is the ability to post up to 30 journal entries to adjust up to 15 G/L accounts. This is set up as a traditional debit/credit display with columns.

Although it is a separate program than the General Ledger, a payroll program is also included with the Accountant package. The payroll section can handle up to 60 employees. The information the system holds on each employee includes: name and address, social security number, starting date, ending date, hourly rate, exempt status, payroll classification, insurance amount, any extra Federal withholding Tax, and any other regular deductions from the employee's check. After the employee information has been entered, payroll hours can then be input into the system.

The Accountant Review Continued

The system can accommodate regular, overtime, and vacation hours, as well as other deductions. The input payroll section of the program is also used to correct an incorrect payroll input. To correct a wrong input, you cannot just type in the correct number. Instead, you must compute how many hours you were off and add or subtract this amount with a new input. I found this method annoying, and would have preferred a direct method of correcting bad payroll inputs. The payroll section will allow you to print a payroll report which will include the check number of each check written. The system will also print the payroll checks for you, however, you must buy the checks that the program is formatted for, as there is no way to alter the check printing format. The system can handled state and federal tax withholding, and will compute the necessary amounts for you. Payroll related reports available include: a quarterly state unemployment compensation report, employee names and addresses report, employee pay rates report, W-2 printing, Y/T/D detailed report of all employee earnings and their deductions, and a current month's payroll journal report.

The Accountant also has an Accounts Receivable and Accounts Payable package. Both of these are completely independent, and are not a part of the General Ledger or each other. The accounts receivable section allows up to 100 customers with a maximum of 300 outstanding invoices. The system keeps the invoices on file until they are paid or credited. When in the Account receivable section, invoices must be input with descriptions, and then paid invoices can be eliminated. The accounts payable section works very similar to the accounts receivable section, with information about vendors instead of customers, and invoices input and paid. Reports that can be run within either include: an aged analysis (with current, 30-60 and over 60 day aging of all outstanding invoices), a detailed report of outstanding invoices, vendor or customer information, customer statements (for A/R only) and a voucher listing of invoices to be paid to a particular vendor (for A/P only).

The manual comes in a three ring binder, and includes a quick reference guide to the menus. Also, the manual and the reference guide provide check lists for various tasks that must be performed monthly to keep the records straight. Occasionally, the manual will run some words together, but for the most part it is readable. I found it to be sparse on information in some parts, and talky in others. For instance, I would have liked a more detailed discussion of some of the reports in the General Ledger section. However, in other places the manual rambles on and on about the hazards of 1541 disk drives, and recommends various times that the reader consider the 1571 instead. Also, the manual warns the reader occasionally that their disk drive may lock up. According to the producers of The Accountant, if it hasn't finished the task in "X" minutes, turn it off and try again. Although the program never locked up on me, it worries me that the writers of the manual expect it to.

I was very annoyed at the way the program handles editing. This was not only true of certain entries, as mentioned above, but also with simple typos that are recognized immediately after typing them. Instead of using the Inst/Del key, the programmers chose to use the back arrow key on the left of the keyboard. This drove me crazy, because every other piece of software I own uses the before mentioned key. Why can't software developers stick to the standard?

The Accountant is not for the browser. At $149.95, this software is too expensive for anyone but the serious small business person. I would advise you to check with a professional accountant before you decide if The Accountant is the right program for your business.

Sam's "Commodore 128 Reference Guide" *by Randy Margolis*

I recently noticed in my local computer store a newly published volume called 'Commodore 128 Reference Guide for Programmers' written by David L. Heiserman, and produced by Howard Sams & Co. As it is quite a hefty book I was not surprised by the $19.95 cover price and, being a pushover for material of this type, I parted with the twenty, went home, and eagerly dug into it. The obvious comparison for this book is going to be the 'official' version commissioned by Commodore to Bantam Books and that's the reason I think Sams miscalculated somewhat in the title, and partially in the execution, of this treatise. It's really more of a 'Tricks and Tips' type of work with a BASIC 7.0 encyclopedia tacked on; and as such it succeeds very well. I see this as a supplement to the Programmer's Reference Guide rather than an alternative to it.

The author won my vote by stating right off that this is a C-128 guide and therefore will not refer to the C-64 mode except when absolutely necessary. The result is 550 pages of pure C-128 information, which should be sufficient to excite most serious owners. He states (and rightly so) that if you need data about C-64 mode you should consult the C-64 Reference Guide. Bravo!

The first twenty or so pages describe how to get going on the machine for new users. This section basically duplicates the information in the System Guide that comes with the computer, but is much more clear and concise. This clarity and the intelligent way he organizes the subject matter is by far the book's major strength. All related

Sam's "Commodore 128 Reference Guide" Continued

items are grouped together, relieving the necessity of flipping back and forth trying to get the complete picture of a particular subject.

Chapter two is the Basic 7.0 dictionary. This is the part that should have been left out. It is so fraught with errors and omissions that I can't believe it was written by the same person as the remainder of the volume. While there are some excellent observations, (such as the warning about relative parameters of Locate not working on early versions of the C-128) there are also many misleading and downright erroneous statements. An example: under the SLOW statement it is stated "The system must be operating at the SLOW rate in order to display any information on the screen". Unforgivable! Buyers of this book would do well to avoid referring to this section.

The remaining chapters (with the exception of a pitiful ten page 'intro' to CP/M) are outstanding; and, since the really poor material constitutes only about a fifth of the book, I think that it is a good value at the price. Chapter two is a full 50 page discussion of DOS-related procedures; everything in one place, neat and tidy. Included are tables of DOS error messages, the disk organization scheme, and short programming examples which effectively explain each topic. Machine language aficionados are really going to love this book. Mr. Heiserman breaks new ground by placing equivalent ML subroutines (fully commented, but easily entered with the resident ML monitor) with the related BASIC example! Now I'm definitely not a machine language expert, and the side-by-side illustrations advanced my knowledge considerably. The only thing wrong in this chapter is that there should have been a clearer distinction between 'drives' and 'units'.

Next comes a 40 page tutorial on using the built-in ML monitor. It does not pretend to be an 8502 machine language lesson, it just tells you everything you need to know about using this handy utility. Included are two charts detailing the 8502 instruction set, similar to the one in the Bantam Guide.

Section 5 is the 'Introduction to CP/M'. Tear it out; it's useless.

The next three chapters deal with, respectively, the text screen, bit-mapped graphics, and sprites. Now, old Sparrow James will confirm that graphics and myself are total strangers, but when I finally get that brain transplant and am able to draw something more esthetic that a 'smiling face', these sections will come in handy. They are also liberally sprinkled with detailed machine language routines adjacent to their equivalent Basic counterparts.

The author must truly be interested in audio because the thirty pages he devotes to sound and music techniques is the most impressive part of the book. The first subsection, entitled 'Preliminary Considerations' is a fairly detailed discussion of the physics of sound! After this is everything you want to know about the Basic 7.0 sound statements, and how to drive the SID chip with machine language. This is one of the best treatments of the subject I've seen.

The next part, covering the keyboard, is a wonderfully technical but easy to understand discussion of exactly how this section of your computer operates. You'll never have trouble defining the HELP key or 'Shift/Run/Stop' combination again. There are plenty of charts to assist in your understanding of how this amazing machine communicates with the outside world. Consistent with the rest of the book, you'll find the assembly language routines in their proper place, and all the tables are in both decimal and hex. Mr. Heiserman puts his charts and tables where I feel they belong: with their associated text. Want to learn what the ALT key does? This is the place. There follows a fairly good treatment of peripherals (joystick, mouse, modem, printer, etc.) with emphasis on modem communications. This section could have been more complete: the Ram expansion is almost totally ignored.

Chapter 13 is the finest C-128 memory map I have seen. It's definitely the first one I've come across that is in order by location. Can't see why everybody doesn't arrange this essential programmer's tool in such a straightforward manner. In addition, there is a short description of every important Rom routine (with permissible values where appropriate), 11 pages of Basic jump vectors, and a detailed Kernal section complete with the actual entry point of the associated routine and preliminary steps necessary to successfully implement the procedure.

The book concludes with an ambitious discussion of that most confusing of C-128 topics, memory management, bank switching, and configuration registers. I'm still plugging away at this subject, but several previously sketchy (to me) concepts were clarified in these pages.

If I sound enthusiastic about this Guide, it is because the author employs new approaches to difficult and sometimes confusing aspects of Commodore 128 programming that will make many subjects a lot more understandable to those of us that are not computer geniuses. I only wish that the publisher had named the work more appropriately and left out the first two chapters. I hope that this review will cause more folks to seriously consider owning the book, who might otherwise dismiss it as another ripoff of the official Guide.

Trinity *by Bill Nicholson*

All microcomputers can claim a broad variety of software titles; productivity, education, telecommunications, graphics processing, music development, and entertainment (usually games). The C-128 has out-performed most other computers with the quality and variety of applications packages. But for those of us who use our computers as full-time companions, we have had to repeatedly downgrade to C-64 mode to entertain ourselves, tossing aside more than 1/2 of the qualities we bought the C-128 for.

There are a couple of games that have been available almost since the inception of the machine, but these have been written by users, without the tricks and resources of the larger software development houses. In fact, until April of this year, no one could put their hands on any commercial entertainment software that ran in C-128 mode.

A few of these have been brought to our attention, but the first we have had an opportunity to use is "Trinity," written by Brian Moriarty for Infocom. Like all games that have come from Infocom, it is an interactive text adventure game. If you have no experience with this type of game, a brief explanation is necessary.

The player is given a set of circumstances to imagine him or herself in, and a desired outcome (not always given, but it is nice to have). Using one's wit and wiles, the object is to get from Point A to Point B. By no means is this a straight line, but often is a tortured and circuitous route that pushes back the frontier of your thought processes.

For those of you who have played this type of game before, Trinity is an example of Infocom's attempt to make them harder and more intriguing. The name they give this is Interactive Fiction Plus, which means that it requires a minimum of 128K of RAM to hold all the data in the program. You will not see Trinity in versions for the Apple IIe, the Tandy computers, or the C-64, so finally a game for OUR kind of computer.

So, having said all that, what is this game? It is a game that places you in London literally moments before World War III starts, with but one hope of survival. Clues are dropped all over the place, guiding you to the salvation. The basis of all action in the game is from reaction to the clues around you. Be prepared to restart the game several times after being atomized in the first attack of WWIII.

The bottom line is that this game is challenging, and entertaining, but at the same time irritating and confusing. There are times when the course to salvation and the rest of the game is blocked by mean-spirited natives, cliffs, monsters of many descriptions, and protocol. This constantly leads to frustration, but by patience and guile, it is fairly probable that success will be yours.

The game runs well and quickly, in spite of frequent disk drive accesses, and uses the facilities of the 80-column screen as well as an all-text game can. It does seem to use the faster disk reading abilities of the 1571, although the program comes in 1541 mode. This requires flipping the program disk during boot-up, slowing down the 1571's speed. I can see 2 reasons Infocom produced the game in this format: Everyone can read 1541 disks, and it prevents the possibility of disk-reading problems if the announced 1571 ROM revision is released. I have yet to try to copy the program data to a 1571-formatted disk, but I suspect this will not work, since the disk program is looking for a 1541 disk.

Is this game worthwhile? If you have never played any "interactive fiction" games, ask yourself some questions before laying out the $25 or so that this games costs: do you spend a lot of time playing arcade-style spaceraider types of games, or do you prefer games that take a lot of setup and strategy to conquer? If you prefer the former, this is probably not the game for you, since it can be very slow-moving and tedious at times. If the latter is more your speed, this is worth a look at. Personally, an overall rating of 7 out of ten, and a big one-oh for being a true C-128 game.

Making Waves *by Bill Nicholson*

What are musical filters? Filters, in electronic music, as they do everywhere else, allow certain things to pass through, while removing certain things you want kept out. In C-128 SID music, Filters allow certain frequencies to pass through and remove frequencies above or below a specified range.

Why would you want to limit the tone of a sound to a certain set of frequencies? To create a purer tone, when trying to emulate an acoustic instrument, and to create new sounds. For example, if you are trying to create a cymbal, you might start by using the drum preset (number 3). But, upon listening to it, you find it sounds more like a snare drum, rather than the "higher" pitched cymbal. To create this "new" sound, you can do one of two things: try new, higher frequencies through the use of the ENVELOPE command (see TC-128, September 1986, "Making Waves") and fiddle with the wave shape. Or you could try blocking the lower frequencies to create the cymbal's piercing sound.

Making Waves Continued

There are 3 types of filters available to the SID chip: a high-pass filter, a low-pass filter, and a band-pass filter. These 3 filters do exactly what they say they do. The low-pass filter allows the low frequencies to pass, while filtering out the higher ones. The high-pass filter allows the higher frequencies through, and the band-pass allows a band of frequencies to pass into our hearing. Each of these filters use the same format in setting where the cut-offs happen:

FILTER [freq],[,low-pass] [,band-pass] [,high-pass] [,resonance]

The frequency setting is the point where the cut-off occurs. Although this is a precise number, there are frequencies on either side of this number that will continue to bleed through. This value is from 0 to 2047. The following three parameters are binary in nature; that is, 0 turns that filter off, and a 1 turns it on. You can have more than one filter turned on at a time, creating different effects. Resonance determines how forceful the filter is. This is set with a value of 0 to 15.

While only one filter envelope can be invoked at a time, by activating more than one filter at a time, you can create a more intriguing sound. By using both the high and low-pass filter, you can create a frequency response with a 'hole' in the middle (the area surrounding the specified frequency). This is known as a "notch-reject filter." This creates, in essence, the opposite of the band-pass filter, which allows only the frequencies near to the assigned level to be passed.

By using the high or low-pass filters with the band-pass filters, you can further define the audible frequencies. Using the high-pass and band-pass filters together would create a more distinct low frequency cutoff than is possible using the high-pass filter alone.

The main thing to remember when creating FILTER envelopes is that almost anything is correctable.

Expanding Your VDC Ram *by John Kress*

The very first time I had fired up my new C-128 I loved the 80 column screen. I remembered the days of my Vic-20 with 23 characters per line. Now I had a computer with an 80 column screen and I thought to myself 'Now I own a real computer'. Then I came across a very basic 80 column graphic program that used a joystick, and I thought that this is great, but it is too bad that you can only use one color on the screen at a time, and secretly dreamed of owning an Amiga. Then came the day that I found out that the memory dedicated to the Video Display Chip could be expanded from the stock 16K to 64K of Ram. Right then and there I was hooked (I believe it's referred to as 'Technolust' in these parts) and wanted to learn all I could about expanding the Ram to its maximum.

I should explain a bit about the VDC and its Ram. The 80 column screen uses a separate chip for screen processing, and that chip has its own dedicated RAM. Unlike the VIC chip, for the 40 column screen, which uses part of the Basic Ram for screen memory, the VDC has 16K of RAM tied into it that cannot be accessed in the normal peek and poke method. The VDC is in some ways a separate processor that operates independently from the rest of the C-128, and is able to work at a faster clock speed than the VIC chip. That is the reason why the video display goes blank on the 40 column screen when the FAST command is issued.

Why expand the VDC memory?

Since my major computer interest is graphics, I wanted to find out how things might be improved with more memory. One of the first things I pondered was the graphics screen with attribute Ram for color. That would allow more than one color on the screen; although you're still limited to just one color in any 8*8 pixel area, you can have more than one color. Then there is the ability to have a combination of screens in VDC memory, up to 4 graphic screens, up to 13 text screens, or a combination of both. There is also the possibility of making up different fonts (print styles) in memory and being able to change them.

(Editor's note: The execution of this process will void your computer's warranty. Readers who attempt this process do so at their own risk. Neither this author or Twin Cities 128 will take responsibility for losses due to the attempted execution of the process outlined in this article.)

The parts you will need are :

* Two (2) 18 pin IC sockets, to be used to hold the Ram chips. These can be found at most any electronic supply house.

Expanding Your VDC Ram Continued

• Two (2) 4464 Dynamic Ram Chips, with a speed of 120 ns. Yes that's correct, 4464 Ram chips. I know that the Commodore Programmers Reference Guide says 4164 Ram chips, but 4164 chips are a sixteen pin type. The 4416 chips are an 18 pin type so if you try to use the 4164's you'll have two extra holes and nothing will work right.

The 4464's can be a hard item to locate, as most of the manufacturers are either discontinuing them or require minimum orders and are reluctant to release 'just' two. If you have some friends or a local computer group or club, get together and purchase a larger quantity, you may have more luck with a volume purchase.

You will need the following tools:

Wire cutter or nippers.

A small pencil type soldering iron (35 watts is be enough, much more may get too hot.)

Some rosin core solder or better yet,
Solid core solder and a paste type non-corrosive flux.

Small phillips screwdriver.

Small pair of pliers.

Now you're ready to begin:

Notice: Integrated Circuits are extremely sensitive devices, and very sensitive to static charges. A small static shock to a IC is as deadly to them as a bolt of lightning would be to a human. Take extreme caution to avoid a static discharge to the computer.

First thing to do is remove all power cords and serial connectors from the C-128. This means everything, since the main board must be removed from the machine.

Turn the computer over and locate and remove the six (6) screws holding the computer case halves together, then carefully separate the halves. This is critical, because the keyboard and power light connector along with a ground braid for the keyboard are still connected and could be damaged. Carefully remove the screw holding the ground braid, along with the connectors for the power light and keyboard, taking note of their proper connection. I found that the keyboard connector was installed with a brown wire closer to the power switch, but things may be different on other machines.

The next step is to remove the screws holding down the computer board and the ground casing. There were six screws around the perimeter of the main board/shield assembly, remove all of these. Then you must use the pliers to gently bend the tabs holding the upper ground casing to the lower casing. There is one more screw mounted on the ground casing in about the top middle, this must be removed also.

There is one spot where the casing halves were soldered together on my computer and that was in the front right side, near where the numeric keypad sits. Carefully unsolder the halves and lift the top half away. What you now see is the main board with a small metal box on it, near the center of the board. The chips that need to be replaced are inside that metal box.

Inside the box are both of the screen controllers, the 8564 Vic chip and the 8563 VDC chip. Remove the lid to the box and locate the two chips that are identified on the circuit board with the marking U23 and U25. On my board these chips were labeled with a large F and the part number MB81416-12. The number may vary due to the manufacturer, but the markings on the board will identify the correct chips.

Now for some very delicate surgery. Since the Ram chips will be discarded, I suggest that you carefully cut away the chip pins (leads) from the body of the chip. This will make removing the remaining portion a lot easier, and leaves less of a chance of damaging the main board than trying to remove the whole chip, intact. The next step is to carefully unsolder the pins from the circuit board. Make sure that you are working with the correct area, as things do get rather confusing when you turn the board over and notice all of the many soldered areas. You might be best off to mark the correct area with a felt tip pen to avoid confusion

Expanding Your VDC Ram Continued

The next step will be to install the IC sockets into the circuit board. Try fitting the sockets into the board to make sure that they will fit all of the way through the holes and that there is enough material poking through for a good solder joint. Also be sure to orient the sockets with the small identification notch on the correct end of the board. The notch is printed on the board so you should have no trouble.

When you have the sockets installed correctly, lightly bend the pins over to hold them in place and you're ready to solder them in place. Again be careful, as drips of solder can cause a short, and a cold solder joint will cause a poor connection.

Now you're ready to install the 64K chips in the sockets, carefully inspecting the pins for proper alignment. Most of the time chips will be made so that you will have to bend the pins inward, toward the center, to fit properly in the socket. But too much bending will break off a pin.

Once you've installed the chips you're ready to start the reassembly process, just reverse the disassembly steps and make sure to solder the ground case halves again. Reconnect your cables and power up the computer.

If all went well, you should see nothing unusual. The same old cold start up message and nothing different. Some of you might notice a different pattern on the screen when the initialization process is going on but after that you should see the same old screen. The cold start routines for the C-128 initialize the VDC registers, and the VDC chip is told that it still has the 16K chips in it. In order to tell the VDC you've upgraded the Ram, you will have to set bit 4 of register 28. When this is done, for some reason, the character information gets scrambled, but a call to the copy rom routine at $CE0C clears the problem.

If your screen does not appear the way is should, look for a cold solder joint or a pin that is not installed into the socket correctly. I found that when I tried to remove the old chips in one piece, I had damaged a portion of the copper foil on the circuit board and had to trace the circuits out and install a jumper wire from the 8563 socket to one of the Ram sockets.

I hope to be passing more information to you regarding the VDC chip and some programming tips in the coming months, so keep looking.

Forgotten BASIC *by Miklos Garamszeghy*

Basic 7.0 on the C-128 offers a number of improvements over Basic 2.0 used on the C-64 and VIC-20. One of these is error trapping. How many times have you run a program, only to have it crash because you entered the wrong thing at a prompt? If you are like me, this has probably happened to you more than once. With Basic 7.0's error detecting and correcting routines, you can recover from most errors without crashing.

There are two error detection methods available on the C-128, one for Basic errors and the other for disk errors. To enable Basic error detection, your program needs a statement such as:

```
10 TRAP xx
```

where xx is the first line of your error handling routine. The TRAP statement should be near the beginning of the program or major subroutine that you want to protect from error crashing. More than one TRAP statement can be included in a program, but only the most recently encountered one will be active. Thus you can have different error handling routines for different sections of your program. Error trapping is disabled with the statement:

```
100 TRAP
```

with no line number specified.

When an error is detected, the program flow is diverted to the error handling routine mentioned in the TRAP statement, just as if a GOTO had been encountered. Once in the error routine, you can determine the nature and location of the error using the reserved variables ER and EL. ER contains a numerical code corresponding to the type of error that was encountered. The error codes are listed in the C-128 System Guide and other similar reference manuals. Some of the errors, such as a syntax error, cannot be corrected unless you edit the program. Others, such as Division By Zero or File Not Found can usually be corrected by specifying a new value for a given variable. A text description of the error can be read with the function ERR$(ER). For example, to detect a divide by zero error, you would use an expression such as:

```
10 IF ER=20 THEN PRINT ERR$(20).... etc.
```

where .... etc represents your additional code. ER and ERR$() can also be used in direct mode with a PRINT ER or PRINT ERR$(ER) statement. The Help command can also be used in direct mode to list the line where the error occurred and highlight the error statement in a multistatement line. If Help is used in program mode, execution

Forgotten BASIC Continued

will stop and the error line will be listed with the error highlighted. The Run/Stop key can also be detected with TRAP and ER. It has been assigned an error code of 30.

The reserved variable EL contains the line number where the error occurred. If the error occurred in direct mode, EL will have a value of 65535. To detect any error in a given line, say line 100, of your program, the error routine would contain a line such as:

```
20 IF EL = 100 THEN PRINT "ERROR IN LINE 100".... etc.
```

Two caveats should be mentioned about using EL and ER in programs. The line number referenced in an EL statement will not be renumbered when you use the Renumber command, so check all the EL's after renumbering a program. The second drawback is that you cannot TRAP an error within the error routine itself. TRAP is disabled until control is returned to the main program. This brings up the next item of discussion: error recovery and returning control to the main program.

An error detection routine would not be much use if you could not correct the error or at least return to the normal execution of the program. The error correction depends on the specific application and may involve setting default values for parameters or printing an error message and asking for a new input etc. This is beyond the scope of this article. Returning to the main Basic program is done with one of the following statements: RESUME, RESUME NEXT, or RESUME xx, where xx is the line number to continue with. With the first form, Basic will try to re-execute the statement where the error occurred. In the second form, Basic will execute the statement immediately following the one where the error was encountered. In the third form, the program will resume execution at the specified line number. Until one of the RESUME statements is encountered, Basic 7.0 disables further error trapping. Thus if a second error occurs during the error handling routine, the program may crash. RESUME will not purge the stack of the most recently unused GOSUB, FOR-NEXT, DO...LOOP etc. parameters before returning control to the specified line, so be careful of nested subroutines and loops. An error trapping routine may have more than one RESUME statement:

```
100 IF EL=100 THEN RESUME 10
120 IF ER=4 THEN RESUME NEXT
```

This helps in directing the flow of the program back to the specific area required after detecting and/or correcting the error. It also minimizes the chance of crashing during error trapping by allowing you to perform error correction outside of the main error detection routine. After the execution of the RESUME statement, ER is set to a value of -1 and EL is reset to 65535, both indicating a no error condition.

Disk errors can be conveniently detected using the reserved variables DS and DS$. DS contains the error code of the most recent disk access and DS$ contains the text description of the error. DS and DS$ depend only on the status of the last disk access, not the drive number, device number or file number where the error was caught. A list of DOS error codes can be found in the appropriate disk drive user's manual. DS and DS$ remain at their error values until the next disk access is attempted. A simple statement such as:

```
10 IF DS THEN PRINT "DISK ERROR:";DS$
```

will detect a disk error. DS and DS$ can be used either in program mode or direct mode, you do not need to open the disk command channel (#15) first. With the C-64 and VIC-20, it was not possible to read the disk error channel in direct mode without a DOS wedge. The Basic variable EL for error line detection is not used in conjunction with DS and DS$.

As a matter of good programming practice, you should read the disk error channel whenever you perform normal disk operations, such as OPEN, INPUT#, GET#, PRINT#, BLOAD, etc. The only exception is after issuing a burst mode command to the 1571 drive with a PRINT# statement. This may cause your program to crash because reading the error channel cancels the burst mode commands by temporarily commandeering the command channel and issuing a different direct access command to the drive.

In reviewing the list of error codes for both Basic and DOS, you may have noticed that some of them are the same. What happens in such cases? The first level of error detection is always with Basic. Thus if you do not have a TRAP statement, and the Basic error is encountered, the program will likely crash. For example, a "file not found" error (Basic error #4 and DOS error #62) should be handled by a TRAP statement. However, to turn off the blinking error light on the disk drive, you must also read the disk error channel with a statement such as:

```
50 ZZ=DS
```

Most versions of CP/M Basic, such as MicroSoft MBasic, used on the C-128 in CP/M mode also include error trapping functions. The MBasic command to activate trapping is:

```
10 ON ERROR GOTO xx
```

The forms of the RESUME statement are the same as for Basic 7.0. Error code numbers are assigned to the reserved variable ERR and line numbers are assigned to ERL. Unlike Commodore Basic, error line numbers in ERL are

Forgotten BASIC Continued

correctly renumbered with the RENUM command. The ERROR xx statement can be used in either program or direct mode to simulate an error condition. This allows you to define your own MBasic error codes. For example, error codes 31 to 49 and 68 to 255 are not normally used. A statement such as:

```
20 ERROR 200
```

can be used to control program flow by detecting it in an error handling routine with:

```
100 IF ERR=200 THEN.....
```

An interesting feature of MBasic error detection is that it remains active, except for within the error detection routine itself, until an:

```
ON ERROR GOTO 0
```

statement is encountered. Thus if you make any errors in direct mode after running a program with a still active error trapping routine, you will magically taken back into the program and its error handling routine.

Sparrow's Slick Tips *by Sparrow James*

Sparrow's Slick Tips: The art of taking your C-128 farther, faster. We are always looking for new programming tricks and application program tips which C-128 users could find useful. If you have a special trick or technique, why not share it with the rest of the known universe by sending it to Twin Cities 128, P.O. Box 4625, Saint Paul, MN 55104.

00101110: Looping Around the Stack: The DO..UNTIL/WHILE..LOOP structure of Basic 7.0, besides lending itself nicely to structured programming, as a rather convenient "hidden" attribute it can also be used to rescue a complicated program from the evils of poor structure or over-nesting loops. When a DO..LOOP has been satisfied, unlike other Basic 7.0 looping structures, it deincrements the stack not for only itself, but it scraps all stack references to any loops that might be nested within that DO..LOOP. In a somewhat similar fashion, the GOSUB..RETURN structure upon RETURN will also scrap all stack references to any DO..LOOP or FOR..NEXT loop that is nested inside of the last called subroutine. Armed with this knowledge, the experienced Basic programmer should be able to make Basic 7.0 turn some special structural tricks, but beware that improper exploitation of this technique code leads to the creation of indecipherable "spaghetti code" or frequent out of memory errors. For a more complete and technical discussion of the above, see New Loops: The Commodore 128 Basic stack, in the September 1986 issue of the Transactor, Volume 7, Issue 2, by Jim Butterfield.

00101111: GEOTRICKS: Though most GEOS users are oblivious to them, the following shortcuts are available from the desktop and applications programs:

1. Scrolling through directory pages on the desktop can be greatly accelerated by simply typing the number of the page you desire to see, instead of clicking the "dog-earred page" icon.

2 In GEOPAINT, one can enter pixel edit mode by simply clicking the pencil icon twice, instead of going through the options menu at the top of the screen.

3. By clicking the eraser icon twice in GEOPAINT you can erase the entire contents of the current display window in both normal edit as well as pixel edit modes.

4. Clicking the box tool icon twice in GEOPAINT will allow selection of the entire current display window for manipulation.

Note: "clicking twice" sometimes called "double clicking" requires some practice to get consistent results, also note that the effectiveness of double clicking might be hampered somewhat by the particular input device you are using.

Rumor/Opinion/Mayhem *by Loren Lovhaug*

Perhaps I am losing my patience, or perhaps I am just insensitive, but I have come to the conclusion that there are some people who should not or ought not own computers. This is a pretty daring (or maybe just plain stupid) statement to be written by somebody who is attempting to earn a living as a computer journalist. After all, it can be argued that as a C-128 information provider I have only to benefit as the number of C-128s sold rises. But as contrary to logic as it may seem, I truly believe this.

Let me explain: When I am not trying to earn a scant living composing and editing the publication you are reading now or spending my time with my wife and son, I occupy my time teaching computer classes, frequenting local bulletin boards and national telecommunications networks, and helping people use their C-128s at home, in school, or in business. In these endeavors I come into contact with a variety of people of varying ages, abilities and backgrounds. Many of these people are beginning computer users and owners who are attempting to learn more about their machines in an effort to utilize their tools to enhance the quality of their daily lives.

I am happy to report that the vast majority of these people I come into contact with become moderately successful in their efforts, regardless of whether their goals involve using their computers for work, entertainment, education or self-fulfillment. I wish I could take credit for their successes, but alas I cannot, because as an instructor, helper, mentor or whatever, I have very little to do with the success (or failure) of an individual with their computer. This stems from the fact that the keys to achieving a successful computer experience are internal, not external.

Perhaps the old adage, "You can lead a horse to water, but you can't make him drink" applies here. No matter how much we at Twin Cities 128 try, we cannot MAKE your C-128 work for you, you will have to do that yourself. With this in mind, here are the characteristics that I feel separate the successful C-128 owner from those that are destined for frustration:

1. Invest the time: Perhaps the primary ingredient for success with your C-128 is your willingness to take the necessary time to learn about it. This often involves two major activities: A) Reading the instruction manuals and documentation, and B)Practicing or experimenting with your C-128. These steps may appear obvious and relatively easy, however the majority of user frustration stems from failure at these activities. It is truly astonishing how many people refuse to read instructions and documentation, or refuse to sit down and practice and experiment with their machines. I have heard all the excuses like, "The manual was poorly written" or "I have no time to play, I just want to get up and running". But I have yet to see a C-128 hardware or software manual that was so poorly written that it was totally unintelligible, and I have yet to meet a C-128 user who truly put in the time and the effort, experimenting and exploring their machine, and failed to succeed. People who do not have the time or the discipline to study and explore simply ought to avoid computers entirely. Lazy people simply do not succeed with computers, C-128 or otherwise. Perhaps the best advice one can follow when attempting to learn how to utilize their C-128 is: Budget a generous amount of time for learning and proceed slowly, paying attention to detail. I realize that reading and practicing of these activities are time-consuming, and sometimes tedious, but they are simply necessary and in the long run they actually pay off in both time savings and satisfaction.

2. Attitude and desire to learn: If you approach the task of learning to use your C-128 in a positive enthusiastic manner, putting in the necessary time and effort will be easy and fun. Perhaps the key to maintaining a good attitude while learning how to utilize your C-128 is realizing that failures and setbacks are part of the learning process. Allowing setbacks to taint your outlook on yourself or your C-128 will simply lead to frustration and ultimate failure.

I personally know of one former C-128 owner who allowed his frustration to get the better of him and ended up buying another computer system that supposedly was easier to use. But much to his disappointment he began having problems shortly after he began working with his new system, and his inability to deal with his setbacks is literally making him miserable. This person's problems are not stemming from his computer equipment, instead they are caused directly by his own impatience and inability to cope with minor difficulties. People who can't deal with minor problems also will not succeed with computers because microcomputer technology is still a long way from perfection and problems will always manifest themselves. (This is not to say that microcomputers like the C-128 cannot be relied upon for accurate and faithful service, instead it is just a friendly reminder that expecting perfection from anything is too much to expect!) People who have a zest for learning and approach every new problem as a challenge to be overcome are much more likely to experience success with their machines.

3. Seek information: Successful C-128 owners are masters at seeking out information that is vital to their quest. When problems do occur, the first place to look for answers should always be your manual. Often you will find the complete answer to your problem contained somewhere within your hardware or software documentation. I cannot tell you

Rumor/Opinion/Mayhem Continued

how annoyed I get when somebody calls me on the phone and asks me a question about a specific piece of software, and the "miraculous" solution I provide for them is a verbatim recitation of that software's manual. In these cases, I am not surprised when they admit they did not bother to consult the manual, but I am surprised that their manners are so poor that they don't consider this obvious and in my mind obligatory step.

But there are times when problems may occur which go beyond the scope of the manual. This where local users groups and bulletin boards or telecommunications networks can become invaluable. Those seeking to become proficient with their C-128 should seek out local users groups and become involved in telecommunications because these are the fastest and most inexpensive routes for gathering new information and meeting individuals who can guide you through your difficulties.

Another source for information is printed material. The C-128 boasts an impressive library of support texts which cover all aspects of operation and levels of understanding. Also seek out the various Commodore magazines. Although I know of only one C-128 specific publication, others I recommend highly are The Transactor, Commodore Microcomputers, and Compute's Gazette.

I hope these paragraphs have not depressed you. The truth is that the vast majority of people do very well with their C-128s and don't seem to experience any difficulties with the concepts I have mentioned above. As for our role in this whole process of getting the most out of your C-128, I only hope that each issue of Twin Cities 128 contains at the very least one idea or hint that makes your experience with the C-128 a little easier, or a little bit more enjoyable.

NEWS AND RUMORS

- Yes, (as we said last month) the long awaited C-128 and 1571 revision replacement ROMs should be available around or just after Thanksgiving. We now have some details on these long awaited silicon saviours. The ROM replacement kits will be available both at local authorized service centers or through direct order from Commodore. There actually will be two kits: a combination kit which will entail the new C-128 ROMs and 1571 upgrades in one package for around $25 and a single 1571 upgrade kit which will contain only the 1571 ROMs for those who own more than one 1571 for around $15. The C-128 ROMs correct a few bugs in the C-128s kernal such as round-off errors in the internal math package, the now infamous CAPS-Q bug and the problem with negative value relative coordinates on the BASIC 7.0 bit mapped graphics commands. The 1571 ROMs correct some internal buffering design conflicts between pre-release prototypes of the 1571 that some software companies designed their software around and various end-user releases of the 1571. The 1571 ROMs also correct DOS bugs having to do with the use of relative and sequential files on the back side of a double-sided diskette and the 1571's apparent confusion when switching to single-sided mode (1541) or reading so called "flippies". Both sets of ROMs are socketed and can be end-user installed although user installation will invalidate your warranty (what warranty?). Those individuals who have bought C-128s or 1571s within the last 90 days will be given free upgrades through their local service center.

- Applications for the 1700/1750 RAM expansions are now exploding onto the horizon. If you have been reading Twin Cities 128 over the past few months you know that we have been telling you about various packages which are to be released to utilize the RAM expansion. I am not in any way exaggerating when I say that these packages will revolutionize the C-128 and bolster its competitive standing against the rip-off Korean/"COW-CHIP" clones (no bias here eh?). Here are the ones that have been officially announced, with more on the way:

Digital Solutions has announced upgrades of its Pocket Series software with many enhancements to an already superb line including RAM disk capabilities for document editing, module loading and spell-checking.

Berkeley Softworks' long-awaited C-128 version of GEOS will be released soon RAM disk ready.

An upcoming revision to Paperclip II (personally I think Batteries Included missed the boat by bringing it to market non-RAM disk ready) will allow use of the RAM disk for spell-checking and long documents.

And most the most wonderful development of all: Commodore is planning on releasing RAM disk software which will automatically configure the 1700/1750 as "pseudo disk drives". This software will be packaged with all upcoming 1700/1750 and 1764 RAM expansion units (the 1764 is the 256k RAM expander designed for the C-64c) and should be available in the public domain for existing owners of these units, opening up a variety of possibilities for programmers and existing applications to make use of the 1 megabyte/sec data transfers.

Software Recommendations *by Avonelle and Loren Lovhaug*

Choosing the right commercial software is often an extremely difficult proposition. Even an experienced Commodore 128 owner, armed with review information from various sources can, at times, experience difficulties. With good software, a computer can sing for its user; with bad or inappropriate software, the computer can be rendered useless.

Because at Twin Cities 128 we have access to a huge variety of C-128 commercial software, and use it in our daily lives and in preparation of this publication, we thought it might be helpful if we shared some of our insights in a software recommendation article. But we decided to take a unique approach. Instead of simply listing the titles we like best, we decided to try to match specific recommendations to the differing needs of various individuals. We have decided that the most effective vehicle for such recommendations was to present various hypothetical, but typical individuals and their needs along with a few recommendations that in our minds would be wise choices to consider. Remember, these are simply opinions, and only you can best determine accurately what software will best satisfy your needs.

Mary: Mary is an undergraduate student, majoring in communications. She is fairly new to computers, although she did learn some Basic programming in high school. She needs a full featured word processing program that will allow her to satisfy the specific requirements of her instructors. But she also needs one that she can learn to use quickly, since she is constantly under deadline pressure and has very little spare time. Mary also wants wants a simple database program to keep track of information about members of her drama club and her friends who are at other colleges. However, Mary is on an extremely tight budget and can only afford to spend no more than $40 per program.

Recommendations: Mary ought to consider the powerful and versatile word processor, Pocket Writer 128. Pocket Writer 128 is a very easy to learn "see what you get" word processing program that is available now for well under $40. With Pocket Writer, Mary should be able to produce documents that fit her instructor's requirements nicely (it has provisions for super and subscripts, double and triple spacing, automatic headers and footers). As for database programs, Mary we suggest Datamaster 128 or Pocket Filer 128. Both of these programs should allow her to do the kind of rudimentary database operations she requires while staying well within her $40 spending limit.

Randall: Randall works in the development office of a large non-profit corporation. This corporation uses a number of IBM PCs running Lotus 1-2-3, and he is looking for solid spreadsheet software that will allow him to on occasion work at home and allow him transfer data from his C-128 at home to Lotus 1-2-3 at work.

Recommendations: Our first suggestion for Randall is to consider the acquisition of a modem and a powerful telecommunications packages such as Bobsterm Pro 128 or Sixth Sense 128 which will greatly enhance his ability to communicate between his IBM PC at work and his C-128 at home. As far as spreadsheet programs go, he ought to consider either Vizastar 128 or Multiplan 128. Both of these programs are extremely powerful and boast extensive functions for financial analysis. Both programs should be able to create files that are exportable into Lotus, provided you transfer them to IBM formatted media, either via modem or a disk transfer utility like the Big Blue Reader (commercial), or Crosslink (public domain). Vizastar 128 uses a command structure that closely parallels Lotus, so in some ways may be easier for Randall to learn. Multiplan is significantly cheaper.

Ken: A writer of technical manuals for various equipment, Ken needs a full-featured word processor that includes many "extras" specific to his profession. Some such features include indexing, automatic table of contents creation, advanced column manipulation, telecommunications abilities, an integrated spell checker, and a thesaurus.

Recommendations: Probably the word processor that best fits Ken's requirements is Paperclip II (see review in this issue), since it includes everything Ken is searching for except the thesaurus. Superscript 128 also would be a good choice and includes many of these extra niceties. Another choice is Pocket Writer II, which will be available very soon and appears to include the kind of powerful features Ken wants. Fleet System 3 includes a sophisticated spell-checker and thesaurus. One other consideration for Ken is the ability to transfer text to a laser printer for the production of near typeset quality text. The following software has the ability to transfer text to a laser printer: Paperclip II, Vizawrite Classic 128, geoWrite 2.0 and geoLaser (available on the Writer's Workshop package under GEOS) and probably a few others.

Harold: As an attorney, Harold needs professional word processing and a sophisticated programmable database for his billing needs and scheduling needs. In addition, Harold needs a spreadsheet for financial planning and reports. He is also looking to integrate word processing, spreadsheet and database files.

Recommendations: For a database, Harold should choose Superbase or Pocket Filer. Superbase is one of the most flexible and versatile database programs available for any microcomputer, and will allow him to set up a complicated automated system that will meet all of his needs. For word processing Superscript 128 might be a good choice since it

Software Recommendations Continued

can be operated in a co-resident environment with Superbase 128. He should also look at the Pocket Series (Pocket Writer, Pocket Planner, and Pocket Filer) and the Timeworks series (Wordwriter, Swiftcalc, and Data Manager) since these programs feature a high degree of integration and can easily manipulate data created by programs in their series as well as others.

Sandy: Sandy is a secretary who puts together the company newsletter with her C-128. She wants to be able to use multiple fonts, and graphics. Her company has recently purchased an Apple Laserwriter for use in the company's art department which she has been given clearance to use.

Recommendations: There is no doubt that Sandy ought to consider GEOS 128. Applications running under this operating system will allow the use several laserwriter fonts, and allow her to easily include graphics in her articles without having to physically cut and paste them into her text. There are also programs available under the GEOS system which will allow her to utilize pre-packaged clip art designed for use with the Print Shop, PrintMaster, and the Newsroom. Also available are text conversion utilities which will allow Sandy to convert text created with a variety of different word processors into geoWrite format.

Ted: Ted is a sales manager that loves to program computers. He is an adept Basic programmer and is a fairly good assembly language programmer. He is looking for a good Basic 7.0 compiler and a good C-128 mode assembler. Ted also is interested in learning about other computer languages such as Pascal, C, and COMAL.

Recommendations: Ted should consider the following Basic compilers: Basic 128, Petspeed 128, and Blitz 128. All three of these packages do a fairly good job of compiling Basic 7.0 programs for faster execution. Basic 128 has a powerful set of compiler directives and includes and option for compiling directly into machine language. Both Petspeed and Blitz are easier to use than Basic 128 and are more "automatic". As for assemblers there are several good ones to consider: Merlin 128, TSDS, Freedom Assembler, and The Rebel Assembler are a few. Also Commodore will be very soon releasing its own assembler designed and written by members of the C-128 design team. There are many versions of C and Pascal available for the C-128 such as Super C 128, the Proline C compiler, Kyan Pascal 128, and Super Pascal. There is also a version of COMAL 2.0 product enhancement called the "super chip" which adds several enhancements to the COMAL language and activates many of the C-128's features available to the COMAL programmer.

Susan: Susan is the regional manager of a merchandising company. She is looking for a full featured telecommunications package for her C-128 that will allow her to communicate with the main offices mainframe system as well as microcomputers in other regional offices. The telecommunications package she is looking for must have a large text buffer and have the ability to communicate effectively at at least 1200 baud and preferably 2400 baud.

Recommendations: Susan ought to consider Bobsterm Pro 128 or Sixth Sense 128. Both of these packages will communicate effectively at the speeds she requires. Both packages feature extensive buffers, Bobsterm Pro has 60k, Sixth Sense has a 64k buffer that is expandable through the addition of either the 1700 or the 1750 ram expanders. Both programs include built-in text editors for buffer text manipulation and programmable macros. Bobsterm also includes a special remote mode which turns your C-128 into a mini-bbs system allowing you or other people to access your computer remotely.

Allen: Allen is an advanced mathematics instructor at a college preparatory high school. He wants to use his C-128 to help him with various record-keeping chores such as student scheduling and grade calculating. He also wants to use his C-128 to prepare worksheets and study guides for his class. Allen also wants to use his C-128 as a teaching aide to demonstrate advanced mathematical and statistical concepts via Basic 7.0.

Recommendations: Allen ought to look into a good spreadsheet and/or database for his recordkeeping needs. We have mentioned several of each above, but to recap we like the following spreadsheets: Vizastar 128, Multiplan 128, Pocket Planner 128. As for databases we like: Superbase 128, Datamaster 128, Pocket Filer 128, and Record Master 128. For his worksheet creation needs, we recommend Pocket Writer 128 because of its ease of use and "see what you get" mode of operation which will make design of his worksheets easier. He also might consider GEOS 128 since its ability to mix text and graphics would greatly enhance his ability to create effective study materials. Allen might also want to consider the acquisition of one of the Basic 7.0 compilers mentioned earlier since they will increase the speed of the operation of the Basic 7.0 programs he writes for class demonstrations.

We hope that these "hypothetical" recommendations help you with your own software choices and application ideas. Our recommendations should not be your only source of information before you purchase your software. In addition, keep in mind that we have not personally used every piece of software available for the C-128 and there may be some packages which will be released soon or ones which we have inadvertently missed which may also be suitable for these scenarios.

Freedom Assembler 128 *by Miklos Garamszeghy*

The Freedom Assembler 128 is a full-featured symbolic assembler for the C-128 with a twist: it comes on a ROM cartridge so that it loads instantly and does not use up any of the computer's RAM. My first impression of this arrangement was a flashback to the days of using the VICMON cartridge on my VIC 20. (Remember that beast from the dinosaur age of computing?) In order to use Freedom, I first had to remove my trusty 1750 memory expansion cartridge. A drag, I must admit, but workable because Freedom doesn't support the extra Ram anyway. It would have been nice if Hughes Associates had designed the cartridge so that the Rom chip could be removed (it was soldered in place along with several other components) and inserted into the extra Rom socket inside the C-128. After all, what good is an internal ROM socket if no one takes advantage of it?

I must admit that this is my first crack at using a symbolic assembler on the C-128 in native mode, although I have had considerable experience using CP/M and MS-DOS based assemblers. I have even stooped to using COMPUTE's LADS assembler on the C-64 on occasion. Up until now, I have been writing all of my C-128 machine language source code with BASIC 7.0's MONITOR command. This is adequate for short to moderate length ml routines, but is an absolute pain in the backside for debugging and rewriting longer code. Thus the advantage of using a symbolic assembler: when debugging, rewriting, adding or deleting code, you are not concerned with absolute machine language addresses for branches, subroutines, jumps, etc. Addresses are referenced by symbolic labels (or variable names, to use a familiar BASIC term). All the absolute address calculations are handled automatically by the assembler as you add or delete code to your source file. For the benefit of the novice, both a monitor (like BASIC 7.0's MONITOR command) and an assembler (like Freedom) are designed to convert a series of mnemonic statements, often referred to as assembly language, into machine executable code, or machine language. (The microprocessor itself can no more understand assembly language or BASIC commands than it can Egyptian hieroglyphics. Hence the need for a monitor or assembler program to act as an interpreter to translate the mnemonics into machine code.) The major difference between a monitor and an assembler is their scope of operation. A monitor is generally a single line assembler in that it only converts one line of mnemonics to machine code at a time without looking at either what it has already done or what it is about to do. Thus all references to other parts of the code must be addressed manually by finding the correct entry points. The more complicated a piece of code is, the more difficult it is to keep track of all the various branches. An assembler, on the other hand, works on the entire set of many lines of instructions as a whole. Alpha-numeric labels are used in place of absolute addresses and/or byte values in the instruction file. The absolute values and addresses are calculated later by the assembler as it converts the mnemonics into machine code.

Freedom has a rich vocabulary of pseudo ops and is fairly simple to use once you get used to the somewhat unorthodox syntax of the pseudo ops. (A pseudo op is a command recognized by the assembler which is not a true assembly language statement, such as a command to define the starting point of the program in memory, or to define an address label.) More on this later. Program source code is entered into the computer with the normal C-128 BASIC editor in the usual form: a line number followed by some statements. In this case, however, the normal BASIC statements are replaced by assembly language ones such as LDA #$00. Freedom recognizes the BASIC REM statement as its normal function, that of a non-executable comment statement to document the code. Bank 0 is used to store the mnemonic source code in a fashion similar to a normal BASIC program. The available memory contains enough space to give about 4k bytes of assembled machine code. Linking to external files for longer source codes can also be done.

The mnemonics for the assembly language statements are identical to those used by the MONITOR command, i.e. standard 6502 type format. Freedom also supports several extended sets of mnemonics for specific chip variations such as the 65C02 used in some non-Commodore machines. Presumably, this is to allow you to write source code for custom hardware based on 6502 type microprocessors.

After entering the statements (or LOADing them in as a program from disk or tape), the assembler is activated with a SYS 57000 command. (For once software has been designed for users who think in decimal! 57000 is a nice easy number to remember. A round hex number like $d000 is murder to remember in decimal, 53248, for a SYS command.) Freedom works with either the 40 or 80 column display in both FAST (2Mhz) and SLOW (1Mhz) modes. Of course, the 40 column screen will be blanked out in FAST mode. A command is provided for switching between modes from within the assembler. In a few seconds (depending on the length of your source code), you have a completely assembled ml program, almost ready to run. The only problem is that the program is in Bank 1 instead of the usual Bank 0. In order to run the program, it must first be transferred to Bank 0, either with the MONITOR T(ransfer) command, or with a combination of BSAVE (from Bank 1) and BLOAD (to Bank 0). This is but a minor inconvenience which is more than compensated for by the extra memory and added versatility you get by operating this way.

Freedom Assembler Review Continued

So far so good. Now for a few sore points. To someone raised on a diet of CP/M and MS-DOS assemblers like RMAC and MASM, the syntax of most of the Freedom pseudo ops is somewhat awkward. For example, in MASM (and keeping in tune with the western style of reading left to right) used on the IBM-PC, a label is declared in the form:

```
label=expression
```

In Freedom, the equivalent is:

```
=expression,"label"
```

While this syntax is equally valid for what it does, it seems harder for most people to learn, especially if they have used other assemblers before. Even most Commodore compatible assemblers, such as LADS, follow the left to right type syntax. In addition, the need for quotation marks around all label references, both where they are defined and where they are used, is a potential source for errors. (Believe me, it is very simple to leave off a quote mark when writing source code!) Other assemblers, such as MASM and RMAC, allow you to redefine labels in the middle of a program. They also support much more sophisticated conditional branching pseudo ops in the form of IF THEN type statements. Neither is possible with Freedom. Freedom is designed to work from tokenized source code files. While this has the advantage that you can enter them like a BASIC program with the normal C-128 screen editor, you cannot use files created with a word processor or some other types of assemblers. The editing facilities of BASIC are good enough for short segments of code, but leave a lot to be desired in terms of sophisticated features such as search and replace which are available on even the most rudimentary word processor.

The reference manual is skimpy but adequate for its intended audience of experienced programmers. It does not delve into the basics of machine language programming such as how to use the various assembly language statements (such as LDA #00), but users of Freedom would generally have a good working knowledge of this anyway. Besides, there are many good texts available about 6502 assembly language programming. The manual does not mention that the C-128 must be in the default Bank 15 before you activate Freedom with the SYS 57000 command. This minor omission could cause serious headaches for even an experienced user. The correct start up command should be BANK 15:SYS 57000.

All in all, the Freedom Assembler 128 is decent program for the moderate to advanced user who writes piles of machine code. For shorter ml programs, however, perhaps the monitor isn't so bad after all.

PaperClip II *by Randy Margolis*

Imagine all the features you desire in a microcomputer word processing package. Think of what is available in your current program. Chances are that the recently released PaperClip II has most everything you thought of. I don't think I would be stretching things too far to suggest that this offering is one of the most complete word processors available for an eight bit computer. And, it certainly is superior to many that I've personally seen running on IBM-type computers; programs costing many times more than PaperClip II. In addition to being one of the most full-featured word processors available, PaperClip II adds a previously unheard of enhancement: a complete telecommunication module which is always just a keystroke away! This integrated terminal program is so complete that it could be sold as a separate program itself. More about this later.

PaperClip II is a 'post-formatted' word processor, which means that special formatting characters are included in the text. However, the characteristic which turned many people off to the previous versions of PaperClip, the lack of a word wrap mode has been corrected with the inclusion of a default word wrap mode which may be switched off for charts and tables, or if you just like working without it. On-screen underlining and boldface is supported in the video preview mode only.

Many of the improvements in PaperClip II are small convenience type features. For example, if you place a comment in the first line of your text with the name you wish to give the file, PaperClip II will offer this as a filename when you invoke the "save file" function. This type of convenience makes the program quite pleasant to work with.

A major change in the program becomes evident when you press the F2 or F4 keys. Each controls a different pull-down menu with which you configure the program to suit your individual needs. F2 displays the 'Screen Options' menu, in which you set the default screen colors for text, background, and the various text enhancements (underline, bold, etc.) during video preview. The F4 key invokes the 'File Options' menu. From this menu you select from approximately 30 printer files provided on the distribution diskette. These inform PaperClip II which printer you are using, and control it accordingly. In the unlikely event that your printer is not supported, you can construct your own printer with a separate utility program supplied. If you've ever wished that C-128 applications handled two single disk drives more effectively, your prayers are answered by PaperClip II. On the file options menu there is a choice called 'Drive Arrangement', which toggles between one and two drives. If you select the latter option, PaperClip II treats

PaperClip II Review Continued

your two drives as one dual slotted device with slot '0' and slot '1'. Hereafter, whenever you call for a text file to be loaded, both drives will be searched for the file, setting up a kind of 'search path' for this common equipment arrangement. Speaking of text loading, PaperClip II uses the burst mode of the 1571 drive for truly awesome loading speed. If the program senses that you have a 1541 drive, it invokes a different type of fast loading routine which should speed up these loads as well. If you do have two drives, a 'Global Copy' command will copy a chain of linked text files between them with no user intervention, making backups simple. The 'Sequential Format' menu selection has two choices: either the familiar Commodore ASCII or you can opt to save sequential files as True ASCII, which greatly aids in constructing text uploads for many bulletin boards.

After you have made your selections from the two menus, choose 'Save Current Configuration to Disk' and, if you name this file as suggested by PaperClip II, the configuration will be automatically loaded each time you start the program. You can have as many other configuration files as you want and load them from within the program, supplanting the defaults.

The text editor is very similar to previous versions of PaperClip, which is to say very easy to use. Enhancements include being able to place an invisible 'bookmark' in the text to which you can return with a single keystroke at any time. Two separate bookmarks can be set at a time. Also, any common phrase can be assigned to a single key. This makes repeatedly typing out a long, frequently used phrase a thing of the past. Just hit <ESCAPE> and your chosen key and the phrase will be typed for you. The left arrow key above Control now serves as a 'delete right'. This erases the character under the cursor and pulls in the following text, giving you two distinct options for this function (the other being the normal Inst/Del key). The F1 key jumps down one screenful of text, and the F3 goes in the reverse direction. To rapidly move to the end of your text file, tap the Help key and you will be there instantly. Likewise, touch Clr/Home twice and you will be transported to the top of text in a flash.

All the former column commands have been retained. For some people, these capabilities alone are reason enough to use PaperClip word processors. This very magazine uses the column functions to help create the dual column layout. You may also use these functions to act upon columns of numbers. There are commands which perform arithmetic on both columns and rows of numbers and will place the result (completely formatted with commas and dollar sign, if you wish) anywhere in the text.

Since this is a post-formatted product, many will want to use the Video Preview command to check their documents before printing. A brand new feature permits you to reverse the direction of video scroll to view prior portions of the text. Also, during the preview, underlined text will be underlined on the screen, and other enhancements will show up with the colors chosen on the 'Screen Options' menu. If you tap the 'C' key, video output will switch to continuous instead of halting at each page break. However, in single page mode you can switch back and forth between viewing and printing at the touch of a single key. Indeed, you may even print the page just viewed! These abilities, coupled with the option of stopping momentarily for a little editing and picking up your view/print at the same point, illustrate the enormous flexibility of PaperClip II in regard to output options. The inimitable 'checkmark' commands are retained. These are the formatting commands which are placed in text to control margins, pitch, headers, footers and so on. However, you can now use English words instead of the former cryptic letter combinations. For instance, "centering on" is now equivalent to the old "cn1". This assures compatibility with all previously written PaperClip text files.

If you have a modem, you'll love what happens when you press the NO SCROLL key at the top of the keyboard. This drops you instantly into the Communications, (Terminal) mode. Even though your text has disappeared from the screen it is still present in the text buffer (hit NO SCROLL again to exit the terminal) which doubles as a 39k capture buffer for telecommunications. In addition to capturing into this buffer you can also send it, thus enabling you to compose a bulletin off-line and send it after connecting to your favorite BBS.

Two pull-down menus allow you to set up for communications use. The first is called 'Communications Options' and in it you set such things as your modem type, baud rate (up to 9600!), what control characters do, and the two software clocks. One of these clocks is a stopwatch for keeping track of time on a system; it automatically resets and starts upon connection. The other is a time-of-day clock which you set from the menu. Eight different modems are supported, including one for direct connection to another computer with a cable. This permits 9600 cps file transfers between two computers connected together with an RS232 cable. The modem list includes the 1600, 1650, 1660, 1670, Mighty Mo, HesModem II, and Hayes. One of these should be just right for your equipment.

Hit F3 and the 'Dialing Options' menu pops down. On this menu is room for ten different phone numbers which can be remembered by PaperClip II and dialed automatically if your modem supports this. The program is intelligent enough to remember the particular communication setting which were in force when you enter each of the numbers and reconfigures

PaperClip II Review Continued

the system accordingly when you choose to dial that number. You can even 'tag' certain selections from the phone list and instruct PaperClip II to dial that tagged list over and over until a connection is established.

In addition, the terminal can be set to answer the phone (if your modem is capable) and act as a mini-BBS. You don't even have to wait for a call, however. Suppose you are talking to a friend on the phone, and decide to transfer a file to him. After deciding who will be the originator of the call, you can, without hanging up, instruct PaperClip II to "pick up an extension" and start telecommunicating with your friend's computer at the other end of the line, after which you can break the computer connection and resume your voice conversation. This is one feature I have never heard of before, but one which sounds extremely useful nonetheless. As for file transfers from disk, protocols available include Punter C1, Xmodem, and Xmodem CRC (an improved version). Using the proper protocol for the service you are calling, these transfers work flawlessly.

The settings on both these menus are saved into the configuration file on your boot disk by choosing that option on the 'Dialing Options' menu. I hope the possibilities of this arrangement are becoming clear. By having both programs combined into one you can work on a document, occasionally switching to the terminal to call your favorite on-line service and, if the line is busy, simply zap back to the word processor and continue working.

Have you ever captured a message from a bulletin board, saved it to disk, and then spent hours removing the carriage returns and formatting the text? Well, PaperClip II solves this problem automatically for you! A special Unformat command in the word processor will remove all carriage returns (except those on an otherwise blank line) and format the text for you instantaneously. You'll find this an amazingly useful feature and you will find it nowhere else.

Unfortunately, no program is perfect, and PaperClip II's flaw in its spelling checker. There seems to be a serious bug in the integrated spelling module. The supplied dictionary contains over 38,000 words and it will check your document. However, when adding words to your personal dictionary (there is room for about 70,000 on a 1571 formatted disk) PaperClip II runs into trouble. After adding words to the dictionary, the program kept asking about words which should already have been added to the disk. Although most of the additions are made, there will always be a couple which don't make it into the word files. Checking is relatively speedy, but this bug must be eradicated before the module can be depended upon. Since I received the first production version of PaperClip II, I trust that Batteries Included has been made aware of this problem and is diligently working on a fix.

All in all, I can heartily recommend this software to anyone interested in putting words on paper. You may also forget where you stored all those public domain terminal programs after using the built-in communication software for awhile. PaperClip II is protected by a key or 'dongle' which must be plugged into the joystick port (allowing unlimited backups of the disk), and retails for $79.95. In my opinion, a real bargain!

The Timeworks Series *by Miklos Garamszeghy*

The Timeworks 128 series is billed as an integrated word processing, spreadsheet and database system for the C-128. The series consists of three programs, each sold separately, called Wordwriter 128, Swiftcalc 128, and Datamanager 128. The programs feature "easy to use" keyboard-operated pull down command selection menus for most operating commands by using the <esc> key to select the menu mode then the cursor movement keys to select items from the menus. The programs do not seem to support a mouse for the menus. If you like this type of command entry system, then perhaps the Timeworks series is just the thing for you.

First, my impressions about Wordwriter 128: this review was <u>not</u> written with it! (That perhaps says it all.) After using Paperclip for a number of years and more recently Paperback (Pocket) Writer on my C-128, I have become accustomed to a certain way of using word processors. (You can't teach an old dog new tricks, and anyone over 20, which I am long past, is certainly an old dog by computer standards.) Don't get me wrong, Wordwriter 128 is a very good and powerful word processor in its own right. It works with the 80 column screen and supports a wide range of text editing and formatting commands (all of which are shown on screen), document chaining, form-letter printing and a 85,000 word spell checking dictionary with options to add more custom words to sub-dictionaries of your making. All of the usual disk commands are supported, such as directories, formatting a new disk, etc. There is even a built-in calculator mode, independent of the main text screen, which can be handy. Numerous times I have tried to do a mental calculation while in the middle of writing something. This could be handled quite easily by the calculator mode of Wordwriter.

Now for a few downers. First of all, lines are limited to a maximum of 250 characters. Although this is still well within the bounds of reasonable size, wide tables may require longer line lengths. The text files are saved in SEQ format only. PRG type screen code text files are not supported at all. This would not normally be a problem, except

Timeworks Series Review Continued

that the files are in true ASCII, not PETSCII. Thus text files produced with Wordwriter are not generally compatible with other C-128 word processors, or vice versa. This also means that you cannot use Wordwriter to edit or create PETSCII style data files for other programs. An external utility program is supplied, however, on the program disk for converting files from other word processors. In addition, two files are saved for each document produced: one main file containing the text and a second file, with the characters .PAR tacked onto the end of the name, contains the printer control codes and formatting data. The .PAR file is created using the control code sequences that were entered using the "Set Printer Codes" option of the print menu. This option allows you to set the code sequences for features such as underline, double strike, superscript, etc. for your specific printer. If you want to use these codes again, you must load in the .PAR file that they were saved in before you enter your new text. This is equivalent to selecting a printer file with most other word processors (such as Paperback Writer and PaperClip) except that the control codes can be easily changed without first exiting from the document editing mode. This would normally be a rare occurrence once you initially set up your printer, however, unless you frequently change printers in the middle of editing a document.

A few of the keys used by Wordwriter are not quite what one might expect. For example, the <del> key deletes the character under the cursor (not the character to the left of the cursor as it normally does) while the back arrow "<--" key (above the <control> key) works as a sort of backspacing delete key (like the normal <del> key), except that the remainder of the line does not shift back to fill in the void. For a program that seems to be big on pull down screen menus, the really useful ones, such as print formatting (margins, centering, justification, etc.), have been left to manual entry from the keyboard with "check mark" type commands. The formatting changes are not visible immediately, you must first select the Format function from the main level of pull down menus.

The spelling checker is adequate as these things go. There are, however, some minor flaws to this one. The maximum document size that can be spell checked is about 10 pages, although longer ones can be worked with in non spell check modes. Longer documents must be composed of linked files for spell checking. Up to 255 pages worth of material can be linked for either spell checking or printing. Although there are provisions to create your own sub-dictionaries, the main dictionary is in an encoded format so that it is not possible to edit or add to it. This may be of concern to professionals who use a lot of specialized jargon and/or foreign languages in their writing.

SwiftCalc 128 is a very good full featured electronic spreadsheet for the C-128. It is designed to work with an 80 column display only. The addressable range of spreadsheet cells is 250 rows by 250 columns. This is a total of 62,500 cells. The actual usable range will be limited by what is in the cells. For example, there is simply not enough memory in the 128 to contain a number in every cell (62K cells x 4 bytes per cell (floating point number) = 248 K bytes!, not including cell addresses, overhead, etc). The program uses the same style of pull down menus that Wordwriter uses. For IBM-PC affectionados, this gives it the look of a scaled down version of Lotus 123. SwiftCalc incorporates numerous commands for copying information from cell to cell, moving cells, changing the cell format and/or width, etc. In fact it contains all that you could want in a spreadsheet program. The auto calculation mode is a good feature included with all the best spreadsheets, however, in the C-128 it tends to be a little slow, especially if there are many formulae in the spreadsheet.

The graphics are relatively good and are, again, displayed on the 80 column screen. They do, however, suffer some serious limitations. The first is that their use is only mentioned in the "supplemental" manual. Another is that, with the exception of the 3-d bar chart, you can only display one variable at a time as a function of a label entry. This, in effect, means that you cannot plot a simple X-Y type graph. The X-axis is always a label entry (such as a name or date etc). This may be adequate for home budgetting type purposes, but it is certainly not good enough for serious scientific, financial or engineering type calculations.

SwiftCalc is supplied with an interesting little utility called "Sideways". This program allows you to print out your spreadsheet report sideways with a dot matrix printer. This is especially useful if your spreadsheet contains many wide columns. When printed normally, only portions of a wide spreadsheet (usually only about 80 characters wide, depending on the width of your printer) will fit on a single page. Sideways takes advantage of continuous fan fold paper to, in effect, give your spreadsheet printout infinite width. Unfortunately, by doing this it also limits the number of rows that you can print in a single pass. It is, nonetheless, a gimmick that many people will find handy, or will find a use for. The program is available separately for the C-128, C-64 and numerous other computers such as MS-DOS models.

Datamanager 128 is a relational database program for the 80 column C-128. It allows you to store information (or data) in user definable "records" which can be indexed, sorted and stored for later retrieval. As with most other good database type programs, you can design the format of your data entry screens to suit the needs of the

Timeworks Series Review Continued

application. The screen format can be saved for later use in creating other Datamanager applications. One word of caution when using Datamanager. You must start a new database file on a completely blank disk, otherwise the program will yell disk errors at you. However, Datamanager contains no visible disk commands which would allow you to format new disks. If you do not format a new supply of disks before you try to create a new database, you will not be able to save it! (The fine print of the manual does mention this, but it took me a good deal of frustration before I eventually found the problem.)

Datamanager can incorporate password protection on your files so that only authorized personnel can view or change them. Timeworks cautions users not to forget the password once it has been assigned because the program contains no obvious method for recovering password protected data. The files can be searched and sorted by any one of a number of keyword fields. A special X-SEARCH feature allows multiple features to be compared for a match. For example, you could search your file for the address of a friend name John (assuming you had many in this category), whose wife was named Sue, and the couple had 3 children, each of which was born on a Tuesday afternoon at 2 o'clock. Of course, your database would have to contain the relevant data in a searchable field to be able to find this kind of a match in search.

The report writing utility is one of the most useful features of a database. It allows you to extract the information you want from the database and print it out in the format that you want. As all good report writers should be, Datamanager is very flexible, allowing you full control over what is printed where. The report writer can be used in conjunction with the various search and sort routines to print out only the records that you are interested in. The report can be output to the screen, a printer or a disk file for later use. Special utilities are provided for printing out address labels and name tags, etc. The label information can also be used by Wordwriter for form-letter creation.

The Timeworks series is billed as an "integrated system". In my view, this should mean the ability to share files between the various applications programs. For example, a data file produced with the spreadsheet should be editable by the word processor and then reusable by the spreadsheet afterwards. What often passes for integration, I'll call it semi-integration, is that one application, say the database, can write a text file to disk in the same manner as it would to a printer. So far so good. Any word processor should be able to read this file. The only problem is that the database cannot read it back in again. Thus true two-way integration is lost. In this regard, the Timeworks series is only semi-integrated: once the database and spreadsheet files have been deliberately converted to the word processor format, they cannot be read back in by the original application. I can equally well read in the converted file with another word processor, such as Paperback Writer. Truly, this does not make it an "integrated system".

Individually, each of the applications programs in the Timeworks series is good, with SwiftCalc, perhaps, being one of the better spreadsheets available for the C-128. The manuals for all three programs, while thin, are generally well written and easy to understand. The command structures and menus are very user friendly. Each of the programs would be a good bet for the novice to intermediate level user. If you like the command structure of one, you will probably like them all!

Expanded VDC Ram Programming *by John Kress*

In last month's article the replacement of the VDC 16K ram chips with the larger 64K rams was detailed. But now that they've been replaced, how do you use them?

First off, you will need to tell the VDC chip that the ram has been expanded. When that is done you'll find that everything on the screen is garbled. This is due to the fact that the 64K chips are addressed slightly differently than the 16K chips. This is all done internally by the chip and the solution is rather easy. After setting the correct bit in register 28, simply copy the rom into the VDC ram via the kernal routine DLCHR. The following is a ML outine that both sets the proper bit in register 28 of the VDC and recopies the ROM character set into the VDC ram. To see the results of just the 64K initialization process, simply change the JSR in line 2022 to a JMP.

Enter the program through the 128's Monitor, and save it to disk by entering S"INIT64" 08 2000 2028 and return. To activate the program, just enter SYS 8192 and you're on your way.

In the above example I wrote out the routines to read and write to the VDC registers, but if in your programming you would be using bank 15 there are routines in the Rom you can use. They are SCDCC to write a value to a register and CDDA to read a value from a register. You can also use these routines from Basic in the form: SYS DEC("CDCC"),A,X. . this form you would use the accumulator for the register value you wish to store, and load the X Register with the register you select to write to. In the case of reading the register's values you'd use: SYSDEC("CDDA"),0,X

Expanded VDC Ram Programming *by John Kress*

and get the value into a variable by A=PEEK(6).

In my experience, I have not yet found any compatibility problems with a software program and the 64K VDC chips. I've tried many programs with the VDC chip initialized to 64K and 16K and all programs have worked as intended. CP/M mode doesn't seem to be affected by this either.

Graphics Screens

In my experimenting I've come upon a few items of delight that were not realized before the 64K chips were installed. One of my major interests with computers has been graphics capabilities. Now with the upgrade completed I've been able to put together a graphic program using the attribute memory for color assignments on a graphic screen. One of the first things I found is the upper nibble (higher four bits) of the attribute byte for each 8*8 pixel area now controls the background color for that 8*8 area. In normal text mode the upper nibble of the attribute ram contained the information for flashing characters, underlining, reverse display, and alternate (lower case). So in graphics mode you can have 2 colors within an 8*8 area, and with care in preparing a graphic screen you could have quite an impressive looking display.

Also a nice feature is the ability to flip pointers to a different screen. In a matter of 40 or so bytes of machine code you can change instantly from one type of screen to another. You can change from a graphic display to a text screen and back without having to transfer the graphic display out of VDC ram to a safe area or having to recopy the character information, which is overwritten with 16K chip graphics. The ability to have an instant menu screen appear at the touch of one key without delays is very useful. There is the possibility of graphic screens in 3 different areas and flipping the pointers for animated display, manipulating one screen while another screen is being displayed.

Text Screens

Allowing for 16K of character information, there is a possibility of having 12 different text screens in memory at one time. Although the VDC routines in rom and use of the kernal routines to print will only work with the one screen that resides at $0000, I've found that the block copy routine, which moves a block of up to 255 bytes of VDC ram, can transfer one complete screen into another area of ram almost instantly. Once a screen has been established you can set the pointers to display that screen or, if you need to manipulate the screen, transfer it back to $0000 and do what's needed.

Even though a screen resides in a different area of ram than the original screen area, you can still manipulate it. You will have to do a bit more work on it and have to keep track of the areas that you're processing. With the many features of the VDC and using the registers, you can quite easily manipulate any area of ram.

```
02000  4C 1B 20 JMP $201B ; Jump around the subroutines
02003  8E 00 D6 STX $D600 ; This is a subroutine
02006  2C 00 D6 BIT $D600 ; to store the value in
02009  10 FB    BPL $2006 ; the Acc. to a register
0200b  8D 01 D6 STA $D601 ; selected by the value
                            of the X register.
0200E  60       RTS
0200F  8E 00 D6 STX $D600 ; This is a subroutine
02012  2C 00 D6 BIT $D600 ; to get the value of a
02015  10 FB    BPL $2012 ; selected by the number
02017  AD 01 D6 LDA $D601 ; in the VDC register
                            that is selected by
                            the number in the X
                            register.
0201A  60       RTS
0201B  A2 1C    LDX #$1C  ; Select register 28 and
0201D  20 0F 20 JSR $200F ; read the value in it.
02020  09 10    ORA #$10  ; Set bit 4 of the byte
02022  20 03 20 JSR $2003 ; and replace new value
02025  4C 0C CE JMP $CE0C ; in reg.  Re-do the
                            character set in ram.
```

The Assembly Line *by John Kress*

Assembly language programming is getting right to "the heart" of your C-128. Many Twin Cities 128 readers have expressed an interest in learning some of the more technical tricks, that they might make their C-128s jump through hoops. With this in mind, we kick off a new series entitled: The Assembly Line. This series will give both the beginner as well as the more advanced assembly language programmer insight into the exciting world of machine language programming on the C-128.

The Assembly Line Continued

The first new routine to be covered, actually two very closely linked routines, are INDFET or FETCH and INDSTA or STASH. These two are very similar in the way that you prepare pointers before using the routines, and go hand in hand with each other. Their function is to either store a byte or get a byte from another bank in memory, using zero page pointers to the address where you want the byte to go or be placed. Some of the uses for these routines would be to save an area of ram into a 'safe' or unused portion of another bank, to copy a graphic screen into BANK 1, or to transfer information out of one area and save a copy of it, in another bank, and continue manipulating the original copy.

INDFET or FETCH ($FF74) uses one Zero Page address and the .A..X,.Y registers, and the returned value is in the .A register. In preparation you'll need to set up the target address of the byte to be retrieved in two Zero Page bytes, in the typical low byte, high byte manner. Then you load the .A register with the Zero Page pointer, .X register is loaded with the target bank, and if needed the offset to the Zero Page Address is loaded into the .Y register. If no offset is needed load .Y with #$00.

A routine to get 255 successive bytes from bank 1 memory and to store those byte in bank 0 would read like this:

```
04000  A9 00     LDA #$00 ; Set up the target
                            address
04002  85 FA     STA $FA  ; of $2000 and store in
04004  A9 20     LDA #$20 ; Zero Page at $FA, low/
04006  85 FB     STA $FB  ; high format.
04008  A9 00     LDA #$00 ; Set up storage address
0400A  85 FC     STA $FC  ; of $1D00 and store in
0400C  A9 1D     LDA #$1D ; Zero Page at $FC, low/
0400E  85 FD     STA $FD  ; high format.
04010  A0 00     LDY #$00 ; Start the .Y offset at
                            $00
04012  A2 0D     LDX #$0D ; Select bank 1 for FETCH
                            BANK
04014  A9 FA     LDA #$FA ; LOAD .A with Zero Page
                            pointer.
04016  20 74 FF JSR $FF74; And FETCH the byte.
04019  91 FC     STA ($FC),Y ; Store the byte at
                               Z.Page addr.
0401B  88        DEY      ; decrement the offset
0401C  D0 F4     BNE $4012; and check if full
                            circle to zero yet
0401E  60        RTS      ; If zero then done.
```

The STACH or INDSTA routine is much the same, with one exception. Since the .A register will be holding the formation that is to be transferred, the Zero Page address cannot be stored in the .A register. To get around this problem the Zero Page pointer is stored in the common ram area at $02B9. Otherwise the routine is identical.

A routine to store in bank 1, 255 successive bytes would look like this:

```
04000  A9 00     LDA #$00 ; Set up the target
04002  85 FA     STA $FA  ; address for the STASH
                            routine to be
04004  A5 20     LDA $20  ; Zero Page addressed at
04006  85 FB     STA $FB  ; $FA, Low/Hi fashion
04008  A9 00     LDA #$00 ; Set up the bytes to be
                            transferred
0400A  85 FC     STA $FC  ; and Zero Page pointer
0400C  A9 1D     LDA #$1D ; at $FC, Lo/Hi fashion
0400E  85 FD     STA $FD  ;
04010  A0 00     LDY #$00 ; Set .Y offset to #00
04012  A9 FA     LDA #$FA ; Load the zero page
                            pointer
04014  8D B9 02 STA $02B9  ; and store in common ram
04017  B1 FC     LDA ($FC),Y ; Load the STASH
                               target byte
04019  A2 0D     LDX #$0D  ; Choose Bank 1 and
                             Kernal
0401B  20 77 FF JSR $FF77  ; STASH the .A register
0401E  88        DEY       ; Decrement the offset
                             pointer
0401F  D0 F1     BNE $4012 ; Check if it's zero
04021  60        RTS       ; If zero then done
```

One more note about these routines, the .X register is loaded with the value you chose for the desired memory configuration. For a complete description of these configurations, see the Commodore 128 Programmers Reference Guide, pages 458-465.

Forgotten BASIC *by Miklos Garamszeghy*

In simple terms, a light pen is a device which contains a small photocell or light sensitive semiconductor which can be used for "reading" areas of a display screen. This is where the simple part ends. Suffice to say, that by sisticated synchronization circuitry, the position of a light pen pointed at a display screen can be determined. A ght pen can be useful in graphics programs (you can "draw" with it on the screen just as you would on paper with a normal pen) and for rapid selection of program menu options. Indeed, because of its ease of use as a data entry tool,

Forgotten BASIC Continued

light pens are becoming more popular for computer assisted drafting (CAD) programs for mainframe, mini and micro computers as well as with commercial data processing software.

The C-128 supports a light pen in both 40 and 80 column modes, although they work somewhat differently in each mode. There are already several graphics packages available for the C-128 (such as CADPAK-128 from Abacus Software) which can use a light pen. In addition to the commercial products, it is very easy to incorporate light pen routines into your own programs in either Basic 7.0 or machine language.

BASIC 7.0 contains a function, PEN(x), which allows you to read the coordinates of a light pen. The format of the function is: Z = PEN(x)
where: x = 0 for the X (horizontal) coordinate of the 40 column screen (in pixels)
1 for the Y (vertical) coordinate of the 40 column screen (in pixels)
2 for the X coordinate (column) of the 80 column screen (in characters)
3 for the Y coordinate (row) of the 80 column screen (in characters)
4 for the synchronization signal for the 80 column screen

The important difference to note between the 40 and 80 column modes is that in 40 column mode, PEN always returns a value in pixels even when in text only mode (each character is 8 pixels wide by eight high), while in 80 column mode, PEN always returns a character position (row and column). This is due to the difference in the way that the 40 and 80 column video chips operate. The 80 column chip also sends out a synchronization or trigger signal which indicates that the timing loop is starting for light pen reads. (The actual position of the light pen is calculated by the time it takes the raster scan to be detected by the pen from the start of the sync signal.)

The numerical values returned by the PEN function depend upon a number of factors in addition to the position of the light pen. These include the characteristics and sensitivity of the pen itself and the type, brightness and contrast of the monitor. Because of this, you will have to "calibrate" a light pen each time before it is used. This is easiest done by touching the pen to a known position or calibration point on the screen (say a dot in the middle of the screen) and comparing the known values for the X and Y coordinates with those measured by the PEN function. The true values for future reads will be the measured value plus the difference between the calibration known and measured values.

Because light pens read "light", they work best reading the position of bright objects on dark backgrounds. Therefore, for calibration or menu selection type applications, you should use a white mark on a black background, while for "drawing" on the screen, you should have a light background with dark drawn lines.

The light pen in 40 column mode is controlled by the 8564 video chip registers at 53267 ($d013) for the X position and 53268 ($d014) for the Y position. The screen is 320 pixels wide, therefore you need 9 bits to read the horizontal screen position. Unfortunately, there are only 8 bits available. Therefore, horizontal light pen resolution is to the nearest two pixels. The screen is only 200 pixels in the vertical direction. Therefore 8 bits is sufficient to allow 1 pixel resolution in the Y direction. Bit 3 of the VIC chip interrupt register at 53273 ($d019) will be set to a 1 when the light pen has been triggered.

Reading the 80 column chip light pen registers (and indeed any of the 80 column registers) is a lot more difficult than for the 40 column chip. The 8563 80 column video chip contains only two directly accessible registers: the address register at 54784 ($d600) and the data register at 54785 ($d601). These two registers are used to pass data between the 37 internal registers of the chip and the external world. The vertical (Y) light pen position is read from internal register 16 ($10), while the horizontal (X) position is read from register 17 ($11). Bit 6 of the address register is turned on when a light pen interrupt has occurred. To read the light pen registers, you must POKE (or STA in machine language) the address register with the register number (decimal 16 for the vertical position, or decimal 17 for the horizontal position) that you want to read, then PEEK (or LDA in machine language) the data register to read the value.

All registers mentioned above are in the Input/Output (I/O) block of BANK 15. They can be read with machine language or with BASIC PEEK's. However, in BASIC it is far more convenient to read the positions with the PEN function.

Listing 1 is a short demo routine for a light pen which will allow you to doodle on the screen. The program also demonstrates the use of other advanced Basic 7.0 features such as the DO-WHILE-LOOP procedure, LOCATE and DRAW. Line 10 sets up the various screen colors for multi color mode. You can change these to any that you want, bearing in mind what I said previously about light and dark colors. Lines 20 and 30 are used to calibrate the pen. Lines 40 and 50 are used to draw lines on the screen. Lines 60 to 90 are used to read the light pen. The routines

Forgotten BASIC Continued

will work with a pen that has an integral "tip switch". If your pen does not have a tip switch or push button activator, you will have to hold down a key (such as <return>) when you want to read the pen. If you press a number key from 1 to 5, you will change the drawing color, while if you press the letter "C" (for clear), you will erase the screen. The letter "F" will fill in (or paint) the area of the screen located by the light pen with the current drawing color. If you wish to save your doodle to a disk file for later use, you can use the Basic command:

```
BSAVE"filename",B0,P8192 TO P16383
```

To retrieve your screen, you will first have to set your colors and then the graphics mode with a: GRAPHIC 3,1 command. The file is then retrieved from disk with: BLOAD"filename"

Listing 1

```
10 color0,12:color1,2:color2,6:color3,7:color4,2:
   trap 100
20 graphic3,1:draw,80,90to80,110:draw,75,100to85,100:
   char,2,23,"touch pen to centre",1
30 gosub60:x0=x-80:y0=y-100:c=1:color0,2:color1,3:
   graphic3,1
40 gosub60:xi=x-x0:yi=y-y0:locate(xi),(yi):draw(c)
50 dowhilejoy(1)=7:gosub60:draw(c),(xi),(yi)
   to(x-x0),(y-y0):xi=x-x0:yi=y-y0:loop:goto40
60 getkeya$:x=pen(0)/2:y=pen(1)
70 ifa$>="1"anda$<"5"thenc=val(a$)-1:elseifa$="c"
   then graphic3,1
80 ifa$="f"thenpaint(c),(x-x0),(y-y0),1
90 return
100 ifer=14thenx=x0+1:y=y0+1:resume40:elseresume40
```

Sparrow's Slick Tips *by Sparrow James*

00101111: Beating problems with reverse relatives: If you think your relatives are a problem, try using any of BASIC 7.0's bit-mapped graphics commands using relative coordinates that are less than zero. According to your System Guide the command: DRAW 1,100,100 TO 80,80 would draw a line on the screen from position 100,100 to position 80,80, but instead it only results in an Illegal Quantity Error. This error is caused by BASIC 7.0's inability to recognize the fact that you are trying to plot to a relative position instead of a non-existent absolute position. Fortunately the soon to be available ROM kernal upgrades for the C-128 will correct this problem, however the following example demonstrates a technique for achieving relative coordinate plotting without the upgrade:

```
DRAW 1, 100,100 TO RDOT(0)-20,RDOT(1)-20
```

The RDOT function returns various information about the current location of BASIC 7.0's pixel cursor. When an argument of zero is used the RDOT function returns the horizontal coordinate of the pixel cursor while an argument of one returns the pixel cursor's vertical coordinate.

00110000: Better directory printouts: When a disk directory is listed to a printer, especially a non-Commodore printer, the disk name header which appears in reversed video on screen can cause illegible results. The reverse video effect can be eliminated by the following: POKE 7173,32

00110001: Overlaying your work: Although BANK 0 of the C-128's random access memory provides an ample amount of memory for BASIC program storage, there are times when it is advantageous to use code overlays in your programs. This can be done from disk by using BASIC 7.0's BLOAD command or from the 1700/1750 Ram expansion via the FETCH or SWAP commands. The key to constructing a good overlay system is planning. If you are not careful, poorly planned overlays will corrupt your code and cause some bizarre results. One of the easiest methods to insure successful overlays is to pad your overlays with colons so that each overlay occupies exactly the same amount of memory of the longest overlay. You can use the C-128's built-in machine language monitor to determine the number of bytes you will need to pad in each overlay. One important caution: When saving your overlays to disk be sure to remember the second memory location parameter of the BSAVE command must be the end memory location of your overlay code (plus padding) plus one.

00110010: 80 column CHAR beware: There is a minor bug in Basic 7.0's CHAR command when used to place text on the C-128's 80 column screen. It appears that a "shadow image" of the 8563 VDC control registers "bleed through" into BANK 0, and corrupt any data, including Basic program text which resides at $D600 and $D601. It appears that after a CHAR command, a value of 15 ($0F) is left in BANK 0 location $D600 (54784) and a value that changes according to the cursor position in $D601 (54785). There have been at least two revisions of this chip in production so it is possible your machine is not affected, but I believe that this bug actually exists in the Basic Roms and not the 8563 itself. Test your machine by executing this program in 80 column mode:

```
10 GRAPHIC 5,1
20 BANK 0: POKE 54784,27
30 CHAR 1,10,10,"THIS IS A TEST"
40 BANK 0: A=PEEK(54784)
50 IF A<>27 THEN PRINT"THE VALUE CHANGED":ELSE PRINT"THE VALUE REMAINED THE SAME"
```

If the value changes when this is executed, do not use the CHAR command on the 80 column screen in long programs.

Rumor/Opinion/Mayhem *by Loren Lovhaug*

In this month's installment of Rumor/Opinion/Mayhem we will take the opportunity to reflect upon rumors from various sources concerning the Commodore 128 in 1986, by presenting a synopsis of 1986's rumors. This is our attempt to wade through all of the bizarre speculation in search of truth and reality.

1. IBM PC "compatibility" for the C-128: In early 1986, fueled by speculation about the now notorious "empty Rom socket" in the C-128 and the 1571's prowess at reading MFM disk formats, many individuals were convinced that an "IBM mode" would become a reality. Of course, for a variety of technical and financial reasons such a modification to the C-128 is impractical, if not impossible. However, it is interesting to note that the C-128 has gained a rather limited form of IBM PC compatibility through the use of the 1571 disk drive and special packages such as Big Blue Reader and Crosslink which allow IBM PC formatted disks to be read and written to for transfer/conversion to normal 1571 or CP/M formatted disks.

2. An enhanced version of the C-128: Throughout the year many people have speculated about the possibility of Commodore marketing an enhanced version of the current model of the C-128. Early in 1986, it was disclosed by Dave Haynie, Commodore hardware engineer, that Commodore Engineering was investigating the possibility of designing a 65816 (the microprocessor at the heart of the Apple //gs) C-128 compatible machine, but this was abandoned because of the high cost of the 65816 and other components crucial to such an endeavor. Over the past year we have heard many stories about possible enhancements to the existing C-128, the most viable of which seem to focus around three major areas: 1) Adding additional Random Access Memory either as internally bank switched memory or external memory (like the 1700/1750 RAM expansion units) packaged with the machine. 2) Expanding the 8563 VDC RAM to 64k (as outlined in the October issue of Twin Cities 128) to enable true multi-color graphics on the 80 column screen. 3) Adding an enhanced version of BASIC 7.0 which supports 80 column graphics, sprites, and true RAM disk support.

While we are not sure exactly what Commodore's long term plans for the C-128 are, we do know that there is a proposal for a motherboard change that is being considered. We think it is doubtful that such a re-design would incorporate more directly accessible internal or external random access memory. However, one of the things being considered as part of the motherboard modification is the inclusion of 64k of VDC RAM. We also know that Louis Wallace and David Darus of C-128 Ultra-Hires fame, and Commodore software engineer, Fred Bowen have been working on an 80 column graphics extension called Basic 8.0 which does support the typical plotting commands, sprite-like structures and the Ram expansion as a high-speed graphics buffer. Indications are that this product will be marketed initially as disk based extension, but nobody has ruled out the possibility of Basic 8.0 being placed on Rom for the empty Rom socket. For next month's issue we are going to try to get an interview with either Louis, David, or Fred and acquire a pre-release copy of Basic 8.0 so we can tell you more about it. Hedley Davis, also of Commodore has developed "true" Ram disk software which is supposed to be fairly transparent which also may be incorporated into Basic 8.0, but for now is intended to be added to the 1700/1750 demo disk and released to the public domain.

3. Commodore is about to release a 3.5 inch disk drive. For a short period last summer this seemed to be in doubt, but we are happy to report (as we have reported before) the 1581, 3.5 inch, 800k disk drive is a reality! In fact, just as we were putting this issue of Twin Cities 128 to bed, we received a Christmas present from Commodore. That's right, a prototype 1581 in an Amiga 1010 case (we have been told the production plastic cases are on the way). Next month, we will be able to tell you more about this fantastic little disk drive (we only unboxed it 24 hours ago). My first impression? File those tax returns early, because it will be love at first byte!

4. The C-128 is dead. My only response to this bogus rumor is: If the C-128 is dead, somebody really ought to tell Berkeley Softworks, Digital Solutions, Xetec, Batteries Included, and the above mentioned individuals from Commodore who are developing C-128 products. They seem to think the C-128 has a viable enough future to invest their time and money to develop "top-of-line" commercial products for this machine. Of course your local "deep throat" did not take the time to consider that fact, nor did he/she stop to consider the C-128s huge installed base (1.1 million units worldwide) and the fact that a microcomputer is only as dead as its users.

5. The legendary C-128 kernal and 1571 Rom upgrades have been cancelled. False although these babies have been delayed for what seems like an eternity, we know that they do exist and should be available shortly. What can we say? We can only tell you what they tell us? So fear not, brave Superbase 128 guru, your 1571 back-side sort concerns will soon be ending (we hope).

6. Twin Cities 128 won't survive one year! Ha ha ha, doomsayers...we made it! Next month we will be publish our 1st anniversary edition full of surprises and information you simply can't find anywhere else. Although we are a sure bet not to make the Fortune 500...ever, I am happy that during our first year we kept our heads above water while steadily improving our quality and content. Thanks, readers and subscribers for investing in our future.

Pocket Writer 2 *by Loren Lovhaug*

The advertisements for Digital Solution's new Pocket 2 series software claim: "The best just got better!" In the case of Pocket Writer 2, I am convinced Digital Solutions has a valid claim. I have several C-128 mode word processors, and while each one of them has their particular forte, my word processor of choice has been Pocket Writer 128 (formerly Paperback Writer 128). Pocket Writer 128, has several outstanding features which make it one of the finest word processors on the market today, all of which have been retained in Pocket Writer 2. In addition to all of the standard text editing and manipulation, some of these outstanding features include:

-True "what you see is what you get" editing including true on screen underlining, italics, boldface, and super/subscripts.

- 64K of document space (about 30-35 pages of standard text) which can be partitioned into two separate 32k work areas so that you can have two documents in memory at one time. This feature is extremely useful for borrowing text from one document to another or doing major re-writes of large articles.

- An easily accessible help facility and logical command structure which makes the program extremely efficient and easy to use.

- The ability to read and create documents in a variety of different file formats including program files, sequential files, and true ascii. This allows Pocket Writer to achieve a high degree of file compatibility with other Commodore word processors.

(Editors note: For a more complete appraisal of the original Pocket Writer 128 see our review in the January 1986 issue of Twin Cities 128. Also of interest may be our feature on choosing a word processor in the March 1986 issue of Twin Cities 128.)

Which brings us to the upgrade of Pocket Writer 128, Pocket Writer 2. First off, let me state that Pocket Writer 2 is a very significant upgrade to Pocket Writer 128 and is a "must have" if you already own Pocket Writer 128. The upgrade can be obtained by calling (Digital accepts some major credit cards) or writing Digital Solutions (30 Wertheim Court, Unit 2, Richmond Hill, Ontario, Canada L4B 1B9, 416-731-8775), only $19.95 (plus $3.00 for shipping & handling) for registered owners and includes a new program disk and a new greatly enhanced manual (more on the manual below). Unlike the policies of other software companies you do not have to send in your original program disk to receive the upgrade. For those individuals who do not already own Pocket Writer 128, you can acquire Pocket Writer 2, either by buying it directly from your local dealer (suggested retail is: $59.95) or by purchasing Pocket Writer 128 and sending for the upgrade following the procedure listed above.

As every good upgrade should, Pocket Writer 2 corrects perceived deficiencies in the original product. In my mind there is only one major flaw and three minor irritating characteristics in the original Pocket Writer 128.

The spell checking program included with the original Pocket Writer 128 is my "major flaw". To begin, the original spell checker is a separate 40 column C-64 program mode (the actual program itself is an 80 column C-128 program). This meant that you had to exit your excellent C-128 mode word processing program to spell check your document under all of the limitations C-64 mode i.e. slow speed, slow disk access, 40 column text etc. If that was not bad enough, this C-64 mode spell checker seemed to have a hidden gremlin which would on occasion trash your document or lock up the machine. But be of good cheer, becuase the programmers at Digital Solutions have heard the cries (screams) of the masses and have included a fully functional spell checker as PART of Pocket Writer 2. In fact, not only has the spell checker been improved simply through the incorporation of it into the main C-128 program (and thereby reaping the benefits of the 80 column screen, faster processing, and increased disk faculties), but it also now includes the ability to spell check a document from a dictionary file stored on a 1700/1750 ram expansion unit (more about the RAM expanders later in this review) which increases the speed of the spell checking process by 300%! Even without the RAM expansion the spell check process is very fast and smooth and has erraticated my only major criticism of the original Pocket Writer.

However, even though I am pleased as punch with the new spell checking facility, I would be remiss if I did not mention a few little drawbacks I have encountered while using the spell checker. First, the spell checker does not include a start-up dictionary. In other words, you must either start your own dictionary from scratch (which could take quite some time to build up a reasonable amount of words) or you must pay extra ($13.95) for 30,000 dictionary disk from Digital Solutions. Many other software producers have provided this as a part of the word processing package, although I will concede that those other programs are generally more expensive than Pocket Writer 2 to begin with. Second, although the spell checker offers the option of skipping over an unrecognized word, this option forces you to skip over all occurences of this word, and doesn't leave you to choice of skipping only that one occurence of

Pocket Writer II Continued

the word. Occasionally, this can be irritating. Finally, the only other small difficulty with this spell checker is that there is no way (that I know of) to edit your dictionary since it is in an encoded format. I happen to be human and make mistakes, and every once in a great while I add a mispelled word to my dictionary when I am in a hurry. If you are lucky enough to catch yourself adding a mispelled word to your dictionary using a RAM expander, the solution is to simple but somewhat inadequate, don't copy your Ram dictionary back to disk. Of course this means all of your added words are lost. Also, if you aren't using a Ram disk, you must quit the spell checker, and again you lose all words that you planned on adding to the dictionary disk. However, despite these small quirks, and all things considered this spell checker is quite good.

The other three minor criticisms I had with the original Pocket Writer 128 were: 1) There was no way to save your color, printer, and layout preferences as defaults. 2) Automatic word-wrap was only semi-automatic in insert mode (The original version would word-wrap in insert mode only after the inserting process was completed or the F6 key was pressed). And 3) There was no way to automatically join a document from disk into the middle of a document in memory, because the join command only was capable of appending text to the end of a document in memory. (This perhaps is the most minor point since it is simple to use the range edit commands to move the appended text from the end of a document to its desired location.) Again, to the credit of the Digital Solutions programmers all three of these areas have been dealt with in Pocket Writer 2.

Pocket Writer 2 includes the ability to save all of your colors, printer, and layout (margins, alignment etc.) as well as a few other preferences as defaults on the program disk. When the program is booted these preferences are read from a sequential file on the program disk called "configure" and set the programs default settings to your exact specifications. In addition to the preferences mentioned above, the user is also allowed to configure Pocket Writer to communicate with whatever storage hardware the user owns, including both dual as well 0s single disk drives, and a RAM expansion unit. Each storage device is assigned a number from 0 to 10 and the user may read or write data to/from a particular device easily by specifying which device number to access for a particular operation. This ability to access up to 10 individual storage devices gives Pocket Writer an incredible amount of flexibility. (As a side note, Pocket Writer 2 performed flawlessly with our new 1581 prototype.)

Pocket Writer's automatic word-wrap facility has also been improved and now flawlessly and automatically wraps words to the next line inside of the specified margins even in insert mode. In addition, for those creating charts and automatic word wrap can be toggled off completely.

Lastly, Pocket Writer 2's join command now allows you to join a document from disk into memory at any point within the document you are currently editing, giving Digital Solutions a clean sweep over this user's petty complaints.

Now if the additions to Pocket Writer 2 entailed only the above enchancements, one would be hard pressed to find a more powerful and easier to use product. But there is so much more to Pocket Writer 2 that it is hard to believe the code and program's internal variable storage still come in under 64K! Perhaps the single most important enhancement to Pocket Writer 2 over the original Pocket Writer 128 is its ability to utilize the Commodore 1700/1750 as a super fast Ram-disk for temporary storage. With the a 1750 attached to the back of your C-128 this means you have a 2045 block 1 megabyte/per second pseudo-disk drive available at anytime. This is particularly attractive for serious users who do a great deal of word processing on a daily basis, especially users with large editing tasks or who do a great deal of boiler-plating of text. As mentioned above the Ram disk can also be utilized for rapid spell checking. Pocket Writer 2 provides full support for the Ram disk including the ability to copy both specific files as well as an entire disk to the RAM disk including provisions for pattern matching and wild card characters.

Two other particularly attractive new features of Pocket Writer 2 are enhancements to the program's user interface. Pocket Writer now supports both the new 1351 mouse as well as the 1350 mouse and standard joysticks as input devices for easy to use "pull-down" menus, as well as range highlighting, and cursor movement. The addition of "pull-downs" is a nice feature for people who have a hard time remembering keyboard sequences, and I have found that for some large range operations the my 1351 mouse is particularly advantageous. Of course, for those with "mouse-phobia" (I used to be a charter member of the mouse-haters of America until I was converted to the mice-dom, so I understand) you can still utilize the full complement of keyboard sequences for your word processing needs. Another welcomed user-interface enhancement for advanced Pocket Writer users is the ability to not only shut-off the online help facility (a six-line window at the top of the screen) but to completely dump it from memory all together, adding another 5K to the already abundant document memory. Additionally, you can set your "configure" file to turn off and dump the help facility as a default.

And as if all of the above were not enough, Pocket Writer 2 now also supports the following: The ability to manipulate columns of text (we created the double column layout of this month's issue electronically with this feature); an

Pocket Writer II Continued

hanced delete option which allows the automatic deletetion of letters, words, sentences, and paragraphs in addition to local character, forward and range deletion; An automatic transposition command which allows the you to automatically swap the position of letters, words, sentences, and paragraphs; an automatic word counter; the ability to automatically switch a range from lower to upper case or vice-versa; the ability to perform all text enhancement features i.e. underline etc. on ranges, the ability to read GEOWRITE text scraps; the ability to search for, replace, or delete return markers; negative paragraph indentation; the ability to scratch files with the cursor from the directory; in memory mail merging from the alternate work area; and an awesome 50 line interlaced 50 line video display mode, plus a much improved manual which I rank as one of the best and most attractive pieces of software documation available produced so far for the C-128, complete with index, table of contents, quick reference chart, and two color printing to offset and highlight important points.

Am I happy with this product? You bet I am! This is the best word processor I have seen for the C-128 (and the competition is incredible) and our word processor of choice at Twin Cities 128.

Commodore 128 Programming Secrets *by John Kress*

Commodore 128 Programming Secrets is a good no-nonsense manual on Basic and assembly language programming by William M. Wiese, Jr. and published by Osborne/McGraw Hill. This book is designed for the serious C-128 programmer. Although the text is readable for beginners, it sports more than enough 'MEAT' for the advanced programmer.

The first chapter gives a detailed understanding of the differences between the C-64 side, and a true C-64. Along with a few good examples of what is possible with the C-128's C-64 side, there is a good tutorial on how to access the 80 column screen from the C-64 side, such as using the C-128's extended keyboard, and a few others. Although it is the chapter of least interest to most C-128 programmers, I feel that there's plenty of information there worth reading.

In the following chapter, C-128 Architecture and Memory Management, the workings of the Memory Management Unit (MMU) and Configuration Register (CR) are covered. This is probably the best text I've seen about these two items. Details covered here are: setting the bank configurations; manipulating the Ram, I/O, and Rom configurations; and allocating user definable Zero Page and Page One. Also covered in this chapter is the Initialization process; what happens when you turn on your machine, and the default vector settings when you reset your machine or hit Run/Stop-Restore. The entire chapter is annotated with programming examples and source code listings so that you can follow a program and get a fuller understanding of the text.

The third chapter covers memory usage, such as the zero page storage, low common Ram storage, and also the kernal variables, shadow registers and screen editor routines and their entry points. Much of this is similar to prior publications that cover this area, but a more in-depth explanation about many of the storage areas has been added. I'm sure that most will find this chapter very useful.

Next comes the Basic 7.0 Interpreter chapter. Included here are examples of how the C-128 stores variables, strings and programs. After reading this chapter one should know how to look at a memory dump of the C-128's Bank 1 storage and be able to interpret what is stored there. This section is followed by the most extensive Basic dictionary that I've ever seen for a microcomputer. Included in the dictionary are not only the basic keywords and an explanation of their usage, but their ROM locations. For the serious ML programmer, this is what you've been looking for. The major drawback of these ROM listings is that the parameters passed to these routines, such as 'DIRECTORY' with a pattern match, are not included. But if you are the type who can find use for these pointers to Rom, I'm sure that with the contents of this book, you'd be able to deduce the information to get your program working.

Chapter 5 covers the video system, including escape sequences. It includes information on the 8563 Video Display Controller (VDC) and the VDC registers, and a program to enter an alternate character set for the 80 column screen. Also included is an 80 column graphic demonstration program.

In Chapter 6, the C-128 kernal is covered in a bit more detail, and a good deal of information about the new kernal routines for the C-128 are covered.

Chapter 7, Disk and I/O, brings some very important things to light about the 1571 and 1541 and their bugs. Such things as what can happen to a double sided 1571 disk when used on a true 1541 (not 1571 in 1541 mode). Also covered is 'rst' mode and 'Auto-Booting'. This is followed by the CP/M section of the book, which includes the system configuration, memory maps and a short tutorial on CP/M itself. This is one of my weakest points and also appears to be one of the shortest chapters in the book, so I'd think that a better source of CP/M material would be in

Commodore 128 Programming Secrets Continued

order for those who want to get the most on this subject. [Editors' Note: the information included in this chapter bears a striking similarity to material previously published in TPUG and TC-128 by Miklos Garamszeghy.]

At the close of the book are the appendices, covering the alternate character set's memory dump, C-128 pinouts, the customary conversion tables, and a bit about displaying the 80 column screen on a monochrome monitor.

In summary, I'd have to say that this is one of the best third party publications that I've had the opportunity of viewing, both for the beginning Basic programmer and the experienced 65XX/8502 programmer. There has been a great deal of research that has gone into producing this book, and that is obvious from the moment you start to read the first page. Of all the examples listed in the book that I've tried, none have 'CRASHED', which is an accomplishment over and above many prior publications. If you have $16, and are looking for a good source of in-depth information for your C-128, I'd strongly suggest that you take a gander at this one. Who knows, you might be able to develop the next block-buster program for the C-128 with what you learn from reading this one.

Flic Your VIC *by James O'Hare*

How would you like to maintain 5 or 6 high resolution screens in memory and switch instantly from one to the other? Or perhaps display a HR picture while your program draws another picture off screen and performs some other set-up routines? The C128's bank switching capability makes acrobatics like this easy for the novice programmer. All that's required is a little knowledge of the system architecture and a few simple commands.

The C128 contains two 64K Ram memory banks. Ram 0 is used for Basic program code and Ram 1 for variable storage. Each Ram bank is configured into four 16K video banks (or VBs for short), numbered 0, 1, 2 and 3. The VIC II chip, which supports the 40 column screen, looks for its character and graphics memory in one of these video banks. On power up the VIC defaults to Ram 0, VB 0 but it can be pointed to any of the eight available video banks.

The Ram Configuration Register (RCR), located at 54534, controls which Ram bank the VIC is "looking at". When bit 6 is low (0) the VIC is pointed to Ram 0. When bit 6 is set (1) the VIC looks at Ram 1. Bits 0 and 1 in register 0 of the Complex Interface Adapter #2 (CIA#2), located at 56576, tell the VIC which of the four video banks within the selected Ram bank to address. Both of these registers are accessible from Basic and the proper commands will be explained below.

A brief explanation of high resolution memory requirements will complete our background information. The standard HR screen requires 9K of memory. The first kilobyte contains the background and foreground colors for each of the one thousand character positions on the HR screen. This is called the color matrix. The remaining 8K is the bit map, where each bit corresponds to a pixel on the screen. The GRAPHIC 1 command reserves the memory in Ram 0, VB 0 from location 7168 to 16384 for the color matrix and bit map. As the name suggests, GRAPHIC 1 also causes the VIC to display the high resolution screen rather than the low res text screen.

Armed with these facts you can start to use multiple HR screens in your programs. The simplest approach is to create the screens beforehand and load them from disk. Doodle pictures can be used without any conversion. Alternately you can draw pictures using Basic 7.0 commands and save them as follows: BSAVE "filename", B0, P7168 TO P16384

Next, under program control, issue the command GRAPHIC 1,1 and load one picture into each video bank that you want to use. (This can also be done in direct mode. In that case first return to the text screen with GRAPHIC 0 or the split screen with GRAPHIC 2,1.) The proper loading syntax is: BLOAD "filename", Bx, Py where x is the Ram bank (0 or 1) and y=7168 + (16384 * video bank number). The formula ensures that each screen is loaded into the same relative position in the respective video bank. For example, to load a picture into Ram 1, VB 2 enter: BLOAD "filename", B1, P39936

It's best to avoid video bank 3 when starting out. The C128 needs some memory at the top of each Ram bank for its own use. Overwriting this with bit map data will cause a crash. Remember too that the pictures residing in memory are ordinarily used for program code and/or variables. If your program is less than 7K long and doesn't contain many large arrays you can probably use video banks 0, 1 and 2 in both Ram 0 and Ram 1. This allows a total of six screens. If the program is longer, use only the higher video banks.

Next, we must tell the VIC where your pictures are located. As long as you stay in Ram 0, the default memory bank, the RCR need not be changed. To switch to Ram 1 use the following command:

```
BANK 15:POKE 54534, XOR(PEEK(54534),64)
```

Flic Your VIC Continued

Because of the XOR, sometimes called "exclusive OR", this command works as toggle. Each time it is executed the VIC will to switch to the other Ram bank.

Select the appropriate video bank with this command: POKE 56576, (PEEK(56576) AND 252) OR z

The value of z can be determined from the following table:

Video Bank	Address Range	Z
0	0 - 16383	3
1	16384 - 32767	2
2	32768 - 49151	1
3	49152 - 65535	0

Each time a value in the RCR and/or CIA#2 is changed the VIC will display the bit map screen resident in that particular bank. Remember that the Ram and VB switches are instantaneous. Depending on the application you may wish halt program execution until a key is pressed. If you use Ram 1 you should also toggle back to Ram 0 after the pictures are displayed so that the ordinary text screen is available.

Page Flipping With 64K VDC Ram *by John Kress*

Last month we learned how to initialize the 64K ram chips for use with the 8563 VDC. Now we'll learn more about programming the VDC chip, using the 37 internal registers. Those of you how haven't upgraded the ram, stay tuned as this month we'll learn how to program a second text screen into the VDC memory, and there is room for that in the 16K memory chips.

First off, the access to the VDC registers is attained through two memory locations in the 128's Input/Output Block. Since these locations are commonly used by the 128's editor to control what is on your 80 column screen, we can use the built-in Rom routines that read and write to the VDC to do our programming with. The only stipulation is that these routines must be used from Bank 0 or Bank 15, with the I/O switched in. That is not much of a problem, so we'll use them. The routine to write to a register is at $CDCC, using the .X register for the register you'd select to write to and the .A register holds the value to store in that register. The read routine is at $CDDA, in the same format, .X = register selected and .A will return the value in the chosen register. So if you would want to write to register 12, a value of 16, you would use the following:

```
    LDX #$0C   ; Register No. 12
    LDA #$10   ; Value to Store = 16
JSR $CDCC ; Rom Routine to Write
```

The Value of 16 would now be stored in register #12, which is the Hi Byte register for the start of character memory for the VDC.

Now that we know how to work with the registers, we'll build up a little ML code to initialize the 64K rams and implement a second text screen within the VDC's ram memory. The coding is written so that any C-128 can have a second text screen in memory and switch between the two screens. (If you have a 128 with the 64K rams installed you can change some of the program and set pointers to a screen in any area of the VDC's ram.) First we'll write a routine to initialize the 64K rams, if you wish to use them, then the code is there. Enter the Monitor and start assembling the program at $0C00, which is in the RS-232 buffer area and is a rather secure place to write a short program.

```
0C00  LDX #$1C     ;SELECT REG. 28
0C02  JSR $CDDA    ;READ SUBROUTINE, REG. VAL. IN .A
0C05  ORA #$10     ;SET BIT 4, 64K CHIPS INSTALLED
0C07  JSR $CDCC    ;WRITE TO REG. 28
0C0A  JSR $CE0C    ;COPY ROM CHAR. INFO INTO VDC RAM
0C0D  RTS          ;ROUTINE DONE.
```

The JSR $CE0C is needed to reformat the character information after setting up the 64K ram configuration. If this is not done the character information would be all scrambled up due to the way the 64K chips use the pins for addressing. That is a technical matter that we don't need to get involved with, but is nice to know.

Now we'll build the routines to change the pointers to the different screen areas in ram. But first a bit of information about character ram and attribute ram. The VDC uses two sets of pointers to define what areas of its ram contain information for characters and attributes. Character ram is much the same as the character info used by

Page Flipping With 64K VDC Ram Continued

the VIC chip, and the attributes are much the same as the color ram used by the VIC chip. The attribute ram, though, holds a bit more information than the VIC's color ram. The lower nibble, or lesser for bits, contain the RGBI color assignment for that 8*8 pixel area. The upper nibble, or higher four bits contain the information for alternate (lowercase), reverse, underline, and flashing attributes. This is the reason that the 80 column screen can do much more with the characters displayed on its screen.

The beginning of screen memory is stored in registers 12 and 13, so by changing the value in these two registers, you can display the character information stored in a different area of ram. The attribute memory pointers are in registers 20 and 21, and by reprogramming these we can move the attribute pointer to another area of VDC ram. Now we'll build two routines to change these pointers, one routine to switch to a screen area stored at $1000 and one to switch back to the default area at $0000. Enter these with the Monitor at the next open spot in the RS-232 buffer, $0C0E.

```
0C0E  LDX #$0C     ;SELECT DISPLAY HI BYTE REG.
0C10  LDA #$10     ;SET HIGH BYTE OF $1000
0C12  JSR $CDCC    ;WRITE THE VALUE
0C15  LDX #$0D     ;SELECT REG. 13
0C17  LDA #$00     ;LOW BYTE OF $1000
0C19  JSR $CDCC    ;SCREEN MEMORY NOW $1000
0C1C  LDX #$14     ;SELECT ATTRIBUTE HI BYTE REG.
0C1E  LDA #$18     ;SET HI BYTE OF $1800
0C20  JSR $CDCC    ;WRITE
0C23  LDX #$15     ;SELECT ATTRIBUTE LO BYTE REG.
0C25  LDA #$00     ;LO BYTE OF $1800
0C27  JSR $CDCC    ;WRITE
0C2A  RTS          ;SCREEN NOW SET AT $1000

0C2B  LDX #$0C     ;DISPLAY HI BYTE REG.
0C2D  LDA #$00     ;HI BYTE OF DEFAULT SCREEN
0C2F  JSR $CDCC    ;WRITE
0C32  INX          ;.X REG NOW = 13
0C33  JSR $CDCC    ;.A IS STILL #$00, SO WRITE AGAIN
0C36  LDX #$14     ;RESAT DEFAULT ATTRIBUTES
0C38  LDA #$08     ;TO $0800.
0C3A  JSR $CDCC    ;WRITE
0C3D  INX          ;.X NOW $15
0C3E  LDA #$00
0C40  JSR $CDCC    ;WRITE
0C43  RTS          ;SCREEN NOW SET AT $0000
```

Now the VDC chip can point to two different screens just by calling these two routines, but there isn't any thing of interest at the second screen location. So now we need to put some meaningful data into the second screen. Unfortunately, when you set the pointers to the second screen, the ROM screen controller doesn't print to that screen. In order to do that we'll have to tell the VDC controller the new screen's location. In low ram, there are two pointers that hold the high byte of the screen for 80 column mode. These are $0A2E, 2606 decimal, for the character pointer and $0A2F, decimal 2607, for the attribute pointer. By changing these pointers you can print to a screen anywhere in VDC ram.

Now we'll make two routines to set the pointers for printing to screens beginning at $1000 and $0000. There's a reason that these are separate routines from the above, and that will be explained later.

```
0C44  LDA #$10     ;SET HIGH BYTE OF $1000
0C46  LDX #$18     ;SET HIGH BYTE OF $1800
0C48  STA $0A2E    ;CHARACTER POINTER
0C4B  STX $0A2F    ;ATTRIBUTE POINTER
0C4E  RTS          ;PRINT TO SCREEN 2

0C4F  LDA #$00     ;SET HIGH BYTE OF $0000
0C51  LDX #$08     ;SET HIGH BYTE OF $0800
0C53  STA $0A2E    ;CHARACTER POINTER
0C56  STX $0A2F    ;ATTRIBUTE POINTER
0C59  RTS          ;PRINT TO SCREEN 1
```

Now save the ML code by entering from the monitor the following line: S "2SCREENS.ML" 08 C00 C5A

Page Flipping With 64K VDC Ram Continued

Now we have the basis for developing a program which can use an alternate text screen. There are any number of uses for a second screen, the foremost of which is within a program that uses a menu of commands. With this ability, you have at the touch of a single key, a complete menu screen available, without the delay involved in clearing and reprinting the menu. This is one thing that has bothered me with many of the terminal programs that I've used, when you use the menu command the entire screen is lost and you can't get it back.

To utilize the above code try entering the following Basic program. You'll notice that the one screen can be displayed while the other one is being printed to. This can be useful in tutorial type programs, where you would be prompted to hit a key to continue, after reading the text on the screen. While the key is pressed, flip the screen display pointers in the VDC registers and the next screen is already printed. This is the reason for those separate routines at the end of the ML code.

Now to use these routines, enter the following Basic program:

```
 10 I6 = DEC("C00"): REM I6 - INITIALIZE 64K RAMS
 20 S1 = DEC("C2B"): REM S1 - SCREEN 1
 30 S2 = DEC("C0E"): REM S2 - SCREEN 2
 40 P1 = DEC("C4F"): REM P1 - PRINT TO SCREEN 1
 50 P2 = DEC("C44"): REM P2 - PRINT TO SCREEN 2
 60 SCNCLR: PRINT "PRINTING SECOND SCREEN..."
 70 SYS P2: SYS 32800,123,45,6
 80 SYS P1: PRINT: PRINT "HIT A KEY."
 90 GETKEY A$: SYS S2: SYS P2
100 SCNCLR: CHAR 1,1,16,"HIT A KEY TO SEE SCREEN1"
110 SYS P1: LIST
120 GETKEY A$: SYS S1
```

CP/M - Using Submit Files *by Randy Margolis*

If you work with CP/M much you probably find yourself typing the same commands over and over again. Sometimes these command lines can get somewhat complex, and are always rather arcane. SUBMIT.COM comes with your CP/M system disk, and can automate just about anything you can think of. What SUBMIT does is read from a simple text file with the file type .SUB a series of command lines as if they were typed in from the keyboard. It creates a temporary file (with the file type .$$$) which matches the .SUB file and, one by one, processes the commands and strips them from the temporary file. When the last command is dealt with the temporary file is erased, which is why, unless there is some kind of crash in the middle, you never see the .$$$ files in your disk directory. The only thing you need, other than your imagination, is a means of creating the .SUB file. You may use ED.COM, that awful line editor which comes with the system, but I would suggest that you look far and wide for a full screen text editor, many of which are in the public domain. Your user group should have VDO.COM or VDE.COM in its library, both of which are superb for this purpose.

When you first boot CP/M, the system looks for a file called PROFILE.SUB on the boot disk and, if found, starts processing the commands contained therein. Keep in mind that SUBMIT.COM must also reside on the boot disk for the commands to be interpreted. Also, do not cover the write-protect notch on the boot disk since, as previously stated, SUBMIT must write a temporary file to the disk. It is a good idea to leave 10k or so free space on this disk, to allow for large batches of commands to be processed. PROFILE.SUB may be used for parameterizing your CP/M system, or running a specific program. Knowing this, you may set up individualized boot disks for various applications disks, and have the program run automatically.

Here is an example of a PROFILE.SUB which I use:

```
SETDEF *,A: [ORDER=(COM,SUB),TEMPORARY=A:]
CONF PRT1=ASCII8,40COL=OFF,VOL=0
DATE SET
```

SETDEF is a program which also comes with your system that does very nice things for you, especially if you have more than one disk drive, or a Ram expansion unit. This first line of my PROFILE.SUB file uses this utility to first set a drive search chain. The '*,A:' portion says to the system, "First look for a program on the currently logged drive and then, if you don't find it there, look on the A: drive". So, if your screen prompt is A> and you call a program that resides on the A: drive, CP/M will find and execute it. The next part (ORDER=(COM,SUB)) tells the system to first check for a .COM file with the name you specify, and then, if unsuccessful, execute a .SUB file with the same name. Thereafter instead of having to type 'SUBMIT MYFILE' you

CP/M - Using SUBMIT Files Continued

just have to enter MYFILE and, provided SUBMIT.COM is available, away you go. Finally, the last section of the line tells CP/M to create the temporary file required by several different utilities (the .$$$ file) on a specified drive. This is especially useful if you have a 1700 or 1750 RAM expansion attached. If you do, and specify TEMPORARY=M:, then you don't have to worry about there being enough room for temporary files on a floppy disk.

The next line invokes a file called CONF which further configures your system. This file should be available in almost any user group library, and sets some controls that will optimize your session for you. The first part tells CP/M what type of printer you have (ASCII or CBM), and what secondary address to send to your interface (if one is needed). My Xetec Super Graphix uses a secondary address of 8 to lock it into Upper/Lower case mode with no auto line feeds. The next portion turns off the 40 column screen making the system run about 8% faster, and the final part turns off the speaker, which also quickens the clock speed.

The last line of the PROFILE.SUB allows me to set the system date. This is handy if you have turned on the Date and Time encoding option with the INITDIR program.

Here is another example of SUBMIT freeing you from looking things up in a reference book. Sometimes, I want to backup some data files on another disk into a different user number, thus putting them out of the way, but safely copied:

```
PIP A:[G9]=B:*.DAT[A]
```

This copies all changed files of type .DAT from disk B: to user area 9 on disk A:. I found this was a pain to remember and was always looking up the proper syntax, so I put the one line into a file called ARCHIV.SUB. Therefore, all I have to do is type ARCHIV and the backup will be performed.

You can also put program input into a SUBMIT file, using the '<' character. This is a nifty example for a Ram expansion user:

```
PIP
<M:=A:PIP.COM
<M:=A:ERASE.COM
<M:=A:PCFILE.COM
<M:=A:*.DAT
<M:=A:*.NDX
<
M:
PCFILE
PIP
<A:=M:*.DAT
<A:=M:*.NDX
<
ERASE *.DAT[C]
ERASE *.NDX[C]
```

This example first copies PIP, ERASE, and the public domain database program PCFILE into your Ram disk. Then is copies the data files used by PCFILE (.DAT and .NDX) from the A: disk. The final '<' is just a carriage return to end PIP. Next the PCFILE program is run from the Ram disk. Note that your SUBMIT file is not finished, but is suspended until you quit from the application program called. If the application allows viewing of a disk directory you will see a temporary file SYSIN56.$$$ in the list of files, which are the remaining SUBMIT commands. When you quit from PCFILE, PIP is re-invoked and copies the altered data files back to the A: disk. These data files are then erased from the Ram disk using the 'C' option of ERASE which requests confirmation from the user for each such erasure.

You can also put a message on the screen in the middle of a SUBMIT operation using a feature of PIP which copies a text file to the screen:

```
PIP
<CON:=A:MSG.TXT
DIR B:
```

In this case the file MSG.TXT warns the user to put a disk in drive B: and to press return. Note the absence of the lone '<'. This forces the user to press return to end the PIP program and continue with the SUBMIT file's commands.

The Assembly Line *by John Kress*

This installment of the Assembly Line deals with stack manipulation. That is, using the 65XX/8502's stack for programming desired results. The opcodes we will be using are PHA and PLA. PHA means push the contents of the .A register onto the stack. PLA means to pull one byte off of the stack and place it into the .A register.

Playing around with the stack can have some disastrous results: if you happen to leave out an important PLA instruction, your program can end up jumping into never-never land, or crash. The reason for this is the function of the stack and also the PC or program counter. When the 8502 encounters a JSR instruction, the current address of that instruction is pushed onto the stack. Then when the CPU comes upon the RTS instruction, it pulls off the stored address, adjusts the PC and continues execution of the program at the memory location following the stored value. This is very much like the basic GOSUB and RETURN commands, when a RETURN is encountered the program returns to the instructions following the GOSUB command. So, for this reason it is very important to return the stack to the original condition prior to using an RTS instruction.

Saving a register on the stack is a very quick operation, using a one byte instruction code, as opposed to a multiple byte instruction. Therefore the execution of the PHA instruction is fast in comparison to a STA $XXXX. In a case where you enter a subroutine that may cause the loss of vital information in the registers, you can PHA, then transfer the contents of the .X register to the .A register, TXA and PHA, followed by the .Y register, TYA and PHA. When done with the subroutine you'd need to PLA all of the registers and restore them to their proper register before an RTS is issued. If not, the program will jump to the address of the values stored on the stack plus one.

This brings us to the point where we'll take this information about the way the stack stores things and develop some new techniques for using the stack. Take, for instance, a portion of a program that would be decoding a keypress, and checking if it's one of the desired keys from a table of keys. You can very easily check against the table by comparing the address of the table with an offset in the .X register, where .X would be the length of the table. Then JSR to the correct subroutine from a table of subroutines, when the correct key has been found. If the correct key is not found, then you'd just RTS when the .X register is decremented to 0.

Or in a sequence where two keys, such as the C= or CONTROL, followed by another key, are checked the following is a good alternative. First the check for the first key of the sequence would be tested, then a JSR to the testing the second key. Using the .X register to check your table of valid keys, use the .X register to LDA the .A register with the high byte of the subroutine that corresponds to that key sequence then PHA on the stack. Next LDA with the low byte of the subroutine minus one, then execute an RTS instruction. At this point the processor will pull off the address you've pushed on the stack and start execution of the routine at that address and then upon the RTS instruction, will return to the program where the JSR to the key decoding routine was called from. If no valid key is found, when the .X register is zeroed, just return from the sub-routine. Either way your program returns to the same point.

Try the example program below, which is a short example of this technique. The program resides in the RS-232 buffer area, where it is not in the way of Basic programs, and can easily be modified to suit your needs.

```
00C00  20 E4 FF JSR $FFE4   ;GETIN routine will return ASCII value of key pressed in .A
00C03  F0 FB    BEQ $0C00   ;No key? try again.
00C05  A2 03    LDX #$03    ;Load .X with key table length
00C07  DD 1C 0C CMP $0C1C,x ;and compare the valid table of keys
00C0A  F0 04    BEQ $0C10   ;Correct key found? load address from address table
00C0C  CA       DEX         ;if not decrement .X and
00C0D  10 F8    BPL $0C07   ;keep checking table.
00C0F  60       RTS         ;Return if value is not in table.
```

This is the main portion of the program that would load the address bytes into the .A register and push them onto the stack. But the .X register is holding a byte value that needs to be adjusted to load a 16 bit address. Therefore the .X register will be transferred to the .A reg. and shifted, or doubled, with the ASL operation. Then the .X register will be returned and the program continues.

```
00C10  8A       TXA         ;Move .X to .A reg
00C11  0A       ASL         ;Shift left one bit
00C12  AA       TAX         ;and restore into the .X reg.
00C13  BD 20 0C LDA $0C20,x ;Load .A with the hi byte
00C16  48       PHA         ;and store on stack
00C17  BD 21 0C LDA $0C21,x ;Load .A with low byte
00C1A  48       PHA         ;and store on stack.
00C1B  60       RTS         ; RTS in effect returns to the routine.
```

The Assembly Line Continued

Now enter the following byte values into the program with the 'M'emory modify command of the monitor:

```
>00C1C 31 32 33 34 3F FF AF FF A0 7D CB 3E 00 00 00 00
```

Activate the program by the basic command 'SYS 3072'. If you've entered it correctly, the only key entries that will do anything are the numbers 1 through 4:

1 will send the C-128 through the system reset routine, complete with the opening Commodore message.
2 will send the C-128 into the MONITOR.
3 will perform a DIRECTORY of the default drive.
4 will perform the routine sequence ESC 'R'.

Sparrow's Slick Tips *by Sparrow James*

00110011: CP/M file conversion with Bobsterm 128: On occasion it is desirable to convert sequential text files from CP/M formatted disks to normal 1571 format and vice-versa. One way to accomplish this task is through the use of the fantastic commercial telecommunications program: Bobsterm 128. Bobsterm 128 has the ability to read from and to CP/M formatted disks from C-128 mode. With these capabilities and Bobsterm's 60k buffer and built-in text editor text file conversion is a snap. Here is what you do: First load Bobsterm 128 into your C-128 then:

If you want to convert and save a conventional sequential file to a double-sided CP/M formatted disk:

1. Choose F to fill the buffer from the main menu.
2. Next you will be asked whether you want to fill the buffer from keyboard, disk, or from a specific disk byte. Choose press D for disk.
3. Now you will be asked to supply the name of the file you wish to read into Bobsterm's buffer.
4. Once you have supplied a valid filename you will be asked for the type of file you wish to read into the buffer, select 1 for sequential. The file will then begin being read.
5. Once your file has been read into the Bobsterm buffer, hit the run/stop key twice and you will be returned to the main menu, from there choose D for dump buffer.
6. Now from the dump buffer menu choose D again for disk, then you will again prompted for a filename, this time for the file that you will be creating on your CP/M formatted disk. Finally, once you have supplied a valid filename you will asked for a filetype, this time choose 8 for CP/M. Now you will be prompted to insert a double-sided CP/M disk and press return. Once you have placed a CP/M formatted disk into your 1571, press return and your file will be saved to the CP/M disk.

To convert and save a CP/M text file to a conventionally C-128 formatted disk:

1. Choose F to fill the buffer from the main menu.
2. Next you will be asked whether you want to fill the buffer from keyboard, disk, or from a specific disk byte. Choose press D for disk.
3. Now you will be asked to supply the name of the file you wish to read into Bobsterm's buffer.
4. Once you have supplied a valid filename you will be asked for the type of file you wish to read into the buffer, select 8 for CP/M. You then will be instructed to insert a double-sided CP/M formatted disk and press return. Once you insert a CP/M formatted disk and press return, the file will then begin being read.
5. Once your file has been read into the Bobsterm buffer, hit the run/stop key twice and you will be returned to the main menu, from there choose D for dump buffer. Now choose E from the dump buffer options menu for edit buffer.
6. You are now in the Bobsterm editor, press the commodore logo key and the P key together. This will convert your text from normal ASCII to PETASCII. The screen will blank for a second while the conversion is taking place. When the conversion is completed, press the run/stop key to return to the main menu.
7. From the main menu choose D for dump buffer. Now from the dump buffer menu choose D again for disk, then you will again prompted for a filename, this time for the file that you will be creating on your 1571 formatted disk. Finally, once you have supplied a valid filename you will asked for a filetype, before you choose, make sure you have your 1571 formatted disk in the drive, and then choose 1 for normal sequential and your file will be saved onto your 1571 disk.

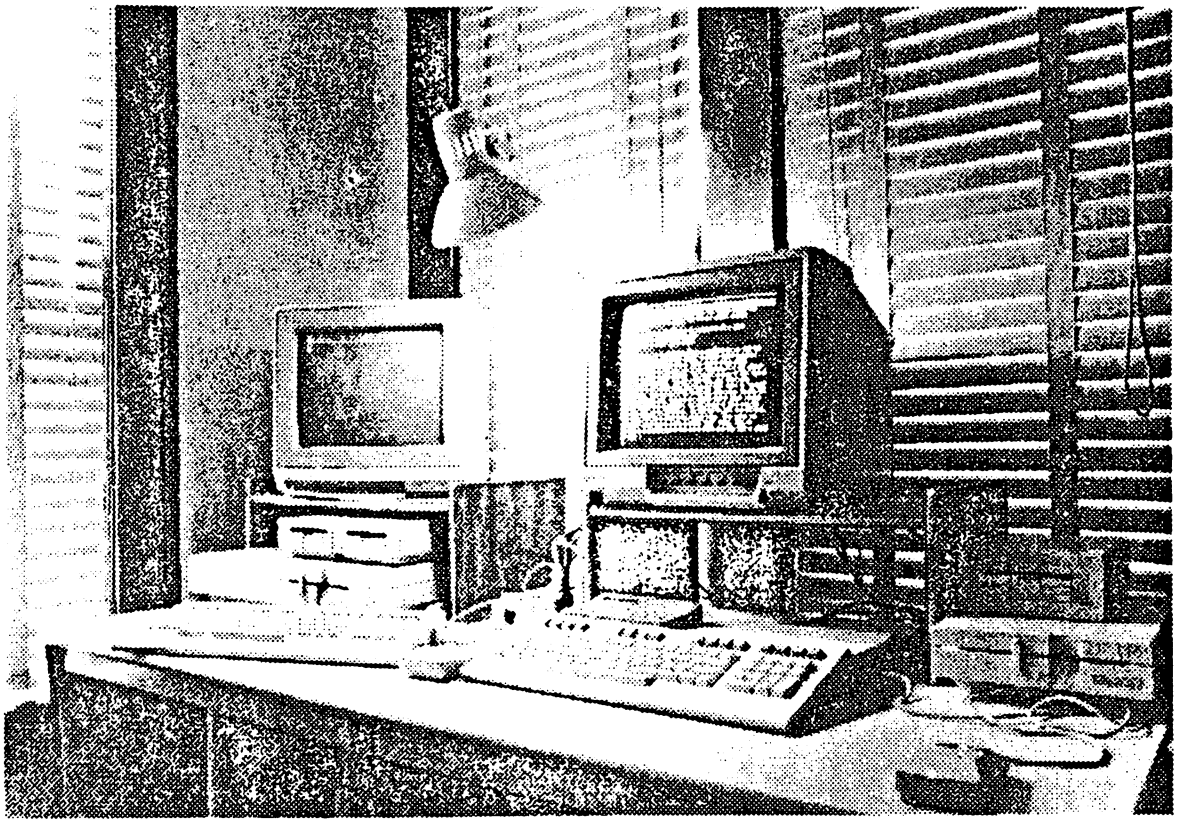
00110100: Pocket Writer 2 as a disk file management tool: In addition to being a fine word processor, Pocket Writer 2 as the ability to be a super powerful disk file management tool. Pocket Writer 2 has the ability to read disk directories into memory as normal document files using by the shift d (capital D) command after pressing the commodore logo key. You can even use pattern matching options and wild cards to select directories of specific files. The number of disk directories you can read into memory is limited only by your document space, and ample 64k. Once you have read your directories into memory you can use the powerful editing, sorting, and column features of Pocket Writer 2 to organize your disk catalog and add comments as you wish. And because you are using a word processor instead of

Sparrow's Slick Tips Continued

the conventional disk filing program it is a easy to print out your directories in various pitch sizes and typestyles. In addition, Pocket Writer 2's new enhanced scratch from directory option and RAM disk capability make it easy to clean-up and re-organize your disks.

00110101: Steal a quick Hex conversion from your MONITOR

If you want to convert a number to hex in the range of 1-255 and have it print to the screen at the current cursor position you can use: SYS 47298, X. This is a portion of the MONITOR rom that reads the current values of the various registers and displays them under the register titles when you would type the 'R' command.



Rumor/Opinion/Mayhem *by Loren Lovhaug*

Forgive me for the self-indulgence, but I am going to take this opportunity to pat Twin Cities 128 on the back with what I call my State of Twin Cities 128 Address. We are one year old, and I am quite proud of what we have accomplished. Many people in the Commodore computing world told us that a "C-128 specific" publication just would not have enough appeal to gain enough readership to break even let alone be profitable. In addition they were quick to point out that without the benefit of large corporate sponsorship or big-time national advertising we were destined for failure. But they were wrong!

Don't misunderstand me: Twin Cities 128 is not taking Wall Street by storm. Of all the Commodore magazines you can possibly think of, I can guarantee you we are by far the poorest and least profitable, but we are paying the bills, and in the tumultuous microcomputer world that is something. Besides, I guess there is a lot to be said for doing something that you enjoy. We have come a long way in just one year of publication. Our progress is especially impressive when you consider we only have one full-time staffer (myself), and only two part-time staffers: Avonelle Lovhaug, my incredibly overworked wife, and Bill Nicholson, a man who has made countless unselfish contributions without which this publication would have gone the way of the 8 inch floppy disk long ago. I think we can be proud of the attention and cooperation we are starting to receive from some of the major players in the Commodore computing world, including Commodore themselves. It is very difficult in this industry to gain respect without large amounts of money. Fortunately, there are individuals in the Commodore computing world that respect our goal of providing C-128 owners the very best C-128 information possible.

I am also proud of the fact that we still produce our magazine entirely with our C-128's. It has always bothered me to a certain extent that all of the "glossy magazines" (except Info) preach the power of inexpensive Commodore computers and yet in practice don't make use of them in production of their product. This is especially important to me because I believe that every issue of Twin Cities 128 is a tribute to what can be achieved by just a few individuals who have learned how to utilize fairly inexpensive tools to their maximum efficiency and effectiveness. I believe that relying only upon the C-128 in our production progress adds unmatched credibility to our opinions and reviews, because you know we have become experts at using our C-128s in a professional setting (i.e. we use this stuff every day...not just when its "candy-coated" review time. I challenge any of the so-called "major Commodore magazines" to "climb into the trenches" with the end-users who ultimately line their pockets. Until they do, I believe every opinion they present is suspect, every review they print is tainted. In my book there is a major difference between reviewing a product and "living" with it.

But while it has been a year I can be proud of there there have been disappointments or more aptly challenges that we have yet to overcome. Perhaps the largest challenge we need to confront is our production schedule. Lately it has become increasing difficult to get Twin Cities 128 written, edited, laid out, printed, and in your hands in just thirty days. I could use excuses to rationalize some of our production delays by spewing out claims that we are understaffed, underfunded, and are occasionally lied to etc., but I won't, because while all of these excuses are quite true, they are my problem and not yours. I just hope that our claim to the "best C-128 coverage anywhere" can help you tolerate some of the delays that are associated with our current operation. I am hopeful over the next few months we can do a better job adhering to our schedule. But the fact remains that although I am sincerely striving to streamline our operation in every way possible there are certain inherent aspects of our operation which naturally lend themselves to delays and may never be totally eliminated. You see, our commitment to presenting you with fresh, up-to-date, and innovative C-128 coverage, instead of the stale "written six months ago" stuff you read elsewhere lends itself to the phrase "stop the presses" instead of the phrase, "publish any old junk, just make sure it gets out in time so our advertisers are happy".

Lastly, I want to improve the overall look and professionalism of our publication. In the near future we will be adding the services of a laser-printer which I believe will go a long way towards this end. But I want to enhance our image in as many ways that are possible. You may have noticed over the past few months we have been working to include more direct interviews with the newsmakers in the Commodore 128 world and get direct quotes from named sources instead of the unsubstantiated rumors other magazines attempt to pass off as fact. By talking directly with the principals involved in C-128 development and by openly naming our sources for news and rumors we believe we ultimately publish better and more accurate information. Another area we are going to try to do a better with is the actual proofing of our copy for spelling, grammatical and typographical errors. Each month it angers me to no end how many of these gremlins end up slipping by into print, and I assure you we are on a crusade to wipe out these monsters!

In conclusion, our goal is to make every new issue of Twin Cities 128, better than the last. I believe we have made progress with every issue we have published since our inception in January of 1986. And as always I will be candid with you about our successes and our failures.

Rumor/Opinion/Mayhem Continued

C-128 world buzzing activity:

If I were forced to choreograph the last thirty days of activity in the Commodore 128 world, I guess I would choose as theme music last year's pop hit "Alive and Kicking" by Simple Minds because that phrase more than any other I can think of represents the following fantastic news for C-128 owners:

Commodore renews commitment to C-128 line CES

We have been telling you the rumors of the C-128's death were utter hogwash for months now, but if you were skeptical, all you had to do was visit Commodore's booth at the Consumer Electronics Show in Las Vegas, Nevada. With all the rumors and hype surrounding the possible introduction of new Amiga models that were supposed to "seal the fate of the 128", I think many a rumor-monger was put to shame when they entered the extravagant two-story Commodore booth not to gaze upon new Amiga products, but to lay their eyes on new C-128 products! That's right, C-128's were everywhere...well more precisely, the new C-128D (yep that sexy European version of the C-128 we featured last month) was everywhere, flanked by 1581 disk drives (in production plastic), the new 1351 analog "true mouse" controller. What do these introductions mean? Well, first, I think these new additions to the C-128 product line ought to finally lay to rest the stupid speculation about Commodore abandoning C-128 owners, at least for the rest of this year. And secondly, I think this is a clear indication that it is the 8 bit micros that are paying the bills at West Chester, and probably will be for sometime to come.

Berkeley Softworks revises GEOS 128 philosophy

For five months now we have been telling you that we hoped to have a preliminary copy of GEOS 128 in our hot little hands by the time our next issue went to press, and each time we came up empty. Throughout the month of December marketing assistant Rob Siegel promised me that we would have a pre-production beta-test copy of GEOS 128 before CES in early January to tell you about. Well gang, that was just not meant to be. How could this be, you might ask? Did Berkeley Softworks go back on their word to our fearless managing editor? No, certainly not. Is our brave managing editor losing his ability to cry, beg, plead, cajole, bribe, and otherwise make a nuisance of himself until the big companies cave in and include lowly Twin Cities 128 on their pre-release mailing lists. Let's hope not! So why isn't this issue featuring a comprehensive review of GEOS 128? Well, before you start composing those nasty-grams to Berkeley Softworks, let me do my best Paul Harvey imitation and tell you: "The Rest of The Story".

Those of you who are faithful readers of these pages may recall that it was Berkeley's intention to release GEOS 128 for use on the C-128's 40 column screen, much to the chargin of many C-128 owners who have grown very fond of 640 horizontal pixels. The reasoning behind this decision was simple: by implementing the C-128 version of GEOS on the 40 column screen it would be compatible with all of the existing application programs which run under the C-64 version of GEOS. In addition, the conventional wisdom was that various aspects of the C-128's 80 column video controller were not as condusive to GEOS as the VIC chip with its hardware sprites and other familiar vices. But the good news is, that kind of thinking is PAST HISTORY!

It seems one week before CES, there was a rather drastic change of heart at Berkeley Softworks. The result of which and a dozen all night programming sessions, was a very rudimentary, but albeit impressive CES demo of an 80 column C-128 version of GEOS. Upon contacting Rob Siegel at Berkeley after this CES bombshell we learned that plans are now full speed ahead for a fully operational 80 column, C-128 version of GEOS, including 80 column versions of GeoPaint and GeoWrite to be released sometime this spring.

While this is most certainly good news, we wonder what prompted Berkeley to pull a 180 turn-about on GEOS 128. While I doubt we will ever know for sure, I can't help thinking that Berkeley's decision might have something to do with the developments of the Floridian wonders: Louis Wallace and David Darus. In conversations with engineers Fred Bowen and Dave Haynie of Commodore, we have been repeatedly told of various 80 column miracles (including sprite-like animation) performed by the mysterious WALRUS product dubbed Basic 8.0. While we have yet to actually see this "star-child" we have no reason to believe from the descriptions we have heard that it is anything short of spectacular. And perhaps, just perhaps, Berkeley caught wind of this project and decided that a 40 column version of GEOS 128 would appear too pale in the eyes of C-128 owners once/if the rumored wonders of BASIC 8.0 ever are brought to fruition (Perhaps they also noticed David Foster's lively mouse pointer in Digital Solutions' Pocket Writer 2). In light of all of this we will double our efforts to pursue both GEOS 128 and BASIC 8.0.

Two BASIC 7.0 Compilers Compared *by Todd Madson*

There are a number of Basic compilers on the market for the C-128, among them most notably the Basic 128 Compiler by Abacus Software and the old standby workhorse, Petspeed by Oxford, which is now available in a C-128 mode version called Petspeed 128. Both compilers have strengths and weaknesses, though in my mind there is one clear victor. But first, on to Petspeed 128.

Petspeed 128 is a notable compiler that has existed in various versions since the early days of Commodore computing, and the C-128 version is a four-pass optimizing compiler. Petspeed 128 also has a high level of Basic 7.0 compatibility, as well as a technical section in the manual showing memory maps and the like. Petspeed took most of the programs I fed it and did a good job on them. I also used the Sieve of Eratosthenes benchmark test, which when published in Byte magazine a while back became one of many ways to determine a systems' performance.

To illustrate, in Basic 7.0 the Sieve took 3:02 to complete 10 iterations with 1651 primes found. Petspeed took the Sieve and reduced the runtime to a mere 16 seconds! All is not rosy with Petspeed however: the documentation, while adequate, cannot compare to Basic 128. Petspeed provides a 20 page booklet, while Abacus provides over 80 pages of documentation. Petspeed also has other flaws that hamper its ability as a serious development tool. Once the program is booted up, you are greeted with a full eighty column by twenty five line advertisement for other software by Oxford. If that isn't bad enough, it is followed by perhaps the worst method of copy protection ever devised. Petspeed utilizes "Color Protection." In other words, you must enter a series of letters corresponding to a color in a two dimensional grid matrix that is provided with the program. If you enter an incorrect color in the series, the programs returns you to Basic. If you enter all three colors correctly, you are then prompted for a filename. This is awkward because you are not prompted for your work disk until later, and you cannot obtain a directory or anything else when you are in this section of the program.

Once your work disk has been inserted, the program goes to work. The program takes some time doing what it needs to do so you had best take a break while waiting for programs to compile. One of the more annoying traits of this program is when an error is detected (if you have a 1571 drive you will probably encounter more than one "Parse Tree Error") an absolutely wretched SID chip klaxon siren is sounded, plus the screen flashes from purple to black to red while this happens. A simple error message would suffice - this sort of heavy handed imagery is not only unnecessary, but further shows that while this program compiles things rather well, the user interface designed around it needs to be rewritten. Petspeed also has a lot of problems using 1571 disk drives. If you are going to develop any software using this compiler, it would be advised to format your work disks in 1541 mode (174K) as this program has difficulties with large sequential files. Once your programs are compiled, they can be moved to the appropriate 1571 diskettes. This, in my opinion, is an unnecessary inconvenience and should be rectified as soon as possible. The manual claims that the 1571 drive has bugs in the operating system that cause this, but code can be optimized to take advantage of a machines' limitations. Petspeed 128 also had problems compiling a simple 3 line terminal program. To be fair, Basic 128 also had the same problem, so my advice is: when using RS-232 and compilers twiddling with the code will solve the problem most of the time. Blitz 128 by Skyles Electric Works has been known as the compiler of choice for those interested in compiling BBS programs or terminal emulators.

Petspeed 128 also has options for the compiled code to be set to fast or slow mode, extra programming facilities to optimize source code, advice on making the most of the compiler, and notes on using Petspeed with Machine Code programs.

Basic 128 compiler was written by Thomas Helbig at Data Becker in Dusseldorf, West Germany and is distributed in the U.S. by Abacus Software. Basic 128 differs from Petspeed in that it is designed not only as a Basic compiler, but as a software development system. Basic 128 has several important features that differentiates it from the competition: It can compile in P-Code, which is what most compilers normally produce, actual machine code, or mixtures of both. It also supports fast mode for optimum speed of compiling. There are two levels of code optimization: toggles to set a variable code start, variable memory usage, automatic linkage or disabling of the floating point module. Basic 128 should be considered the compiler to use when developing software on the C-128 without using machine language.

Basic 128 also has an advanced development package that allows you to set options and flags within the compiler, and let you do such mundane and necessary things such as obtaining a directory. The manual is a find in itself as it contains examples such as the Eratosthenes Sieve, advice on how to write programs or alter existing code for Basic 128, and other features like a special extension to Basic 7.0 to allow 640 x 200 point graphics on the eighty column screen. Another nice thing about the manual is that it covers so much ground on how the compiler actually works internally; this also provides hints on how to improve your programs. Optimizing the Sieve in slow mode using P-code gave the same 16 second result as Petspeed did, though selecting true machine code sped up the program even more than that. Basic 128 is a two pass compiler, though it does not seem to suffer because of this. Compiling time is about the same or a little slower than Petspeed.

Two BASIC 7.0 Compilers Compared Continued

Graphics speed on the forty column screen was about the same as it was using Basic 7.0 using both compilers. I could not detect any difference between Basic and the compiled program. These Basic programs were short 3-7 liners from the Transactor that plotted interesting designs on the screen. I'm not sure if this was due to the compiler or if the graphics routines built into the C-128 are already fast enough for most purposes.

If I were in the market for a Basic compiler, I would definitely go for Basic 128 by Abacus. It has superior features and is designed for serious software development purposes. The documentation is great and the author obviously put a lot of work into the program's conception. I give it "Four Stars" out of a possible five.

Petspeed 128 just doesn't cut it when it comes down to doing serious work. It's method of software protection impedes the user in getting any serious work done and discourages repeated use. It is also nearly impossible to use with a 1571 disk drive. If you have a 1571 drive, you should be able to use it when developing software, not neutering it back into the 1541 stone age. Also, the error messages and advertisements become wearing after awhile. If Petspeed had a user interface that didn't waste the users' time with SID klaxons, obnoxious graphic error messages and advertisements, I could be able to recommend it in good conscience. Now I can only recommend it with reservations, lots of them. Petspeed is notable in that it is a four pass compiler, however, Oxford needs to modify the user interface for Petspeed to become competitive with other products on the market.

CP/M Public Domain Essentials *by Randy Margolis*

CP/M, as distributed with the Commodore 128 is really quite a good implementation of this operating system. Not only do you have a totally re-definable keyboard but you can play with screen colors, time and date stamping of files, and other things. There are, however, certain programs and utilities which, if not absolutely essential to the running of CP/M, will make your life much easier. All the items discussed here are in the public domain, and should be readily available in any reasonably supplied Commodore user group library.

First, let me state my own opinion as to the minimum equipment setup required to operate under CP/M. If you own only 1541s (or 1541 clones) you should give serious thought to leaving CP/M alone until you can afford at least one 1571 drive. The operating system is so heavily disk intensive that you will suffer excruciating delays at every turn. In addition, although you can successfully operate with a single drive, your experience will be much more pleasant if you have a second unit, or a Ram expander. If you have to choose between these peripherals, buy the Ram expander first, as it quite delightful to use under CP/M, being lightning fast and completely silent. Just remember that before you power down you must copy everything back to disk! Also, if you lose power during your session, all files in your Ram expander will be lost immediately.

That said, I will now describe the nine items which you should consider 'essential' to your CP/M library:

1. CONF.COM (Size: 6k) - This was written by the fellow who adapted the CP/M system to the C128, Von Ertwine. It allows you to parameterize the system to your equipment. Among other things, you may set your printer type, screen colors and turn off the 40-column screen and speaker (which makes your system run faster).

2. DIRR.COM (Size: 6k) - This lists any disk directory to the screen (or printer) in alphabetical order and, more importantly, with file sizes. It also accurately lists free space remaining on the disk. You'll find that you don't use DIR.COM much after getting used to the luxury of this little gem.

3. SWEEP.COM (Size: 12k) - Sometimes referred to as NSWEEP, this tool is very much like the DOS Shell which you use in the 128 mode, but offers much more. Everything from mass file copying to renaming goofed up filenames to setting file attributes: there is not much in the way of file handling that this utility will not perform, and perform flawlessly. I recommend using Version 2.07, as the later version (2.08) has a sorting routine which does not work on the C128.

4. C1571.COM (Size: 2k) - Also written by Von Ertwine, this utility speeds up writes to disk by a factor of two. As originally implemented, each time a sector is written to disk, it is immediately verified. C1571 turn off this verification. I have found absolutely no loss in reliability in using this program, and running it is the first thing I do after loading CP/M. As the name implies, it is only intended for 1571 disk drives.

5. VDE233.COM (Size: 10k) - After obtaining this program, I guarantee you will forget ED.COM ever existed. This screen editor actually qualifies as a word processor. It fully supports all aspects of printing (including headers and page numbering), macros, search-and-replace, and other features. It can be used for everything from creating SUBMIT files to writing 20-30 page documents. At this writing version 2.33 is the latest one, but

CP/M Public Domain Essentials Continued

earlier versions also work fine, although 2.33 is the most full featured. Make sure you get the 'install file' (VINSTALL.COM) and the documentation file with it. The docs are especially impressive, running a full thirteen pages! I've seen commercial programs whose instructions are not this good.

6. NULU15.COM (Size: 16k) - If you download any CP/M files, then you will need this program. It allows you to extract the member files from a 'libraried' download. Often, related files are collected into one large file with the filetype .LBR. You must have a program like this to get the individual pieces out. NULU15.COM is the last CP/M library utility you will ever need, and has a myriad of functions which allow you to create, as well as dissolve, libraries. As such, you must read the documentation (25 full pages) to utilize it fully. An easier solution is DELBR.COM, a program that allows the extraction of member files with a minimum of fuss. If you're unsure as to your ability to conquer NULU15.COM at this time, try to get both that and DELBR, as it is very handy to create library files for backing up multiple file applications. You can use DELBR right away, and take your time learning the intricacies of NULU15.COM.

7. UNCR.COM (Size: 4k) - Another useful telecommunication-related utility, UNCR.COM stands for 'uncrunch'. If you ever see a filename with a 'Z' as the middle character of the filetype, the file has been crunched, and will be unusable without uncrunching it. This file compression algorithm reduces the stored size of a file up to 40%. It is most effective on text files, but can be used on program files as well. You can also get CRUNCH.COM to create crunched files of your own, which is very handy for long term storage of long documentation files after printing them.

8. MASTCAT.COM (Size: 24k) - This superb disk cataloging program has been around a long time, and is so common that a number of associated utilities have been created independently for such tasks as cross-indexing, etc. It is quite speedy and totally self-documenting. It creates a single data file called MAST.CAT which contains your entire catalog, sorted in alphabetical order, including free space on each disk. Full printer support further enhances its usefulness. If you collect more than just a few CP/M disks, you will definitely want this little jewel.

9. PCFILE.COM (Size w/two associated files: 76k) - Although not strictly a utility, PCFILE.COM (along with the associated files PCSORT.COM and PCEXPORT.COM) is a fine example of a public domain application available for CP/M. It is a surprisingly complete database program and, although it is not Superbase 128, it does many things well. After defining your file layout and adding data, you can search for any record on any field, modify anything, list to screen, printer or disk file (any or all fields), even conditional lists on any field. In addition, you may conditionally extract any records or fields you like. Pretty nice for a piece of public domain software!

I believe that armed with these outstanding utilities and the programs which were supplied with the system disk, you will come to really enjoy your CP/M sessions. There is much confusion associated with new users of this operating system, and these files will do much to alleviate what is really not much of a problem at all.

VIC Chip Smooth Scrolling Techniques *by James O'Hare*

Scenes of stars gliding past an alien spaceship or tanks grinding over a battlefield are familiar to most computer gamers. The effect of motion is simulated by scrolling the background while the foreground object, typically a sprite, remains static or maneuvers only within the confines of the screen. Although scrolling is widely used, the literature offers few explanations of the programming techniques required. This article presents two methods of sideways scrolling on the 40 column screen.

Both methods involve manipulating the lower four bits of the scrolling register (VIC register #22, decimal 53270, hex $D016). The general procedure for right to left scrolling is as follows:

1. Clear bit 3 of the scrolling register.
2. Place some characters on the screen.
3. Set bits 2 - 0 (decimal 7).
4. Decrement bits 2 - 0 until their value is zero.
5. Go to either step 2 (Method 2) or step 3 (Method 1) and repeat.

Scrolling from left to right works on exactly the same principle, but bits 2 - 0 are first cleared then incremented until their value reaches decimal 7.

VIC Chip Smooth Scrolling Techniques Continued

To visualize how the scrolling register works, imagine the screen as a picture and the border as a flexible frame held over it. If you slide the picture a little to the right a gap appears between the left side of the frame and the picture edge. Turning on bits 2 - 0 causes a similar effect by shifting the entire display one character position to the right. In other words, the first column moves over a space and the last column disappears under the border. Pushing the sides of the frame towards each other eliminates the empty gap. Turning off bit 3 does the same thing on screen by enlarging the border to cover the columns on each side.

Moving the picture slowly leftward simulates motion. On screen this is accomplished by decrementing bits 2 - 0. Each time their value is reduced by one the screen display moves one pixel to the left. When the value reaches zero the screen is back to its default position. At that point the whole operation can be repeated.

Method 1 prints some characters to the screen and sets and decrements the scrolling register within a loop. As long as the rows contain identical characters an illusion of continuous leftward movement is sustained. This technique works because the eye detects the slow leftward scroll but not the quick shift right. Try the short program listed below. It is a simple representation of an object moving along a road. You can dress it up by adding sprite data and substituting characters. This technique will also work on the bit map screen if you first enter the graphic mode and use CHAR instead of PRINT to place the characters. To exit the program press any key.

```
15 PRINT [clear home][20 cursor downs]
20 FOR J=1TO40:PRINT"+";:NEXT
25 FOR J=1TO40:PRINT"-";:NEXT
30 POKE 53270,PEEK(53270)AND247: REM CLEAR BIT 3
35 SPRITE1,1:MOVSPR1,160,205
40 DO WHILE Z$=""
45 FOR I=7 TO 0 STEP-1
50 POKE 53270,PEEK(53270)AND 248 OR I:REM SET AND DECREMENT BITS 2-0
55 NEXT
60 GET Z$
65 LOOP
70 POKE53270,PEEK(53270)OR8: REM SET BIT 3
75 SPRITE1,0
```

For most applications Method 2 is more useful. It takes a string of character data from memory and actually moves it across the screen. This requires a machine language routine, listed below. An explanation of ML programming is beyond the scope of this article, but readers of the November issue of "Twin Cities 128" may recognize the indirect indexed addressing technique.

The start address of the string to be scrolled is stored in zero page. Using the Y register as an index, the first 40 characters are displayed on the screen. Next bit 3 is cleared and bits 2 - 0 are set then slowly decremented. The code from 0C1C to 0C28 is simply a delay loop to slow down the scroll. Once the screen has returned to its original position the start address in $FA is incremented. This causes the text to appear one position to the left at the next display cycle. The shift is not noticeable because immediately after the display the screen is offset right again. When the start address reaches the end of the text it is restored to its original value.

Enter the following program with the monitor and save it with the command: S "METHOD2.ML" 8 0C00 0C42

```
0C00 LDA #$80
0C02 STA $FA
0C04 LDA #$0C
0C06 STA $FB
0C08 LDY #$27
0C0A LDA ($FA),Y
0C0C STA $0400,Y
0C0F DEY
0C10 BPL $0C0A
0C12 LDA $D016
0C15 AND #$FB
0C17 ORA #$07
0C19 STA $D016
0C1C LDX #$FF
0C1E DEX
0C1F LDY #$10
0C21 DEY
0C22 CPY #$00
0C24 BNE $0C21
0C26 CPX #$00
0C28 BNE $0C1E
0C2A LDA $D016
0C2D CMP #$C0
0C2F BEQ $0C37
0C31 DEC $D016
0C34 JMP $0C1C
0C37 INC $FA
0C39 LDA $FA
0C3B CMP #$AF
0C3D BNE $0C08
0C3F JMP $0C00
```

After you have saved the ML routine to disk, enter the following Basic program. It POKES a string of data into memory and calls the machine language program. The string consists of 40 spaces, some text, then some more spaces. Run the program and the first eight letters of the alphabet will scroll slowly across the screen until you hit Run Stop/Restore.

VIC Chip Smooth Scrolling Techniques Continued

```
20 BLOAD"METHOD2.ML",B0,P3072
30 PRINT [clear home]
40 FOR J=1TO40:POKE3199+J,32:NEXT
50 FOR J=1TO8 :POKE3239+J,J :NEXT
60 FOR J=1TO40:POKE3247+J,32:NEXT
70 SYS3072
```

Method 2 affords plenty of room for experimentation. Beginners may want to add sprites, use different characters, or alter the delay loop. More experienced programmers can modify the code to scroll more than one line or to run under the interrupt.

Mapping the Commodore 128: An Evaluation *by John Kress*

As a serious machine language programmer, I thought that most of my needs were met when I got my hands on a copy of the Programmers Reference Guide. Then several months later came the McGraw/Hill book Commodore 128 Programming Secrets, which answered the 'Where's the ROM routine for this at?' questions. After I submitted the review for that book I spotted the new release from Compute Books, Mapping the Commodore 128, by Ottis Cowper ($19.95) and the whole picture was changed again.

This tome is probably the single most important piece of reference material that any serious programmer must have in his/her library. As the title suggests, it is a map of the C-128, but that is selling this book far short of the masterpiece that it is. At 689 pages, this is far more than a memory map.

The book begins with Memory Organization as the first chapter, covering Memory Management Unit, Configuration Registers, and bank configurations. The table of bank configurations and the associated ROMs for each configuration is quite good and very readable.

The meat of the book begins with chapter two. This is the beginning of the map, starting with location $0000 and continuing thru $FFFF. Every Ram location or routine that is not used for Basic programming is listed. It begins with the decimal location, its hex equivalent, and the CBM label for the location. Then each location (or pair in the case of vectors etc.) has at least one paragraph explaining the use, type of information, or default value stored there. In cases where there are more than one use for a memory location, all of the cross references for its uses are called out and explained.

Almost every imaginable Basic routine is covered, along with the internal workings of Basic itself. Examples are included for creating an integer value from a string, translation tables for VIC color values to VDC color values, converting an integer into a floating point value, and many more. As a matter of interest, the section on floating point functions is the best one could asked for. Information is also included on converting strings to floating point accumulator (FAC) and back, from memory locations to FAC, multiplication, division, addition, subtraction, interaction between FAC1 and FAC2. In itself, this section alone, is worth the price of the book.

The next chapter deals with the Monitor Rom. In this chapter you'll be able to find many useable routines and gain an insight into how the software engineers at Commodore actual devised the C-128's built-in monitor.

Screen Editor Rom is the next entry, and this is a chapter full of information for the serious programmer. If you want to manipulate your screen, either 40 or 80 column, this chapter lets you in on many of the tricks. There is a lot of good information is in this chapter on accessing the VDC chip via routines present in Rom.

In the next section Input/Output Registers, Color Ram, and Character Ram are covered. There are some interesting entries on the SID chip that many of us are familiar with along with the VIC chip and sprites. Also included is generous information on the Memory Management Unit registers and a nice section detailing the control registers for the VDC (80 column video chip). There is a brief program included to show the 80 by 50 interlaced text screen. Also included is a discussion of the Ram Expansion Controller (REC), which is actually not onboard the C-128, but, rather on the 1700 or 1750 Ram Expansion cartridge.

The final chapter covers the Kernal section of the 128; here are the usual kernal routines and the new 128 only routines. Much of this information is a carry-over from the Commodore 64, but there are quite a few new routines, such as the fast serial mode, the access of the external Ram, and accessing the different memory banks in the 128. The text is quite readable and does an excellent job explaining complex concepts of system operations that appear as only vague references or indistinguishable technobabble in other texts.

Mapping the Commodore 128: An Evaluation Continued

The book closes with the Appendices. Included here is one of interest to many hard core 128 programmers: Bugs and Quirks in 128 Rom. Listed are 17 bugs or quirks that the author has noted, some of which I'd never heard of before. Also found is a very handy 64/128 cross reference map, note frequencies, Basic keyword index, CHR$, screen, and keyboard codes, and an index of memory and routine locations.

All in all I feel that Mr. Cowper and team have produced possibly the single best reference source available to the serious Commodore 128 programmer. The depth and understanding that went into this production is most appreciated.

The Assemby Line *by John Kress*

Of the many nice features of the C-128, the pre-programmed Function keys are among the most useful. These include the eight Function Keys along with the HELP key and the shifted Run/Stop key. All of these keys are programmed to execute various functions, and can be altered, in Basic, very easily with by KEY n, "string"+CHR$(x). This was one of the first features that many of us 'toyed around with' upon first setting up the 128. But Basic has one drawback in redefining these keys: trying to redefine the HELP or shifted Run/Stop keys with KEY 9,"STRING" will return an illegal quantity error. So these keys must be altered in a different way, such as ML. But how does the machine language programmer redefine these keys?

Kernal routines are the backbone of Basic and absolutely necessary for the ML programmer. These building blocks are a collection of small ML programs, each performing a function and take a great deal of the work out of ML programming. There is a new Kernal routine for the 128 that assigns a new string to the function key, called PFKEY. The routine will alter the string attachment for a function key leaving the string at $100A/4106.

Preparing to redefine a function key is as follows: load a zero page pair of pointers with the address of the new string, in the typical low-byte/ hi-byte manner.
Load the .A register with the zero page pointer value.
Load the .X register with the key number to be redefined.
Load the .Y register with the length of the string.
Call the PFKEY routine at $FF65 (65381).
By entering the short example program below, and then when you hit the shifted run stop combination you'll see that that key has a new definition. Enter it with the monitor at $B00.

```
B00  LDA  #$12
B02  STA  $FC
B04  LDA  #$0B
B06  STA  $FD
B08  LDA  #$FC
B0A  LDX  #09
B0C  LDY  #$12
B0E  JSR $FF65
B11  RTS
```

Now enter the following bytes starting at $B12 with the Memory command.

```
B12 53 59 53 33 32 38 30 30
B1A 2C 31 32 33 2C 34 34 2C
B22 36 0D 00 00 00 00 00 00
```

Now to redefine the key enter SYS DEC("B00") and try the shifted Run/Stop combination.

How do the Function keys work? First, a bit of information about the Function keys and their assigned functions. There are ten keys on the C-128 that are considered Function Keys. These are the keys F1 through F8, the shifted Run/Stop key and the HELP key. Starting at 4096/$1000 is a table of information for these keys and their defined strings. The first ten bytes contain the length of the string for each of the keys. 4096/$1000 holds the length for the F1 key, 4097/$1001 holds the length for F2, and so on, through the ten defined keys. The strings assigned to these keys follow, starting at 4107/$100B.

When you call the PFKEY routine the number of the key is stored, then the length of the string is figured, and the string definitions for the higher number keys are moved, to make room for the new keys string. Finally the new length for the string is inserted into the table for the specified key. When the keyboard decoder senses that a function key has been pressed, the function key number is held. The system then adds up the length of all of the prior keys, to get the address of the correct string, and then prints the string to the screen. The entry to the screen is the same as if you typed the command in direct mode.

The Assembly Line Continued

Seeing how this is done, one can understand how putting too many values in a sprite definition can cause a function key to go nuts. When a sprite definition goes beyond 4095/$0FFF, the pointer for the length of the first function key gets wiped out and all the keys are goofed up due to the way their string assignments are figured.

Sparrow's Slick Tips *by Sparrow James*

00110110: Making your ESCape!

One of the most overlooked features of the C-128 is its ESCape sequences. On page 370 of the C-128 system guide there is a listing of these sequences. But just in case these sequences are not enough, here is how you can add more! Locations 824 and 825 ($0338 and $0339) in BANK 0 contain the indirect vector to the C-128's ESC sequence handler. Part of the C-128's BSOUT routine tests if the ESC key has been pressed and then stores the character code of the character that is pressed after the ESC key in the accumulator and jumps to the routine pointed to in locations 824-825. Ordinarily these locations point to 51649 ($C9C1, but there is no reason you can't redirect this vector to your own escape handler. Here is an example that adds two new escape sequences which act like function keys. In this example ESC backarrow (the key above the control key) prints the Basic 7.0 DELETE command, while the ESC up arrow (the key left of the restore key) prints the Basic 7.0 RENUMBER command on the screen. Type in the following assembly language listing using the monitor and then execute the following BASIC program (do not type in the comments).

```
0B00  LDX  #$00           ;Clear X register as a counter
0B02  LDA  #0C00,X        ;Load the accumulator with the Xth character in "DELETE"
0B05  JSR  $FFD2          ;Print the character in the accumulator
0B08  INX                 ;Increment X register
0B09  CPX  #$07           ;Does X=7?
0B0B  BNE  $0B02          ;If X<>7 branch go back and get another character
0B0D  RTS
0B0E  LDX  #$00           ;Clear X register as a counter
0B10  LDA  #0C06,X        ;Load the accumulator with the Xth character in "RENUMBER"
0B13  JSR  $FFD2          ;Print the character in the accumulator
0B16  INX                 ;Increment X register
0B17  CPX  #$09           ;Does X=9?
0B19  BNE  $0B10          ;If X<>9 branch go back and get another character
0B1B  RTS
0B1C  CMP  #$5F           ;Is the character code in the accumulator equal to 95 for the back arrow?
0B1E  BEQ  $0B00          ;If the code is for the backarrow branch to the loop that prints "DELETE"
0B20  CMP  #$5E           ;Is the character code in the accumulator equal to 94 for the up arrow?
0B22  BEQ  $0B0E          ;If the code is for the up arrow branch to the loop that prints "RENUMBER"
0B24  JMP  $C9C1          ;Code is not for back or up arrow, branch to normal ESC handler for processing
0B27  LDA  #$1C           ;Load the accumulator with the low byte of our new ESC handler
0B29  STA  $0338          ;Store the low byte of our ESC handler at location 824
0B2C  LDA  #$0B           ;Load the accumulator with the high byte of our new ESC handler
0B2E  STA  $0339          ;Store the high byte of our ESC handler at location 825
0B31  RTS
```

```
10 A$="DELETERENUMBER"
20 FOR I = 1 TO LEN(A$)
30 POKE 3072+I, ASC(MID$(A$,I,1))
40 NEXT I
50 SYS 2855
```

00110111: Reading and Writing directly from/to VDC registers: Here are Slick ways of writing from/to the 8563 Video Display Controller's internal control registers from Basic. To store a value in any VDC register use: SYS 52684, value, register. To read the contents of any VDC register use: SYS 52698,,register:RREG A (the numeric variable A will contain the specified register's value).

00111000: CTRL-S/No scroll flag: The CTRL-S sequence, like the No Scroll key will halt the execution of a program or listing until a key is pressed. Occasionally this effect is desirable within your assembly language programs. To simulate a CTRL-S type pause simply store any value greater than one in BANK 0 location 2563 ($0A21).

Rumor/Opinion/Mayhem *by Loren Lovhaug*

Recently a very enthusiastic unnamed technoluster (to protect the innocent we will not mention his real name in print, but we will call him "Todd") approached me at a local software store and asked me in a tone that I reserve only for very private conversations with my wife, "Have you read any of the articles on the new Amigas?" Since I try to read just about every Commodore oriented publication I can get my hands on I replied, "Yes, I have." At which point, Todd began reciting at a feverish pace, almost verbatim, all of the statistics he had read about various possible configurations of the Amiga family. When he had finished his litany of technobabble he asked me, "What do you think?" I responded, "Who cares." Downtrodden, and somewhat embarrassed, Todd left me in search of a pair of more favorable ears, which he very quickly found and began his tiresome tirade once again.

Regular readers of most Commodore oriented publications by now have probably seen your fill of preview information on possible new Amiga products. I know I certainly have; and I am not convinced.

While I am amazed with the technological prowess demonstrated by these products, I have yet to find one Amiga dealer, owner, programmer, or writer who has been able to demonstrate to me any significant comparative advantages the Amiga machines possess over my present C-128 workhorses. Oh sure, they can ramble off a plethora of impressive statistics about thousands of colors, two digit clock speeds, concurrent and multitasking operations, and memory expansion into the megabytes, but when it comes to showing me where their hi-tech playmates would actually help me become more productive, they fall painfully silent.

Now please don't misconstrue my intent. This is not going to be the typical Amiga-bashing session you can find occupying most Atari ST bulletin boards. I like the Amiga, and I am glad most Amiga owners like their Amigas, but most people I know certainly don't need one, and most C-128 owners I run into don't want one either. C-128 owners want Commodore to support the C-128 (something many C-128 owners I have talked with don't believe Commodore is doing) instead of concentrating on trying to sell us new Amiga products as an "upgrade" path.

And here lies the true jist of this essay:

It seems clear to me that the people holding the reigns in West Chester (or more likely New York) are in grave danger of alienating the very customers that made Commodore an immensely profitable company in the early 1980's and that have played a major role in helping Commodore recover over the past year from a near devastating slump.

It is certainly no secret to anyone that the foundation of Commodore's success over the years has been in the so called "lower-end" of the personal computer market, and not in what I like to call the "professional world of overpriced computing". The relatively inexpensive and functional computing products Commodore has been selling over the past years, have appeal to a special kind of consumer. A consumer that is sometimes referred to in the hallowed halls of some corporate headquarters as "small-timers" (i.e. Mr. & Ms. lower-middle-class [translation: YOU AND ME]). "Small-timers" are generally termed as hard-sells (difficult) in the computer industry because they generally have a low amount of expendable income for luxury items such as computers (yes..I said it, computers are luxury items). In other words, computer manufacturers/ retailers have to work a lot harder to make the "small-timers" part with their pennies, than they do luring "Mr. Corporation" into pouring his expense-account laden mega-bucks into the latest in dazzling status-symbol state-of-the-art technoplaymates.

This perhaps is why all of the major computer manufacturers in the U.S. today including Commodore are targeting "Mr. Corporation" with their latest wonders, instead of trying to persuade "Mr. and Ms. Lower-middle class" to dedicating a substantial portion of their paychecks to computers they can afford, understand, and use.

At this point, I can hear those of you who are excited about the prospects of a low-cost Amiga (Amiga 500, Amiga Jr., ad nauseam) about to chime in rebutting my above arguments, claiming that such a product might be Commodore's new scheme of supporting low-cost computerists. But I caution you to consider the following: 1) Although higher priced Amiga products have been officially announced, have you really seen anything besides rumors about lower-priced products being introduced in the U.S. and 2) Even if a cost-reduced Amiga does find itself on these shores, the apparent cost of such a machine may be deceptive when you consider the fact that supporting your investment will likely entail the purchase of new software, peripherals, and goodies to truly exploit the "apparent" potential of such a machine. Remember, goodies like high persistence RGBA monitors, video digitizers and genlock devices, and two megabyte RAM expansions don't come cheap. In this same vein I have yet to meet an Amiga owner that does not complain that he/she NEEDS at the very least more memory and/or a second disk drive, to do things they feel they bought the Amiga to do.

This is not to suggest that Commodore is about to make orphans of us 1.2 million C-128 owners. On the contrary, as I have stated here many times before the C-128 and C-128D have been and continue to be defacto profit-reaping best

Rumor/Opinion/Mayhem Continued

sellers. However, the more I talk to people at Commodore I get the feeling that we are being "de-emphasized". That does not mean we are being ignored, after all, they are more than willing to take our cash for a second C-128, or that nifty new RAM expansion or disk drive. But as far as continued (where are those upgrade ROMs?) and future product development it appears that we are becoming a low priority.

Now I realize that technology can't stand still. I don't believe that Commodore should be marketing C-128s in the 1990s. But I do believe that Commodore management has to be very careful about damaging the morale of current C-128 owners. After all, when/if current C-128 owners decide to purchase another computer, it seems reasonable to me that they will be much more likely to make that next computer purchase a Commodore computer, if they believe the company steadfastly supports the product it sells.

And so lies Commodore's dilemma. Whether or not Commodore is or isn't supporting the C-128 properly it seems to me that it might be prudent (if not crucial) for Commodore to convince existing C-128 owners (especially the skeptics) that they are supporting the product. As I remarked in our last issue, I do believe the announcement of a U S. version of the C-128D and the 1581 disk drive are positive signs. However, to many users these new products appear as just a ploy to squeeze a few more dollars from their pockets before the axe is dropped. So any support endeavors that Commodore pursues, to be effective from a public relations standpoint, must be as inexpensive (if not free) as possible for the end user. Here are my suggestions:

1. Get the C-128 & 1571 upgrade ROMS out! I realize that "bug fixes" are difficult to implement and that they take a great deal of time and effort. But from from the conversations I have had with the individuals responsible for the actual "fixing", the "bottleneck" is no longer in engineering. Getting these long awaited and promised "fixes" into the hands of the general public at a moderate price would go a long way in bolstering the idea that Commodore is actively supporting the C-128.

2. Enhance the usefulness of existing C-128s and peripherals by releasing various "in-house" projects to the public domain or market them for a minimal fee. Although some of these "never-released" projects have proved impractical to mass-market, they have cost Commodore a great deal in terms of development resources without earning any return upon the original investment. These include: 1) The native mode RAM disk software for the 1700/1750 Ram expanders, 2) The C-128 Assembler/Editor package, and 3) Plans/schematics for various interfaces and adapters such as hardware UARTS for incredibly fast telecommunications, transparent hard drive interfaces, and clock cards etc. which if released would promote a huge amount of goodwill and would encourage a great deal of third party and public domain C-128 hardware and software development.

3. Actively support and endorse and even help finance third party developers who market C-128 specific products. This of course includes extensive support of North America's only C-128 specific publication.

4. Begin an extremely aggressive educational marketing campaign which includes extremely attractive pricing on C-128 and C-128D systems for educational institutions and the creation of an educational support staff that truly works with educators much in the same way Apple has done in the past. I think many school districts would see the C-128 as an excellent choice for their classroom computing needs IF they were approached in the right manner. Getting C-128s into the schools and supporting educators who use Commodore computers would be perhaps the single most important image-building move Commodore could make. After all, Apple has been coasting on their educational marketing efforts for years and given the abysmal student to microcomputer ratios in our public schools, Commodore would not by any stretch of the imagination be too late.

5. Create a real customer department that not only accepts questions and complaints from end users, but actively solicits them and tries its hardest to achieve customer satisfaction through genuine hard work and honesty.

Do I think any of the above will happen? Maybe. If I were calling the shots in West Chester they would. By the way, if I were the boss in West Chester, or New York. Commodore would be marketing only three machines in North America: the C-128D (with built in 1571 drive) at $350, The PC-10 with 512K of Ram, two floppy disk drives, and the necessary graphics and I/O cards for $750, and the new Amiga 2000 for $1499 for the base system with the PC card as standard equipment. Of course, given the customer support programs and the streamlining of the product line I have just mentioned, I'd probably run the company into the ground...or would I?

SID Player 128 *by Randy Margolis*

If you are reading this, chances are that you are already somewhat aware of the considerable number of really fine commercial programs available for the C-128 in its native mode. Until now, however, these application programs have

SID Player 128 Review Continued

been concentrated in the area of what is commonly termed 'productivity' software: word processors, database programs, spreadsheets, and the like. Two areas that have been sorely lacking in commercial offerings have been entertainment and music. I recently came across a product from COMPUTE! books called 'The Music System for the Commodore 128 & 64' (also known as 'The Enhanced Sidplayer'). For $24.95 you get a complete sound processor system on disk and a comprehensive manual. There are absolutely no programs to type in, making this a legitimate commercial software package rather than a magazine type-in exercise. In addition, since you really need the manual to use the program, COMPUTE! has wisely chosen not to copy-protect the disk. I should mention that if your interest lies in the direction of MIDI capability, this is not the product for you; but if you are interested in a first class music and sound editor, I believe you will be ecstatic with the program. Also, 80 column mode is not supported, although I did not find this limitation constricting.

Having been a great fan of 'Master Composer' (an accomplished music editor for the C-64) but frustrated with its limitations using it on a C-128, I approached this product with much anticipation, and not a little trepidation. Would it do everything the former program would do, while utilizing the considerable enhancements of the 128? Much to my delight, I found that Enhanced Sidplayer is amazingly complete and performs SID magic that I didn't even realize was possible. Another concern I had was that of the ease of note entry. Again, my fears were allayed by an excellent keyboard entry system, paralleled by a wonderfully easy joystick method. The disk is (naturally) self booting and presents you with a menu of functions including a player program (if you only want to hear previously entered music), the editor (the main portion of the system) or any of several utility modules. Also included are numerous sample music files, some of which are of professional quality.

The editor allows you to enter both notes and commands in one voice at a time, but when working on a particular voice you may at any time play all three voices in unison without leaving the editor screen. It is here that you enter notes (along with their pitch and duration) either from the keyboard or with the joystick. You also indicate the key signature and time signature (both changeable at any time). Both the insert and delete keys are active, and there is an 'undelete' buffer (holding up to 128 notes) to protect you from mistakes. Enhanced Sidplayer makes full use of the C-128's memory, allowing approximately 18,000 notes and commands! With judicious use of repeats and defined phrases, single compositions can easily last 30 minutes or more. All accidentals are supported including double flats and double sharps. Triplets are available for any duration up to, and including, 64th notes. Really unusual note rations (such as quintuplets and septuplets) are possible using the 'utility duration' command. With this you actually set the number of jiffies the note will sound. There is a comprehensive chart listing the number of jiffies each standard note sounds at each tempo, and, by doing a little arithmetic, any duration is possible.

A touch of the F3 key brings you to the 'Command Screen', where you can do an incredible number of things with the notes you have entered. Enhanced Sidplayer is capable of audio gymnastics that I had no idea were possible with the SID chip. An example of this is 'portamento' (also known as glissando). This is the effect of gliding to the next note not unlike a slide trombone. Also at your command is true vibrato, in which you have full control over the rate and depth of the effect. Another fascinating toy is the 'pulse-width sweep' command. This varies the pulse width during the note, an effect I find amazingly useful for avoiding that 'computer generated' syndrome which can easily plague your compositions. In addition to just sweeping the pulse width, you can produce another type of vibrato effect by using the 'pulse-width vibrato' command. This rapidly changes the pulse width up and down according to your instructions. The DTN command slightly detunes a voice, and is very useful when two voices are playing in unison. This is used in a bagpipe demonstration on the program disk, in which you couldn't tell the result from the real thing. This technique can also be used to play quarter tones if your particular musical quirks run in this adventuresome direction. Of course, you also have full control over waveform and envelope makeup (including sync and ring modulation), and complete filter control.

In addition to volume settings, there is an extremely useful bump command which makes crescendos and decrescendos easy. With it, you can bump the volume up or down one value any time you want, without worrying about what the absolute value currently is. In order to make note entry less laborious, there are two repetition functions. Simple repeating of a group of notes is accomplished by the HEAD and TAIL commands. This simply repeats the sequence of notes between the two commands as many times as you want. This is very useful for situations where a voice has, say, ten measures of rests. Just indicate HEAD10, then one rest, then the TAIL command, and presto! For more complex repeats, there is the phrase commands. You define the beginning and ending of the phrase, and then call it anywhere else (even in a different voice!), and it will replay with all its embedded commands. This works almost exactly like a Basic subroutine.

those of you who want to have words with your music, Enhanced Sidplayer will happily oblige. Just enter the words in Speedscript or EasyScript format (the latter should support any sequential file format), and run either of two supplied conversion programs which will actually create the 'words' file in precisely the format that Enhanced

SID Player 128 Continued

Sidplayer needs. Finally, place the FLAG command in your music where the next line of verse is to begin. When the song is played in the 'Singalong' mode, your words will scroll by on the bottom of the screen based on your timing! If there is another program that does this I surely haven't seen it.

Want to play your songs in a Basic program? No problem, since COMPUTE! has provided a small ML module which you can load from your Basic program to play any of your compositions. There is even a HALT command which you can imbed in the music to stop play temporarily when you desire. All key memory locations are described so that you can control how your sounds play during the execution of your program.

If all this sounds a little overwhelming, please be assured that you can tackle the many capabilities of this excellent package at your own rate. Every command has default settings which allow you to just enter notes and enjoy their playback without any struggle on your part. When you are ready, you can add commands to pre-existing music and re-save it anytime. I haven't been this excited about a new C-128 offering for a long time. My only complaint is that too few people know about it. For a suggested retail price of $24.95, Enhanced Sidplayer is an astounding bargain!

Merlin 128 *by John Kress*

1987 is proving to be a boom year for the Commodore 128. The great number of products, programs and supplements to the C-128 are making this computer a very popular personal system, and Merlin 128 is another long-awaited entry for this C-128 user.

Merlin 128, from Roger Wagner Publishing, Inc. and written by Glen Bredon, is a complete Macro Assembler Editor System, as complete as most any programmer could ask for. Merlin runs exclusively on the 80 column screen and requires a 1571 disk drive. Some may be familiar with the C-64 Merlin, and feel right at home with this system, while others may find it a bit different but easy to use. If that's not enough the boot disk contains many source code listings and examples of the proper syntax for many of the more difficult operations.

Merlin is not copy-protected, so damaging, drive head-banging is not a problem. The manual suggests that you make a back-up copy and use that for your daily work disk. Included on the Merlin disk is a copy program, which is the one thing that I found that was flawed. When run, I always got a disk error #63, illegal track and sector. I did try some other copy programs without a hitch, however, including the DOS SHELL's disk copy. A small point to knock, this was the only problem I found.

Upon booting the program, the user is at the main menu, the gateway to the Merlin 128 system. Here you'll find various sub-operations within Merlin, such as Enter ED/ASM, Disk commands, Load and Save either source, object, or text files, a Monitor, Run a program, executing one of the utilities for Merlin, etc. The editor is where most of the work will be accomplished.

The ED/ASM editor has some great features for formatting the source listing, set up in the familiar 'LABEL OPCODE OPERAND ;COMMENT' format, like most editors. But, suppose that you don't want a label entry for a line. Press the space bar, and the cursor jumps to the opcode position, press it again after entering the opcode and the cursor is at the operand field, and so on. I found this much more convenient than the TAB key that's used on other systems, because the space bar is an easier keystroke. Along with the normal key entries are a Control and C= (Commodore key) set of key values. These include cursor controls, deletions, insertions, and a recall or OOPS command. Also, there is a set of ALT keys, which are a set of key macros that have a source code listing on the disk. These keys can be expanded, edited and altered to suit your own needs by editing the source file and re-assembling it. These are quite handy for entries that can get tedious, such as a routine to save the system registers. Press and hold down the ALT key and type 'p' and the routine to push all of the registers on the stack is automatically entered to the screen. To restore the registers press the ALT key and 'P' (SHIFTED p) and the registers are pulled and restored to their prior conditions. The versatility of the editor doesn't end there either, as there remains the EDIT function of the ED/ASM. Here you can LIST, all or portions of the source code, DELETE portions, lines, ranges of a line. There are CUT, PASTE and MOVE functions, a OOPS or UNDO last function for mistakes and even a CLIPBOARD to temporarily store lines for later use. This portion of the system rivals that of many good word processors.

The ASM portion of the ED/ASM is a full-featured, two pass assembler with all the bells and whistles imaginable. During assembly you have the ability to input values to the object code, and get input from the keyboard for variable values. You can, at assembly decide to skip over portions of a source listing and DO other portions with keyboard input to screen prompts. With Merlin you can build a program in modular form, assemble each MOD

Merlin 128 Review Continued

d test it out for bugs and work onward and upwards. Merlin also allows RELoactable code modules, which is very handy for the programmer who wants to build a number of routines that are commonly used in many programs.

When it comes to string definitions, Merlin really shines. In this department there are six pseudo-opcodes for defining a text strings and a couple of variations on each opcode. These include TXT, DCI, which will put together a PET ASCII string, ASC and ASI define a true ASCII string, REV will assign a string in PET ASCII in reverse order (cat becomes tac), and STR which will leave the length of the string as the first byte, followed by the string itself. This versatility is just another example of the powers within the Merlin system.

Another desirable feature is that this system is case sensitive, meaning that labels like START, Start, and start, will assemble to be three totally different labels.

MACROS

Macros are a very valuable tool when working in assembly, and Merlin has capabilities here that most users may not even begin to tax. With eight variables for usage, local labels, and full compliment of conditional pseudo-ops, the capabilities are nearly limitless. This allows the user to build very flexible and complex macros, to build up a library of common routines that can otherwise be tedious to enter, again and again. Also macros can be nested (one macro calling out use of another) to a depth of 15. In this aspect Merlin rivals the ED, MAC and RMAC CP/M assembler system.

Storage defining is much the same as any of the many ED/ASM systems I've seen, with the one exception of the FLO pseudo-op. With this a five byte floating point number is assembled from a defined value. This feature, along with a good rom map, will open up a whole new world of math to those who have a weakness in floating point math.

UTILITIES

From within Merlin 128's main menu a number of utility programs may be executed. These utilities can be loaded into Merlin and when executed, will either divert or alter assembly functions. This is more proof of the power of Merlin. The first of these is LINKER, which will assemble a number of relocatable source files into executable object code. With LINKER code can be assembled up to one page short of 42K.

The next utility is the SOURCEROR, a disassembler, of which there are two versions. Version one is able to disassemble an object code file up to 6K in length. SOURCEROR.XL is a disk based version that can handle programs almost 32K in length. The difference is lies in the fact that the .XL version saves the source and label tables to the disk, rather than leaving them in the C-128's memory. The disassembler will assign labels to all recognized addresses and this information is taken from a file on the disk. For example, when disassembling a program all JSR $FFD2 instructions would be listed as JSR BSOUT.

Another useful utility for debugging an object code is XREF and a companion program XREFA. These are cross-reference generators, which produce a listing of all labels, their address in the object code, the source listing line where defined, and each line where the label is used. Output is generated in alphabetical and numerical ordered tables, with options to produce these tables along with an assembly listing, a XREF table, or just an alphabetical or a numerical cross reference table. XREFA is used for larger files and multiple PUT file assembly, and will produce an address only listing. When used with the equates table and the usual label table from the source code assembly, this will give you all of the needed information to trace through a listing.

Printfiler is another utility that will divert the assembly listing that normally goes to the screen, into a disk file. This can be a great time saver as well as a paper saver. When used it will slow down the assembly process, but it is much faster than assembly with outputting a listing to the printer. If, for example, there is an error in the source code listing and you've set output to the printer, you've just wasted a lot of paper. With this utility you have a disk file that can be scratched, or even loaded into your word processor for later print-out. This also is a nice method of developing a disk listing that can be transferred to your EDITOR. With this utility you have options of echoing the listing to the screen, and compressing the diskfile; however, compressing the file will require that you develop an uncompressor on your own.

If you aren't drooling yet, in addition to all this there are a number of executable programs on the Merlin disk. These range from a disk 'ZAP' program to a PI calculator. ZAP will let you look at a disk on screen, view the BAM, read any track and sector, write or edit any track and sector and unscratch or recover a file.

.ie of the old all time favorites is here, SWISH, which is a graphic display using 6502 code to generate the line and color display rather than the C-128's rom routines. These routines are claimed to be about four times faster than the

Merlin 128 Review Continued

rom versions and can be found in an object file titled HIRES.O.

Another program is Ram Test, which will POKE byte values into the system ram and print out any errors.

Also included are a few examples of source code. The P1 series is an example of relocatable files that use the linker program to assemble. These give you a good example of the proper method of writing these type of source listings, as well as having a good number of floating point math examples on board. Other source examples are GETERR, which reads the disk error messages and prints them to the screen. INPUT is a routine to get input from the keyboard and store the keystrokes in memory. There is also a source listing for the Kernal equates, that you can easily put into your assembly listing to save you all of that entry.

CONCLUSIONS

Merlin 128 is a disk based Assembler Editor, which will find a place in the software market very close to those other greats for our machine. The syntax, although a bit different from most assemblers I've used, is well worth any time taken to master. If you are a beginner with assembly, get this one. You will not have to upgrade to a better system. If you are one of those comfortable with 65XX assembly this is the one you'll most likely want to have. I feel that this system is very nearly as powerful as the Digital Research package for the CP/M side of our machines.

I truly think that this package will prove itself and be put into the same class as those other C-128 greats, Pocket Writer, BobsTerm, Superbase, etc. If you see it on the shelf, don't hesitate.

Pointers on POINTER *by Miklos Garamszeghy*

BASIC 7.0 contains an interesting but frequently neglected function called POINTER. This routine returns the memory location of the value of any variable. Why would you want to know this? One answer is that it is an easy way to pass parameters from BASIC to machine language, and even back again.

Before we examine the possible applications for POINTER, let's see what it actually does. The syntax for POINTER is:

A = POINTER (<variable name>)

<variable name> can be any valid BASIC variable, floating point ("B"), integer ("C%"), string ("D$"), or array ("E(1)", "F%(2)", "G$(3)"). "A" is the memory address in BANK 1 associated with that variable. A value of zero indicates that the variable has not been defined yet.

Unfortunately, different types of variables are handled in different ways by POINTER. For numeric variables, POINTER returns the address at which the actual value is stored, in either 5 byte floating point format or two byte integer with 3 pad bytes. For strings, POINTER returns the address of the string reference table. The first byte of the string entry is the length of the string. The following two bytes (POINTER + 1, and POINTER + 2) contain the low and high bytes, respectively, of where the string bytes themselves are actually stored. This is followed by two pad bytes. Table 1 is a reference chart which outlines how different types of variables are stored in memory and how the POINTER function acts on each.

Perhaps the most frequent use of POINTER is to pass parameters, such as text strings, from BASIC to machine language. The following short program uses that principle to re-define the <help> and <shift>-<run/stop> keys on the C-128 in a very elegant fashion: using the KERNAL PFKEY routine at dec 65381.

```
10 T$="<new key definition>"                    where K= 1 to 8 for F1 to F8
20 BANK 1:AD=POINTER(T$)                              9 for shift-<RUN/STOP>
30 LE=PEEK(AD):LO=PEEK(AD+1):HI=PEEK(AD+2)            10 for <HELP>
40 POKE 250,LO:POKE 251,HI:POKE 252,1
50 BANK 15:SYS 65381,250,K,LE
```

Another KERNAL routine which often requires a text string input is the SETNAM procedure for OPENing disk files. The technique outlined above can be adapted for SETNAM quite easily.

This next example will take a string defined in BASIC, alter it with a short machine language routine, the send the changed string back to BASIC again and print it. Look at the new string closely and compare it to your

Pointers on POINTER Continued

original entry. The machine language portion is quite simple and is POKEd into the cassette buffer. You can assemble it with the MONITOR if you are interested.

```
10 BANK15:FOR I=2816 TO 2869:READ X:POKE I,X:NEXT
20 PRINT"<CLR>":INPUT"ENTER SOME TEXT";T$
30 A=POINTER(T$):BANK1:A1=PEEK(A):A2=PEEK(A+1):A3=PEEK(A+2)
40 BANK15:SYS 2816,A1,A2,A3:PRINT"<DOWN>IS THIS YOUR TEXT?":PRINT T$
50 DATA 133, 252, 134, 250, 132, 251, 160, 0, 162, 1, 169, 250
60 DATA 32, 116, 255, 153, 0, 12, 200, 196, 252, 208, 241, 169
70 DATA 250, 141, 185, 2, 169, 12, 133, 255, 164, 252, 136, 132
80 DATA 254, 160, 0, 177, 254, 162, 1, 32, 119, 255, 230, 250
90 DATA 198, 254, 16, 240, 96, 0
```

The second major use for POINTER allows you to BLOAD and BSAVE large arrays of numbers. If complex calculations are required, setting up an array can take a long time. For example, the following short program takes about 40 seconds to execute in SLOW mode or 20 seconds in FAST mode. (The program calculates the order in which tracks and sectors are filled on a C-128 double sided GCR CP/M disk on the 1571 and stores them in arrays T() and S().) A few selected values will be printed out. Write these down if you want for future reference.

```
10 DIM T(1360),S(1360),SM(36):T=1:S=10:TC=2
20 FOR I=1TO17:SM(I)=20:NEXT:FOR I=18TO24:SM(I)=18:NEXT
30 FOR I=25TO30:SM(I)=17:NEXT:FOR I=31TO35:SM(I)=16:NEXT
40 FOR I=0TO679:T(I)=T:T(I+680)=T+35:S(I)=S:S(I+680)=S
50 S=S+5:TC=TC+1:IF S>SM(T) THEN S=S-SM(T)-1
60 IF TC>SM(T) THEN T=T+1:S=0:TC=0:IF T=18 THEN S=5:TC=1
70 NEXT:PRINT T(45);S(97);T(246);S(982)
```

RUN the program, then add the following line

```
80 BSAVE"T + S",B1,P(POINTER(T(0))) TO P(POINTER(S(1360))+5)
```

and type in GOTO 80. This will BSAVE the entire contents of the arrays T() and S() into a 54 block disk file named "T + S". Now press the reset button (or if you are a non-believer, turn off your C-128 for a week and leave it just to make sure that all RAM has been completely erased) and RUN this next program:

```
10 DIM T(1360),S(1360)
20 BLOAD"T + S",B1,P(POINTER(T(0)))
30 PRINT T(45);S(97);T(246);S(982)
```

The array contents will be retrieved from the disk file in 6 seconds flat using a 1571 drive!! This is certainly much faster than calculating the arrays each time you need them. Of course, the more complicated the calculation involved in setting up the arrays, the greater the advantage of BLOADing. The BLOAD speed is only a function of the array size, not its complexity. In addition, the BLOAD method generally takes up less space in a program and will keep your calculation methods away from prying eyes. The most that a snooper can do is to get a listing of the contents of the array. How you arrived at them remains your secret.

Two general caveats should be noted with this method. The first is that the DIM statement must be the in the same order and format in both the BSAVEing program and the BLOADing program to ensure that the array contents are BLOADed into the correct area of Ram. The second is that array contents will move around in memory depending on the use of other variables. Always get a fresh POINTER just before you need it.

Multi Dimension Arrays:
same as corresponding type above, but additional entries in header portion for each of the additional dimensions. A multi dimensional array AB(x,y...n) would have the following format:

65 name
66 bytes

length of total this array entry (i.e. offset to next array in
array pointer table) low byte/high byte format

Pointers on POINTER Continued

dd number of dimensions in array

nn number of elements in dimension n
nn high byte/low byte format

yy number of elements in dimension y
yy high byte/low byte format

xx number of elements in dimension x
xx high byte/low byte format

POINTER(0,0,...0) would return the next address:
00 data, in format for type of array, arranged in order of x major
00 then y, then additional dimensions
00

Advanced Superbase 128 Usage *by Loren Lovhaug*

Superbase 128 by Precision Software is potentially the most sophisticated database program for the C-128 in native mode. I say potentially, because the thing that sets Superbase apart from all other database management programs for the C-128 is Superbase's built-in programming language. Most Superbase users tend to ignore this powerful facility, opting instead to stick to Superbase's convenient but stifling menu system. But with Superbase's programming language you can learn to automate repetitive tasks and perform complicated data manipulations.

But before we begin, please keep the following in mind: 1) This series should not be construed as an alternative to reading the Superbase manual. The Superbase manual represents an exhaustive attempt on the part of Precision software to provide the user with all the information he/she needs to get started and continue to use Superbase effectively. Although the manual is not perfect, there is simply no substitute for it. Our assumption is that you have read the manual and worked through the examples in the tutorial section. If you have not already done so, it is probably wise for you to stop reading this article at this time and come back to it after you have read through at least the tutorial section. This series will not cover the basics such as creating a database or designing a record layout. 2) You should always practice the techniques we present here on "junk or test data" and not on anything crucial until you fully understand the technique. And remember, always make a backup of your important data before you attempt anything which will write or modify data to the disk. A little caution and prevention now can avoid a great deal of headaches and nightmares over the long haul.

Using and creating lists

In just about every advanced database application there lies a primitive task called list identification and creation. As an example consider the following typical database application: a company wishes to periodically send out personalized letters to its customers with outstanding accounts. The above example involves the repetitive creation of various lists which can be accomplished quite nicely with Superbase 128 and just about any word processor capable of doing a mail-merge.

The first step in solving any data manipulation problem is to clearly identify or outline each task our Superbase program will have to accomplish to reach our desired goal. Given a Superbase customer file with the following fields: Account (key), Name, Address, City, State, Zip, Salutation, and Balance, our "outstanding accounts" program our task outline might look something like this:

1. Create a list of customers with outstanding accounts.

2. From the list created in task #1 print address labels. (Note: Since we have customers in foreign countries it might be desirable to print these labels in some kind of group order so that the proper postage can be easily affixed to each envelope)

3. Create a disk file for mail merge purposes using our word processor which contains the following information for each customer: Name, Address, City, State, Zip, Country, Salutation, Account Number, and the outstanding balance.

As you know Superbase 128 uses one field called the key or index field to identify and record individual records. When Superbase 128 creates automated lists, it stores the key or index field data for a particular record in a

Advanced Superbase 128 Usage Continued

sequential file. For this reason these files are called key lists. Three primary commands that can be used to create key lists, namely FIND, OUTPUT, SORT. In addition you can use Superbase 128's OPEN..CLOSE sequence to create key lists as well.

To accomplish task #1 we will use the FIND command to create a key list that we will call "out-list" that contains all of the key fields (the account numbers) of the records with outstanding balances. To accomplish task #2 we will use the SORT command to create a new key list called "out-list-s" that will be sorted by country so that all of the labels for letters going to a particular country will be going to the same country will be grouped together. Then we will use the OUTPUT command to print the necessary address information to the labels on the printer. Lastly, we will again use the OUTPUT command to create a disk file called "out-data" for our mail merge from the "out-list-s" key list (we use the "out-list-s" list so the mail merged letters will be printed in the same order as the labels to facilitate easy "stuffing"). Examine the following Superbase 128 program:

```
10 cr$=chr$(13): rem carriage return
20 FIND "out-list" where [Balance] is ">0"
30 SORT "out-list" on [Country] to "out-list-s"
40 DISPLAY @10,10"Be sure you have labels in"
50 DISPLAY @10,11"your printer and press any key"
60 GETKEY Z$: rem wait for user to press a key
70 PRINT :rem send subsequent output to printer
80 OUTPUT ACROSS FROM "out-list-s" @5[Name];cr$;
   @5;[Address];cr$;@5&[city]&[state]&[zip]
   &[country];cr$;cr$;cr$;cr$
90 DISPLAY .rem send subsequent output to screen
70 OUTPUT DOWN TO "OUT-DATA" [Name][Address]
   [City][State][Zip][Country][Salutation]
   [Account]
```

The above program assumes you have selected the proper database and filenames in advance and have a properly interfaced printer connected to your system. You might want to study the way I used the ACROSS directive, the string variable cr$ defined as the ASCII code to send a carriage return to the printer and the ampersand and at sign directives to format the data on the address labels in line 80 (see your Superbase manual for full explanations of ACROSS, @, and &). Note: In the above example I assumed single column labels with six lines per label. but depending upon your situation you might use line 80 in a slightly different manner.)

Sparrow's Slick Tips *by Sparrow James*

00111001: Default inputs from BASIC 7.0: From: Sparrow James

Often it is desirable to include default responses within your Basic programs as a courtesy and to aid users of your program. In many situations, this type of program design consideration will not only be more convenient for a user, since he/she will not be forced to type the most common responses, but in combination with a well-worded prompt message will result in a more professional looking and user friendly program. The following program example demonstrates a useful technique for creating default responses to INPUTs through the use of the C-128's keyboard buffer at locations 842 - 851 ($034A - $0353).

```
10 DF$="DEFAULT":GOSUB 1000
20 INPUT "PROMPT MESSAGE";A$
30 PRINT "A$ WAS DEFINED AS:";A$
40 END

1000 FOR I = 842 TO 842+LEN(DF$)
1010 POKE I,ASC(MID$(DF$,I-841,1))
1020 NEXT I
1030 POKE 208,LEN(DF$)
1040 RETURN
```

In the example listed above the string variable DF$ is defined as the desired default definition. Then the subroutine at line 1000 is called to place each character of DF$ sequentially into the C-128's keyboard buffer (lines 1000 - 1020) and to inform the C-128's Basic interpreter that the number of "pseudo keypresses" waiting to

Sparrow's Slick Tips Continued

be processed (line 1030). The result is that the default response is automatically displayed on the screen following the INPUT statement's prompt message and question mark. From this point the user simply has to hit return to choose the default response or if he/she chooses the default response can be edited or typed over completely using the insert/delete key. Note: One important thing to keep in mind when using this technique for creating default responses is that the default keyboard buffer is 10 bytes. This means that your default response can only be 10 characters or less in length unless you tell the C-128 to make its keyboard buffer larger.

00111010: Better Video Display on Higher Resolution Monitor: From: Jeff Adams
For those of you with a RGB monitor, here is a little-known trick to really make 80-column display shine. The Commodore C-128 Programmers Reference Guide states that register 9 of the 8563 80-column video controller chip controls the number of raster lines used to display a character. By using the two memory locations that serve as gateways to the 8563 video controller chip, you can increase the number of raster lines per character, giving you a much better looking display. The pixels that make up the character set are placed closer together, making the 80-column mode easier to read and easier on your eyes. Try executing the following one line program on your monitor:

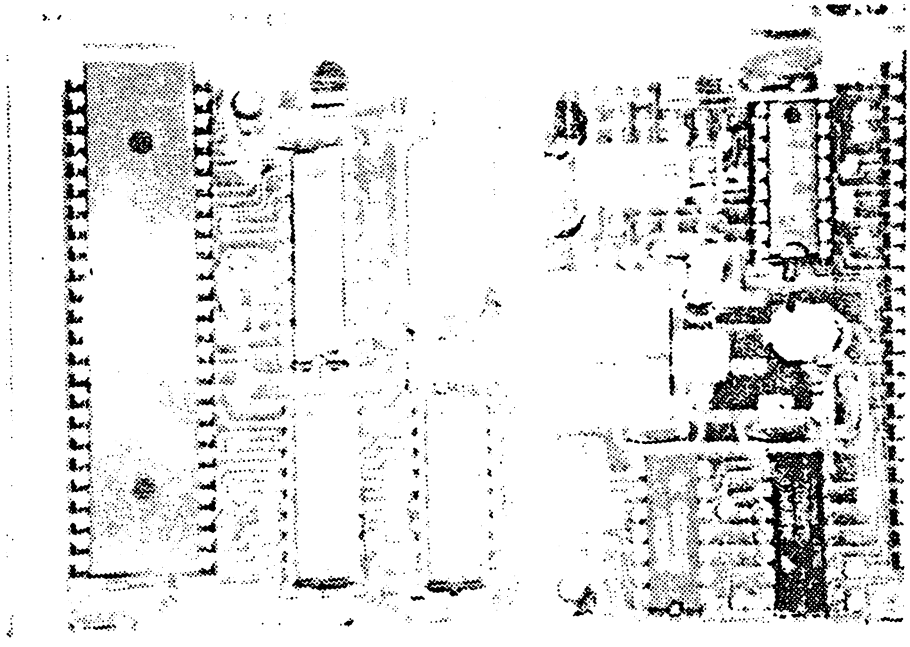
```
10 POKE 54784,9 : POKE 54785,232
```

The value in the second poke (232) is changes the number of raster lines per character. Normally the value of this number is 231. The value of 232 works well with my Magnavox 8563 high resolution RGB monitor. I have also heard that the display on the Commodore 1902A can be dramatically improved with these pokes.

You might try changing the number in the second poke, varying it between 231 and 235. If the screen image starts to roll vertically, then the poke you tried exceeded the capabilities of your monitor. By pressing Run/Stop and Restore, the computer's video chip is returned to the start-up configuration.

These simple pokes also work with several commercial software packages. Try executing the pokes before running the software. I have had excellent results with the 80-column version of Multiplan (TM), Pocket Writer 128 (TM), and VT100-128 (C).

00111010: Using your 1700/1750 RAM Expansion with GEOS: From: Sparrow James
Although the C-128 version of GEOS from Berkeley softworks is not yet available, C-128 GEOS owners who own either the 128k 1700 Ram expansion or the 512k 1750 RAM expansion can utilize this peripheral to greatly enhance the performance of the current GEOS system. The latest upgrade of GEOS supports the Ram expansion cartridges as a Ram disk which allows almost instant loading of GEOS application programs and greatly enhanced data manipulation. The upgrade comes in the form of a program which alters your GEOS system disk. The update program is called desktop and kernal revision 1.3 and can be found on the backside of the 1351 mouse demo disk and the 1764 Ram expansion test demo disk as well as many telecommunications networks.



Rumor/Opinion/Mayhem *by Loren Lovhaug*

Since the last issue of Twin Cities 128, I have been doing a lot of thinking about the effect the new Amiga products announced by Commodore will have upon the C-128. Some of you might remember that in my last installment of Rumor/Opinion/Mayhem I wrote about Commodore's commitment to the C-128 and what steps I feel they ought to take to instill confidence and loyalty within the ranks of current C-128 owners. I also mentioned that while I am impressed to some extent by the abilities of the Amiga, I have no passion to own one, much less a need for one. Many of you wrote me stating that you were also very happy with your C-128s and were not contemplating another computer purchase in the near future. Some of you also expressed concern that a cost-reduced Amiga might in some way damage sales of the C-128, and possibly lead to early discontinuation of the product. After much thought about that theory, I have come to the conclusion that this is probably not going to be the case.

Contrary to what prophets of overpriced computing at Infoworld believe, the microcomputer market is driven by price, not power. After all, it was the search for a better deal that spawned and nurtured the entire IBM-PC clone market, the success of the Atari ST, and every Commodore product except the Amiga. For this reason it appears to me that the introduction of the new Amiga products will likely spur a huge number of C-128 sales. Sound illogical? Read on.

In retail sales, the classic effect of the introduction of a new product is to drive down the price of those existing products which in comparison might seem, for various reasons, obsolete. This trend appears to be universal with its prime examples lying in the automobile and consumer electronics industries. Consider, for example, what happens on automobile lots across the U.S. and Canada as the "model year" comes to a close and the new cars begin to arrive. As the prices on the "old models" begin to tumble, their sales increase dramatically. This is especially true in the microcomputer industry where new technologies and innovations are a constant driving force behind price reductions.

In fact, the phenomenon of substantial price discounts over a product's history has become such a predictable reality in the computer industry that many consumers make a conscious effort to avoid buying recently introduced products. The reason behind this appear to be twofold: First, a lower price makes the product more affordable for individuals on limited budgets who could not previously justify the expense. And second, over time a product often gains acceptance and becomes more widely recognized and better understood, and as a result becomes a more attractive purchase, especially since it takes a great deal of time for most microcomputers to gain a following with third party vendors and the user community in general.

Given these reasons and observations, it seems a fair assumption that the release of the new Amigas will in all probability force price reductions in both the current Amiga 1000 and the C-128 as well. Evidence of this has already begun to surface in various advertisements I have seen as of late.

But what does a price reduction and in turn increased sales of the C-128 mean to you, the C-128 owner? Well, the short history of the microcomputer industry seems to provide some clues.

In 1984, when Commodore attempted to replace the Commodore 64 with the more powerful, but more expensive and ill-fated Plus/4, many retailers drastically reduced the price of the Commodore 64. The result was an amazing growth in sales for a machine which had at that time a fewer number of commercial titles available for it than the C-128 has now. During that sustained spurt of Commodore 64 sales, many software companies, especially those that had been developing software for the Apple and Atari products of that era decided that the C-64 user base was getting too large to ignore and began adapting and releasing a variety of products for the C-64. And as a result of the commercial hardware and software vendors producing more products for the 64, more people bought the C-64, which in turn has encouraged and continues to encourage even more development for a machine that theoretically and technologically reached the end of its product cycle three years ago. Similar scenarios can be traced for the original Macintoshes, the Apple IIe (in the wake of the ill-fated Apple III), and some of the old Atari 8 bit products.

I am convinced that the prospects of lower hardware prices means that the best is yet to come for our marvelous machines. So in that vein, I welcome the new Amigas to the Commodore product line because I feel their introduction will ultimately strengthen the position our machines have in the marketplace.

Lower prices on C-128 hardware is just one aspect of what I have dubbed "The C-128's Second Wave!". In the coming months we are going to see many new products and applications unveiled which will significantly re-shape the way the C-128 is viewed and used. Since its inception, the focus of C-128 software development has been in productivity applications. As a result, C-128 owners have a large selection of affordable, superior quality word processors, spreadsheets, and database management programs. But as we all know, too much work and not enough play, can breed contempt for using your C-128 every day! Fortunately, a new breed of C-128 software titles are on the horizon which exploit the C-128's graphics and sound capabilities. Having seen and tested advanced copies of many of these products I can assure you that C-128 owners are going to have a lot more choices this summer, and a ton of fun!

The Assembly Line *by John Kress and Loren Lovhaug*

One of the nice features within Basic 7.0 that we have not seen before on a Commodore machine is the WINDOW command. With this command we can define any portion of the active text screen as a window or sub-screen. This acts like a picture frame of sorts, that can be held over the screen. All of the following screen activity takes place within the confines of this framed area. When defined, a window can be cleared, printed to, restored to the original screen size, or redefined in another area of the screen. Windows are handy for directory listings on one half of the screen, while saving the text on the other half. But one drawback with the WINDOW command is that any information that is within the border of that defined window, will be lost if the window is cleared or over-written. If your screen has vital information on it, much care in programming must be taken to prevent a window command from destroying it.

This brings us to the subject of this month's column, Non-Destructive Windows on the screen: the ability to define a window anywhere on the screen, without destroying information that might be overwritten. The method used here is for the 80 column screen, but a version for the 40 column screen would follow the same basic principles. The assembly listing shown herein, and in future columns, has been produced by the Merlin 128 Assembler Editor Package, and can be easily altered to fit the syntax of other assemblers. The program as listed resides in the RS-232 buffer area, a rather safe place for a short machine language program. The proper use of these routines would be to define your window from Basic, without clearing the contents of the window. Make sure that you do not flag the window command to clear out the confines of the window with ",1" at the end of the command. If you do, the routine below will not work, and the data within the window frame will be lost. Don't worry about clearing out the window, as the routines will take care of that. There are two main parts to the program, the first is labeled OUT, located at $B08, the second labeled IN located at $BD2.

The following is a brief description of the program and what each major operation accomplishes. The two routines, IN and OUT, are basically identical, the difference being the direction of the data transfer. IN restores the data to the window and OUT saves the data from the window. Since we are working on the 80 column screen, we will use a 8563 programming feature called Block Copy, which has the ability to copy a block of data from 1 to 255 bytes long. Being a built-in feature of the VDC chip, this method is quite fast; much quicker than moving the same number of bytes on a one by one basis. The destination for the saved data will be in the VDC's ram, a very safe haven in the C-128. First, a zero page pointer is established to hold the address of the targeted save area in the VDC's ram. Because the screen information consists of both character (text) and attribute (color) information we must copy both sets of information. The target area is then $1000 for the character information and $1800 for the attributes. Then the window's borders are read from their zero page memory locations and the number of lines and columns are computed. After subtracting the bottom row from the top row the count is corrected with "Inc Numlin" and "Inc Worklin", the same for the column count. Next, the cursor position, which is now at the home position within the window, is read and stored. This is the starting point from which the data will be saved. Using the number of columns and the row count for the window we can then begin the transfer process. But first we will check to see which set of information to transfer; do we want the character or attribute data? The function of PASS is that of a counter which holds the value or count of the number of times through the routine. Initially PASS contains zero, and is incremented after the first set of screen information has been transferred. Upon completion of the first pass through, this value is checked and the zero page pointer for the destination is changed to reflect the attribute storage area ($1800 in VDC ram).

The actual transferring of data begins at the label FIRST. This is where the block copy transfer routine is used, as mentioned earlier. To set up the block copy routine the following steps must be used: 1) Load update registers 18/$12 and 19/$13 with the target address of the fill operation. 2) Set the copy bit (bit 7) of the Vertical Smooth Scroll register 24/$18. 3) Load registers 32/$20 and 33/$21 with the source address of the data to be copied. 4) Load the Word Count Register, 30/$1E, with the number of bytes, minus one, that are to be copied. The program, at this point, has transferred the first row of data from the window. The pointers for the next line are then adjusted, the number of remaining rows gets decremented and the process continues until all of the character data is safely tucked away. At this point the program checks the value of PASS to see if the attributes have been transferred. If not, the program jumps back and does that portion of the save; otherwise, PASS is restored to zero and the routine returns to Basic. At this point you may do anything within the window and restore the prior data by using the IN routine.

Some notes about the IN routine: when both the character data and the attribute data have been restored to the windowed area, the routine will leave the cursor in the home position and the window will be reset to the full screen size. This is done by printing two HOME characters at the end of the routine. Any printing to the screen will therefore take place at this point unless you program the cursor to a different location or define a new window. Also, it is not required that you restore data to a window before you define another window and save that data, but the prior window's data will be overwritten if this is done. To use the recoverable windows program simply type in the object file creator program below and run it. It will create a binary file called window.o which contains the routines described above. To see the program in action type in the demo program. For further study, the source code is also provided.

The Assembly Line Continued

```
10 for i= 0 to 22
20 read a$:b=0
30 for j=1 to len(a$) step 3
40 b=b+1
50 d$=mid$(a$,j,2)
60 poke 2823+(i*16)+b,dec(d$):t=t+dec(d$)
70 next j
80 next i:if t<>38231 then print "data error - check your typing!":end
90 print"Press any key to save window.o":getkey a$:bsave"window.o",b0,p2824 to p3207 on u9
100 data "a9 10 85 fc a9 00 85 fb a5 e4 e5 e5 8d 00 0b 8d"
110 data "01 0b ee 00 0b ee 01 0b a5 e7 e5 e6 8d 02 0b 8d"
120 data "03 0b ee 02 0b ee 03 0b a2 0e 20 da cd 8d 04 0b"
130 data "8d 05 0b e8 20 da cd 8d 06 0b 8d 07 0b ad d1 0b"
140 data "f0 08 a9 18 85 fc a9 00 85 fb a2 12 a5 fc 20 cc"
150 data "cd e8 a5 fb 20 cc cd a2 18 20 da cd 09 80 20 cc"
160 data "cd a2 20 ad 05 0b 20 cc cd e8 ad 07 0b 20 cc cd"
170 data "a2 1e ad 03 0b 20 cc cd 18 ad 03 0b 65 fb 90 02"
180 data "e6 fc 85 fb 18 ad 07 0b 69 50 90 03 ee 05 0b 8d"
190 data "07 0b ce 01 0b ad 01 0b c9 ff d0 ae ad d1 0b c9"
200 data "01 f0 20 ad 00 0b 8d 01 0b ad 02 0b 8d 03 0b ad"
210 data "04 0b 09 08 8d 05 0b ad 06 0b 8d 07 0b ee d1 0b"
220 data "4c 45 0b a9 00 8d d1 0b 60 00 a9 10 85 fc a9 00"
230 data "85 fb ad 04 0b 8d 05 0b ad 06 0b 8d 07 0b ad 00"
240 data "0b 8d 01 0b ad 02 0b 8d 03 0b ad d1 0b f0 08 a9"
250 data "18 85 fc a9 00 85 fb a2 12 ad 05 0b 20 cc cd e8"
260 data "ad 07 0b 20 cc cd a2 18 20 da cd 09 80 20 cc cd"
270 data "a2 20 a5 fc 20 cc cd e8 a5 fb 20 cc cd a2 1e ad"
280 data "03 0b 20 cc cd 18 ad 07 0b 69 50 90 03 ee 05 0b"
290 data "8d 07 0b 18 a5 fb 6d 03 0b 90 02 e6 fc 85 fb ce"
300 data "01 0b ad 01 0b c9 ff d0 ae ad d1 0b c9 01 f0 20"
    data "ad 00 0b 8d 01 0b ad 02 0b 8d 03 0b ad 04 0b 09"
320 data "08 8d 05 0b ad 06 0b 8d 07 0b ee d1 0b 4c f2 0b"
330 data "a9 00 8d d1 0b a9 13 20 d2 ff 20 d2 ff 60"
```

```
10 fast
20 bload"window.o":rem object code for the window save and return
30 scnclr
40 directory
50 sleep1
60 window 0,0,79,6
70 sys dec("b08"):rem transfer out the window data
75 window 0,0,79,6,1
80 print "this is a test of the window";
90 print " save and restore functions."
100 print "to save a window use   sys dec("chr$(34)"b08"chr$(34)")"
110 print "to reset a window use  sys dec("chr$(34)"bd2"chr$(34)")"
120 print chr$(15)"hit a key to continue":getkeya$
130 sys dec("bd2"):rem replace window data
160 sleep1
180 scnclr
200 window25,4,50,20:sys dec("b08")
210 sys 32800,123,45,6
220 print chr$(14)chr$(15)"hit a key to continue."
230 sleep 2:getkey a$
240 window 15,7,41,22:sys dec("b08")
250 scnclr:print"directory"
260 directory:sleep 4:sys dec("bd2")
```

```
 1 SETREG   EQU   $CDCC
 2 READREG  EQU   $CDDA
 3 BSOUT    EQU   $FFD2
 4 SCRPAG   EQU   $A2E
 5 ATTPAG   EQU   $A2F
 6 PNT      EQU   $E0
 7 USER     EQU   $E2
 8 SCBOT    EQU   $E4
 9 SCTOP    EQU   $E5
10 SCLF     EQU   $E6
11 SCRT     EQU   $E7
12 WRITE80  EQU   $CDCA
13 READ80   EQU   $CDD8
14 KEY      EQU   $FF9F
15 GETIN    EQU   $FFE4
16 ***************************
17 * window routine for     *
18 * non-destructive window *
19 ***************************
20 *  local labels equates  *
21 ***************************
22 NUMLIN   EQU   $B00
23 WORKLIN  EQU   $B01
24 NUMCOL   EQU   $B02
25 WRKCOL   EQU   $B03
26 HOMEHI   EQU   $B04
27 WRKHI    EQU   $B05
28 HOMELO   EQU   $B06
29 WRKLO    EQU   $B07
30 ***************************
31          ORG   $B08
32 OUT      LDA   #$10
33          STA   $FC
34          LDA   #00
35          STA   $FB
36          LDA   SCBOT
37          SBC   SCTOP
38          STA   NUMLIN
39          STA   WORKLIN
40          INC   NUMLIN
41          INC   WORKLIN
42          LDA   SCRT
43          SBC   SCLF
44          STA   NUMCOL
45          STA   WRKCOL
46          INC   NUMCOL
47          INC   WRKCOL
48          LDX   #$0E
49          JSR   READREG
50          STA   HOMEHI
51          STA   WRKHI
52          INX
53          JSR   READREG
54          STA   HOMELO
55          STA   WRKLO
56 TRANS    LDA   PASS
57          BEQ   LOOP1
58          LDA   #$18
59          STA   $FC
60          LDA   #$00
61          STA   $FB
62 LOOP1    LDX   #18
63          LDA   $FC
64          JSR   SETREG
65          INX
66          LDA   $FB
67          JSR   SETREG
68          LDX   #24
69          JSR   READREG
70          ORA   #$80
71          JSR   SETREG
72          LDX   #32
73          LDA   WRKHI
74          JSR   SETREG
75          INX
76          LDA   WRKLO
77          JSR   SETREG
78          LDX   #30
79          LDA   WRKCOL
80          JSR   SETREG
81          CLC
82          LDA   WRKCOL
83          ADC   $FB
84          BCC   NOCAR
85          INC   $FC
86 NOCAR    STA   $FB
87          CLC
88          LDA   WRKLO
89          ADC   #80
90          BCC   NOCAR1
91          INC   WRKHI
92 NOCAR1   STA   WRKLO
93          DEC   WORKLIN
94          LDA   WORKLIN
95          CMP   #$FF
96          BNE   LOOP1
97          LDA   PASS
98          CMP   #$01
99          BEQ   DONE
100         LDA   NUMLIN
101         STA   WORKLIN
102         LDA   NUMCOL
103         STA   WRKCOL
104         LDA   HOMEHI
105         ORA   #$08
106         STA   WRKHI
107         LDA   HOMELO
108         STA   WRKLO
109         INC   PASS
110         JMP   TRANS
111 DONE    LDA   #00
112         STA   PASS
113         RTS
114 PASS    HEX   00
115 ***************************
116 *transfer back to window *
117 ***************************
118 IN      LDA   #$10
119         STA   $FC
120         LDA   #$00
121         STA   $FB
122         LDA   HOMEHI
123         STA   WRKHI
124         LDA   HOMELO
125         STA   WRKLO
126         LDA   NUMLIN
127         STA   WORKLIN
128         LDA   NUMCOL
129         STA   WRKCOL
130 TRNSIN  LDA   PASS
131         BEQ   LOOP2
132         LDA   #$18
133         STA   $FC
134         LDA   #$00
135         STA   $FB
136 LOOP2   LDX   #18
137         LDA   WRKHI
138         JSR   SETREG
139         INX
140         LDA   WRKLO
141         JSR   SETREG
142         LDX   #24
143         JSR   READREG
144         ORA   #$80
145         JSR   SETREG
146         LDX   #32
147         LDA   $FC
148         JSR   SETREG
149         INX
150         LDA   $FB
151         JSR   SETREG
152         LDX   #30
153         LDA   WRKCOL
154         JSR   SETREG
155         CLC
156         LDA   WRKLO
157         ADC   #80
158         BCC   NOCAR2
159         INC   WRKHI
160 NOCAR2  STA   WRKLO
161         CLC
162         LDA   $FB
163         ADC   WRKCOL
164         BCC   NOCAR3
165         INC   $FC
166 NOCAR3  STA   $FB
167         DEC   WORKLIN
168         LDA   WORKLIN
169         CMP   #$FF
170         BNE   LOOP2
171         LDA   PASS
172         CMP   #$01
173         BEQ   ALLIN
174         LDA   NUMLIN
175         STA   WORKLIN
176         LDA   NUMCOL
177         STA   WRKCOL
178         LDA   HOMEHI
179         ORA   #$08
180         STA   WRKHI
181         LDA   HOMELO
182         STA   WRKLO
183         INC   PASS
184         JMP   TRNSIN
185 ALLIN   LDA   #00
186         STA   PASS
187         LDA   #$13
188         JSR   BSOUT
189         JSR   BSOUT
190         RTS
```

BASIC 7.0 Internals Review *by Miklos Garamszeghy*

When I was in Austria last fall, I came across a German language book which featured a disassembly of the C-128's Basic Roms. Although I drooled over the prospect of having such a thing in English, this book was of no use to me since I do not read German. However, I patiently waited hoping that someone would produce a translation or, better yet, the equivalent work in English.

Well, my wait is finally over. And I can say without a doubt that it was well worth it. (My only regret is that I had to wait so long!) The book I am referring to now is Basic 7.0 Internals, from Abacus. This is the latest in the series of books from Abacus dealing specifically with the operation of the C-128. All of the books in the series have been a worthy addition to a C-128 library, and this one is no exception. (Let's hope that the good folks at Abacus continue with their support of the C-128 line when everyone else seems to be jumping ship to get on the Big Blue Bandwagon.)

This particular volume picks up the thread tantalizingly dangled by the earlier "C-128 Internals" book. While the previous volume dealt with the KERNAL routines, this new one deals with the other part of the C-128 Roms: the Basic 7.0 code. The Basic 7.0 Rom set comprises 32 k bytes of addressable space from $4000 to $bfff in BANK 15, of which about 28 k from $4000 to $afff is actually used.

Internals is a hacker's delight. More than 80% of its substantial 600 and some odd pages is devoted to an annotated disassembly of the Basic 7.0 Roms. Virtually every step of every Basic Rom routine is painstakingly explained. (Unlike Compute's Mapping the C-128, Internals gives you the actual disassembled source code for each of the Rom routines. While this takes up a great deal of space, it lends itself to very detailed descriptions of how the code itself works.) Although the book is essentially about Basic, it is really aimed at machine language programmers: it shows you how the Basic routines actually work in the machine.

Internals is just the thing for people such as myself, who enjoy delving into the inner workings of a computer. In the short time that I have had my copy of Internals, I have found numerous highly specialized routines (such as floating point math routines, ASCII to floating point conversion, integer to floating point, etc) which I had tried to implement previously on my own (with varying degrees of success) that already exist in Rom. Using these pre-programmed subroutines in your programs can save quite a bit in code space (after all, you have about 28k bytes of Basic Rom code to play with which contains virtually everything from soup to nuts and can be accessed with a simple JSR statement.), not to mention eliminating the frustrations of trying to re-invent the wheel.

As with most of the previous ABACUS works, the book is technically first rate and well written with no noticeable errors. (I did not check every byte of the disassembly, but several random routines which I did check matched the listings.) With a book of this type, this is to be expected because it is based on a disassembly produced directly from the Roms. A few of the explanations of routines seemed to miss the point a bit, but the annotations are, for the most part, reasonably concise and easy to follow for the intermediate to experienced machine language programmer. As mentioned above, I was able to find and use quite a few specialized Rom routines in my own machine language programs.

Just knowing where to look to find a specific routine is very helpful in most cases. In this respect, Internals is well indexed and cross-referenced. One Appendix gives a cross reference between the Rom locations for routines in the C-64 and the C-128. This is quite handy for programmers who cut their teeth on the C-64 (Although I did not go this route, I used to have a VIC-20, I still found this part interesting because it gives you insight into how similar functions have evolved over the generations. It may also lead to a recognition of which old bugs are still present in the C-128.) or those who may occasionally wish to write for the C-64 as well as the C-128 (supporting the lowest common denominator). Another Appendix gives entry points for each of the BASIC 7.0 key words in terms of Rom addresses. This is all in addition to the main index which lists the entry points by page in the Rom disassembly.

The preamble to the Rom listings contains a brief, but reasonably comprehensive, description of how variables are stored in memory, how to use the expansion Ram, wedging into Basic and assembly language graphics. The example of wedging into Basic is a neat little routine which implements two new commands: KEY ON and KEY OFF. These commands set the function key definitions to their C-128 defaults (KEY ON) or to the C-64 style CHR$() equivalent values (KEY OFF). While this may seem to be of marginal interest, it does demonstrate the point quite well: you can very easily add your own commands or customize existing ones by proper understanding of how the system works.

Also included are examples of how to recover from system crashes and Basic NEWs. These explanations are intended to help the reader to better understand how the various routines work. I would like to have seen a bit more emphasis on how all of the various ROM routines can be used by themselves in your own programs. As the old song goes, you can't always get what you want but you might get what you need. In this respect, Internals contains what you need most: the annotated disassembly. How to use it will follow with experience.

BASIC 7.0 Internals Review Continued

An interesting feature, which I have not seen anywhere else, is a complete listing of all possible 256 values for the MMU configuration register. (While this list is simple enough to compile for yourself if you go through all of the possible bit patterns for the MMU configuration register, lazy people like myself prefer to leave this tedious work to others.) The BANK command allows just 16 combinations, but many more are possible and are even more practical than the default ones in some cases. For example, storing a value of decimal 14 in the MMU will give you BANK 0 Ram from 0 to $bfff, the KERNAL ROM and the I/O block. This allows you a huge buffer space in BANK 0 which is directly accessible for I/O operations without having to switch BANK 15 in and out all the time. If you put machine code in BANK 0 above $4000 and wish to use the KERNAL or I/O block, some very sophisticated coding is required. However, if you set the MMU to 14, everything up to $bfff can be used with the KERNAL still visible. For very long machine code programs, this is almost a must.

While reading Basic 7.0 Internals might not help you to program in Basic any better, it certainly will give you a great deal of insight into how your machine works. This can lead to better programming, indirectly. It can also help you to understand when things don't go quite the way that you had intended or even expected them to go. Remember, despite all the recent hype about artificial intelligence, computers are quite stupid: they will do exactly what you tell them to do, which is not always what you want them to do.

Basic 7.0 Internals is especially useful for novice to intermediate machine language programmers, who might wish to see how certain things can be done, and for all levels of machine language programmers who would rather not re-invent the wheel if they did not have to. Who knows, you might find a whole new way of doing things which you hadn't thought of before. My compliments to the authors for a job well done!

C-128 & 1571 Update Roms: Installation & Ordering *by Loren Lovhaug*

After months of unexplainable delays the C-128 and 1571 upgrade Roms are now available. There are two methods by which you may obtain the upgrades. First, you can order them directly from Commodore by writing Commodore Business Machines, 1200 Wilson Drive, West Chester PA 19380, Attn: Service C-128 & 1571 Rom upgrades) and enclosing $21.65 for the C-128 Roms (a set of three) and/or $6.65 for the 1571 upgrade Rom plus $3.00 for shipping (if you order both the C-128 & 1571 Roms at once you need only pay the shipping charge once) and install them yourself, a fairly painless procedure outlined below. The other alternative is ordering them from your local Commodore service center and have them install them for you. Be forewarned that the charge for installation may vary greatly from dealer to dealer.

In general the C-128 Roms correct a few nagging bugs in the operating system such as the infamous "CAPS Q" bug and the dysfunction of negative relative coordinates when using Basic 7.0s graphics. For many people the changes that the C-128 upgrade Roms provide will not even be noticed, this is especially the case if you do not do any programming. This is not the case with the 1571 upgrade Rom. The 1571 Rom greatly enhances the performance of your disk drive, especially when reading single sided, and foreign CP/M formats. Perhaps the most significant improvement in performance is the accessing of data on the second side of the diskette, especially when writing sequential files. Also Superbase 128 users will be happy to hear that this upgrade corrects the problems which occasionally occur with large databases that occupy space on the back side of the diskette.

Installation of the upgrade ROMS:
Note: As with any user hardware modification to your computer equipment there is potential risk of damage to your equipment or injury to yourself. Please read the following instructions and disclaimer carefully, and if you do not feel confident in your abilities to perform this enhancement to your equipment please take them to someone else who does, this will save much frustration and unhappiness.

Disclaimer: Neither Twin Cities 128, Commodore Business Machines, nor the writer of this article shall be liable, or responsible for any damages or injuries which might occur as a result of attempting the modifications outlined below.

C-128 upgrade Rom installation instructions:

1. Unplug your computer and detach all cables and peripherals from it.

2. Take your computer to a work area free from clutter and away from anything which might cause emission of static electricity (in particular carpets).

3. Turn your computer upside down and remove the six screws holding the two pieces of the plastic case together with a phillips screw driver. (Note: This action will invalidate the warranty of your computer. Those computers that are still under warranty are entitled to this modification free of charge from a Commodore authorized service center.)

C-128 & 1571 Update Roms: Installation & Ordering Continued

Once the screws have been removed, gently separate the plastic case by pushing the top section forward and lifting it slowly upward. The top section cannot be removed entirely at this point because it is still attached in three places. First hold the top section up at an angle with the keyboard facing your body and unplug the power light connector on the left hand side of the computer beneath the ESC key. Note that when the connector is plugged in, the red wire is closest to the keyboard. Then locate the ground attachment screw beneath the ENTER key on the C-128's keyboard and remove it with your phillips screw driver. Now you can lay the top section of the plastic case upside down and to the right of the bottom section. Note that the top section is still attached to the bottom section by the keyboard connector (the one with all the multicolor wires going from the keyboard. You need not unplug the keyboard connector, but you should be careful not to pull on the wires excessively or bend the keyboard connector pins.

5. Proceed to remove the six remaining screws on the outside edges of the silver RF shield and the one larger screw on the top of RF shield. The RF shield will still be fastened to the motherboard by several metal "twists" and a solder point on the right side of the computer. Gently push back all of the metal twists on the edges of the RF shield. Now carefully lift the RF shield up on the left side, revealing the left side of the mother board. You can prop the RF shield up with a ruler or simply let it rest against your hand as you work, or if you prefer you can desolder the RF shield on the right side of the keyboard.

6. On the left side of the motherboard with the computer facing you, beneath the keyboard lie two rows of small random access memory chips; farther back, about midway between the front edge and the back edge of your computer lie the three socketed Roms we are going to replace. Perhaps the best landmark in this area of the motherboard is the "infamous" empty Rom socket. The three chips you are going to replace lie in an inverted L pattern just to the right of and below the empty socket. These chips are labeled 318018-02 (just below the empty socket) position U33, 318019-02 (just below and to the right of the empty socket) position U34, and 318020-03 (just to the right of the empty socket) position U35. Once you have located these chips, with a small flathead screw driver, or a chip tool gently begin removing the first Rom. If you use a small flathead screw driver be sure not to pry up the socket as you work both the front and back of the chip and be careful not to scratch or damage the motherboard. Once you have removed the Rom, find the chip in your upgrade set which has a serial number which corresponds to the chip you have just removed. Then gently, being careful not to bend the pins, place the upgrade chip with the notch facing the front of the computer on top of the socket, taking care that each pin matches its proper socket hole, and evenly press the chip down into place. As you begin pressing, make sure that each pin is sliding into place. You may need to gently bend some the pins inward to make sure they seat themselves properly into their appropriate holes. If you should accidently bend the pins too far in either direction, slowly and carefully bend them back into position. Keep in mind, that these pins are liable to break rendering your chip useless if you bend them too much or with too much force. Once you have pressed the chip down as far as it will go, continue with the next chip until all three are pressed firmly into place.

7. Once the chips have all been replaced, set the RF shield back into place but do not put any screws back just yet. Instead, plug the power light connector back into place making sure the red wire is the one nearest to the front of the computer and plug your computer's power cable into place. Then connect either your forty column or your eight column video cable into your computer and your monitor and turn on the computer. If everything is okay, the familiar copyright message should appear with only one noticeable difference, the year in Commodore's copyright will be 1986 instead of 1985. The type a short Basic program and run it. If everything is okay, you may proceed with putting your computer back together after you unplug it. If something is wrong, immediately turn off the power, unplug the cables, lift off the power light connector and make sure you have placed the upgrade Roms into their proper sockets. In addition, make sure you have pressed all of the Roms down all the way into their sockets. If problems persist, don't risk ruining your computer, take it immediately to a service center to have it and the upgrade ROMs you purchased checked out.

1571 upgrade ROM installation instructions:

1. Unplug your disk drive and detach all cables and peripherals from it.

2. Turn the drive upside down and remove the four screws holding the top part of the plastic case on. Once you have the removed the screws, turn the drive right-side up and lay it horizontally up so that the front plate is on your left. Then remove the top plate by gentle lifting it up and slightly to the left.

3. Once you are inside of the drive, on your left will be the actual drive mechanism itself, and on the right will be a metal box with many holes in it. Beneath that box is the motherboard and the chip you are going to replace. The box itself houses the disk drive's power supply. To the left of the box, on the side nearest to you, you should see two silver cables running from the box running to a screw on the metal base. These are the ground cables. Remove

C-128 & 1571 Update Roms: Installation & Ordering Continued

that screw. On the opposite side of the drive you will see a variety of connectors connected to pins on the motherboard. Locate the connector which runs from the power supply housing to the motherboard (it has four wires) and carefully lift it off its pins, being careful not to bend the pins. Remember that the red wire should be the closest one to you, when you reconnect it later. Once the connector has been undone, proceed to remove the four screws which mount the power supply housing to the base of the unit and lift the housing out of the disk drive.

4. Beneath where the power supply housing was located you will see a variety of chips. The one you need to replace is the second one down on the right-hand side farthest from you marked 310654-03 (see diagram below). Remove the chip from the socket taking note of the same precautions listed above in the C-128 section, and replace it with the upgrade Rom with the corresponding number. Then set the power supply housing back into the drive on its posts, but don't put the screws back in yet. Reconnect the connector that runs from the power supply making sure the red wire is nearest to you. Then plug in your drive and cable it to your C-128. Place a disk in the drive, turn it on, and attempt to get a directory. If everything is okay, unplug the drive, and proceed to put your 1571 back together. If something is wrong, unplug the drive, make sure you put the connector that runs from the power supply housing has been reconnected correctly, that you have placed the chip into the socket properly (facing the right direction i.e. notch toward the front of the drive and pins in their correct socket holes) and that the chip has been pushed down all the way. If problems persist, don't risk ruining your drive, take it immediately to a service center to have it and the upgrade Rom you purchased checked out.

And that's it. Special thanks to Fred Bowen and the rest of the gang at Commodore Engineering for sticking with us by fixing our firmware and lobbying to make sure this upgrade was released to the public at a reasonable cost!

1581 Fun *by Loren Lovhaug*

For a long time, the most common complaints heard about any Commodore computing system have centered around their mass storage devices. After all, it was generally true that Commodore disk drives were slower and more expensive, and at times more "buggy" than their counterparts on other microcomputers. The grammarians reading this article will notice that the previous sentence was in the past tense. That is because Commodore's brand new 1581 3.5 inch floppy disk drive is fast, convenient, and above all an excellent value!

Regular readers of Twin Cities 128 may remember that in issue #13 of Twin Cities 128, I provided you with a brief preview of the 1581 based upon two preliminary units that Commodore sent to us for testing. My preview article in issue #13 was rather skimpy because I made a point of presenting only statistical information that was not subject to change as the prototypes were revised. Since that time we have received a unit with the final version of the 1581's operating system along with the 1581's Demo/Utilities disk. So I thought I would take this opportunity to make a more complete and comparative report on this fantastic new C-128 peripheral.

To begin, examine the following comparison chart. I think you will agree the 1581's statistics are impressive:

Speed Statistics (in seconds)	1581	1571	1541
Small PRG DLOAD (10 blocks)	1.20	1.20	7.22
Small PRG DSAVE (10 blocks)	5.04	7.44	8.28
Large PRG DLOAD (148 blocks)	6.60	16.68	94.20
Large PRG DSAVE (148 blocks)	29.50	68.52	96.40
Read SEQ file (11 blocks)	3.14	3.28	7.74
Write SEQ file (11 blocks)	5.22	.9.24	9.90
Read SEQ file (102 blocks)	17.76	24.50	68.20
Write SEQ file (102 blocks)	23.50	55.30	73.10
Read 100 records in REL file	7.50	37.20	56.20
Write 100 records to REL file	10.90	74.90	102.10

Storage capacity statistics	1581	1571	1541
Storage capacity (bytes)	808960	349696	174848
Blocks free after formatting	3160	1328	664
Files allowed in directory	296	144	144
Largest relative file (bytes)	801480	167132	167132

Notes:
1. All tests were conducted in FAST mode using either direct mode commands and the following Basic 7.0 programs:

1581 Fun Continued

SEQUENTIAL FILE TESTING

```
10 A$="WE WILL READ/WRITE THIS STRING TO DISK TO
TEST THE SPEED OF THE 1541, 1571, AND 1581 DISK
DRIVES WHEN USING SEQUENTIAL FILES...."
20 DOPEN#1,"TEST",W
30 FOR I= 1 TO 20:REM 1 TO 200 FOR LARGE FILES
40 PRINT#1,A$
50 NEXT I
60 DCLOSE
10 DOPEN#1,"TEST"
20 FOR I= 1 TO 20:REM 1 TO 200 FOR LARGE FILES
30 INPUT#1,A$
40 NEXT I
50 DCLOSE
```

RELATIVE FILES
(The actual creation of the relative files with 200 records in them was done prior to the execution of these programs which accesses 100 random records)

```
10 DEF FNR(X)=INT(RND(27)*X)+1
20 DOPEN#1,"TEST",L20
30 FOR I= 1 TO 100
40 RECORD#1,FNR(200)
50 PRINT#1,"WRITING TO RECORD"
60 NEXT I
70 DCLOSE
```

For the read test for relative files, line 50 in the above write test was changed to: 50 INPUT#1,A$

2. All tests on the 1571 disk drive were conducted on a 1571 with the new 1571 upgrade Rom installed, thereby improving the 1571's performance in some catagories.
3. The times are averages. Each test was conducted five times to compensate for discrepancies due to manual timing.

It is obvious that the 1581 is quite an improvement storagewise over both the 1541 and 1571 disk drives. But all the storage in the world will not do you much good if it takes forever to read and write your data. Fortunately the 1581 does not suffer this fate. By studying the table carefully you notice that the area where the 1581 real shines above the 1571 and 1541 is in writing data disk and relative file accessing. The main reason for this lies in the fact that the 1581 has an 8K read/write buffer, a full 4 times the size of that on the 1571 and 1541. This means that the 1581 when writing data does fewer physical "seeks" and thereby greatly reduces the time it takes to actually transfer the data from the computer to the disk.

Another nifty feature of the 1581 is its support of partitions and subdirectories. Partitions can be thought of as a way to physically "wall off" a section of a disk so DOS will not write over that section under normal operation. Such walls are useful for a variety of purposes including the hiding of crucial data from users that might accidentally destroy it, copy protection schemes, and allowing of multiple disk formats to reside on the same diskette. Most 1581 users however will probably use these partitions to reserve an area of the disk for one or more "subdirectories". Subdirectories are created by simply "formatting" a partitioned area (once it has been selected) so that it now has its own directory and block allocation map. Once a partition and corresponding subdirectory has been created they are listed in the main directory as a CBM file. To select a partition and subdirectory simply enter OPEN 15,8,15,"/0:partition" from within a program or immediate mode. Therefore all of the commercial packages which allow you to directly send a DOS command, have no problem selecting a partition using the above command string. To return to the root (main) directory from a subdirectory simple issue a DCLEAR (OPEN 15,8,15,"I0").

Although partitions and subdirectories are a relatively easy concept to understand, their creation involves some fairly complicated rules. For instance, you must be careful when selecting a place on the diskette for a partition to "live" so do not overwrite your root (main) directory information which resides on track 40. In addition, to have a subdirectory on your partition your partition must: 1. Start on sector 0 of a specific disk track, 2. The partition size must be a minimum of 120 blocks in length, and 3. The partition's block size must be a multiple of 40.

Fortunately, the software engineers at Commodore have provided users of the 1581 with a set of nifty tools on the 1581 Test/Utilities disk which make life a lot easier. The Partition Aid program is a menu driven utility written which masks from beginners the complex series of DOS direct access commands necessary to create partitions while providing the advanced user with a demonstration of how to create and utilize partitions under custom software control. Although the Partition Aid program automates the partition creation process, it does not automate the actual choosing of a physical location on your disk, which can be worrisome when trying to create partitions on disks that already have data stored on them. Fortunately also included on the 1581 Test/Utilities disk is a SHOW BAM utility and a Sector Editor to assist you. In addition to the above mentioned utilities there is the usual "How to Use" program and two file copies (one which supports the 1750/1700 Ram expander for faster copying) for transferring files between disks as well as a full disk copier. There is also an AUTOBOOT file maker, and several demonstrations of bit mode programming in Basic and assembly language, and a program that compresses bitmaps to maximize the speed at which they can be loaded from disk.

1581 Fun Continued

In conclusion, I must tell you I love the 1581. There is no doubt in my mind that it is the finest disk drive that Commodore has ever produced, and I can't wait till they are commercially available so that I can buy a couple more. For a long time I have considered the acquisition of a hard drive for our database needs here at Twin Cities 128, but now I am convinced that given the extremely high cost of hard drives for the C-128, the relative slow speeds of the hard drive currently available, and the flexibility the 1581 provides because it is a non-fixed disk system. After all, fill up 10 megabytes on your 10 megabyte hard disk and you have got to make some tough decisions, fill up 10 megabytes with your 1581, and the only decision you have to make is whether to buy five or ten more 3.5 inch floppies. Besides, it is rather difficult to put a ten megabyte hard disk in your shirt pocket! I think that the 1581 is an unbeatable choice for C-128 owners with larger database needs, and speed requirements.

More Superbase Programming *by Loren Lovhaug*

In our last issue I explained how to use Superbase 128 to create mailing labels from a database given a specific criterion. In this installment of our programming Superbase series, I will attempt to illustrate how to extract information from multiple database files to speed the entry of redundant information when creating new records as well as demonstrating the use of result fields and perform strings.

Let us imagine the following scenario: suppose you are the owner of a small company which supplies custom made furnishings to building contractors. Your goal is to set up an automated order entry and billing system with your C-128. The idea is to maximize efficiency especially for orders received over the phone. Ideally your order entry scheme might proceed something like this:

1. Customer calls to place an order.

2. If the customer is new, you or someone in your staff enters their name, address, telephone and other pertinent information into a customer file. Upon entry into the customer file, the customer will also be assigned an "account number" (I usually use a person's telephone number since it is unique) to be used as the key field in our database. The customer file will be used to retrieve this information in case you need to contact the customer for some special purpose later, as well as when you actually process the order, send billing statements and promotional materials. If the customer has previously placed an order, the information will already be in the customer file and the order entry system will automatically retrieve it and append it into the order when the person taking the order enters the client's account number.

3. The person taking the order then adds the items the customer wishes to purchase to the order. To accomplish this task, the order taker simply types in an identification code for the product and the order entry system automatically retrieves the description and the unit price of the item from your companies product file and appends that information to your order as well.

4. The customer gives the order taker the instructions for shipping and/or delivery and that information is entered into the order as well. The order taker then verifies the information, checking to make sure everything is correct.

In the above scheme we are actually utilizing three separate database files to achieve our goal. The first file, the customers file holds all the specific information about your company's customers. The second file, the products file stores specific information about our products, such as descriptions and prices. The last file, is the orders file, this file is where all of the specific information about any individual order is stored. Here are the individual files listed with the fields they might incorporate:

```
Customers file:          Products file:          Orders File:

[Account#]               [ProuductID]            [Invoice#] [Date]
[Name]                   [Description]           [Account#]
[Address]                [Unit-price]
[City] [State] [Zip]                             [Name]
[Phone]                                          [Address]
[Misc]                                           [City]
                                                 [State] [Zip] [Phone]
```

More Superbase Programming Continued

```
[ID1] [Des1] [Unit-price1] [quant1] [sub1]
[ID2] [Des2] [Unit-price2] [quant2] [sub2]
[ID3] [Des3] [Unit-price3] [quant3] [sub3]
[ID4] [Des4] [Unit-price4] [quant4] [sub4]
[ID5] [Des5] [Unit-price5] [quant5] [sub5]
[ID6] [Des6] [Unit-price6] [quant6] [sub6]
[ID7] [Des7] [Unit-price7] [quant7] [sub7]
[ID8] [Des8] [Unit-price8] [quant8] [sub8]
[ID9] [Des9] [Unit-price9] [quant9] [sub9]

[Subtotal]
[Tax]
[Shipping]
[Total]

[ShipInstructions]
```

In addition to the use of multiple files to solve this database management problem we are also going to make Superbase automatically do some calculations within the ORDERS file for us. This is done through the use of result fields. Result fields are simply numeric data fields which are filled by results of calculations involving other numeric fields in your record layout. When the numeric fields in a specific calculation have been filled, the result field automatically is filled with the result from the specified calculation. Here are the following fields that would be defined as result fields along with their corresponding calculations and purposes:

Result Field	Calculation	Purpose
[sub1]	[Unit-price1]*[quant1]	Subtotal #1
[sub2]	[Unit-price2]*[quant2]	Subtotal #2
[sub3]	[Unit-price3]*[quant3]	Subtotal #3
[sub4]	[Unit-price4]*[quant4]	Subtotal #4
[sub5]	[Unit-price5]*[quant5]	Subtotal #5
[sub6]	[Unit-price6]*[quant6]	Subtotal #6
[sub7]	[Unit-price7]*[quant7]	Subtotal #7
[sub8]	[Unit-price8]*[quant8]	Subtotal #8
[sub9]	[Unit-price9]*[quant9]	Subtotal #9
[Subtotal]	[sub1]+[sub2]+[sub3]+ [sub4]+[sub5]+[sub6]+ [sub7]+[sub8]+[sub9]	Total of the subtotals for above items.
[Total]	[Subtotal]+[Tax]+ [Shipping]	Adds in tax and shipping.

Here is the program that accomplishes the task outlined above. Study the listing, taking note of how information from one file is transferred (copied) to another. Beginners take the time to look up the commands and structures in the Superbase manual which are not understood. Remember, this series is not designed to replace the reading of the Superbase manual or the hours of practice it will take to master Superbase programming, instead, it is designed to give the reader insight into accomplishing various task and spur further study and investigation.

```
100 database "accounts":file "customers"                          -Select the database, and customer file
110 ask @10,12"Enter today's date (DDMMYY):";dt$                  -Ask user for date to go on order
120 ask @10,13"Enter customer account number (Enter 0 if new):";an$ -Ask for customer's acct. (phone) number
130 if an$="0"then enter:else select an$                          -If new customer make new entry, or else
                                                                   select the record in the product file
140 ac$=[account#]:n$=[name]:ad$=[address]:ci$=[city]:st$=[state] -Lines 140-150 store customer info in
150 zp$=[zip]:ph$=[account#]                                       variables for copying into orders file
160 ask @10,14"Enter Invoice number:";in$                         -Enter Invoice number for orders file
170 file "orders"                                                 -Select orders file
```

More Superbase Programming Continued

```
180 [account#]=ac$:[name]=n$:[address]=ad$:[city]=ci$:[state]=st$   -Lines 180-190 transfer the contents of
190 [zip]=zp$:[date]=dt$:[invoice#]=in$                              variables to orders file
195 store                                                            -Store the record in its current state
200 select in$:f=0                                                   -Select the record for product entry,
                                                                     the variable f is order field counter
                                                                     i.e. f=3 then [ID3] [Des3]
201 do until f=9                                                     -Take a maximum of nine orders
202 f=f+1                                                            -Increment the order field counter
205 f1$="[ID"+right$(str$(f),1)+"]"                                  -Lines 205-208 set up field names for
206 f2$="[Des"+right$(str$(f),1)+"]"                                 the perform strings. This technique
207 f3$="[Unit"+right$(str$(f),1)+"]"                                allows us to do order entry in a loop
208 f4$="[quant"+right$(str$(f),1)+"]"                               rather than using explicit field names
210 select current                                                   -Display currently entered order info
215 ask"product code ('q' quits):";z$                                -Ask for product code
230 if z$="q"then exit                                               -If the user enters q then exit product
                                                                     entry loop; end entering products
235 file "products"                                                  -Select the product file
240 select z$                                                        -"Look-up" specified product
245 nmat display chr$(147);"No #";z$;"Press a key":wait:goto 210     -If product code entered is not in the
                                                                     product files, clear screen, display
                                                                     message, wait for key press, go back
                                                                     and get another product code.
250 d$=[description]:up=[unit-price]                                 -Store product info in variables for
                                                                     transfer to the orders file
260 file "orders":select in$                                         -Select the orders file and the order
270 c$=f1$+"=z$":perform c$                                          -Lines 270-290 copy product data from
280 c$=f2$+"=d$":perform c$                                          the variables into order using perform
290 c$=f3$+"=up":perform c$                                          strings
300 store:select current                                             -Store the order so far and display it
310 ask "C if you want to change this entry:";zz$                    -Ask if the last product is okay
320 if zz$="c"then 210                                               -If c then change last entry; go to 210
330 ask "Quantity to purchase";q                                     -Ask how many of product to purchase
340 c$=f4$+"=q":perform c$                                           -Store quantity of product into order
350 store:select current:loop                                        -Store the record, display the record,
                                                                     go back and enter another product
360 ask "amount of tax:";[tax]:select current                       -Once we have left product entry loop,
370 ask "shipping fee:";[shipping]:select current                    we are asked to fill other order info
380 ask "ship instructions";[ship]:select current                    and the responses are displayed
390 store                                                            -Store the completed order
400 select current                                                   -Display completed order on the screen
410 ask "'y' if this order is correct";zz$                           -Ask if the order is correct
420 if zz$<>"y"then select replace:goto 400                          -If order is not correct, allow the user
                                                                     to correct it, go back and ask if it is
                                                                     now correct
430 menu                                                             -Return to the superbase menu
```

The technique I used in the above example to transfer (copy) information from one database file to another involved two simple steps: First, select the file with the original data and copy the desired data into dummy variables from the desired fields, and second, select the destination file and copy the data from the dummy variables into the destination fields. In the loop which spans lines 201-350, I enhance this process by using a very powerful, but often overlooked feature of Superbase 128 which allows you to build and execute commands stored in strings. This macro programming ability can be a real time and memory saver. In the above example, I built "perform strings" to access the fields which make up the "products ordered" section of the order file. By using the perform strings I could use a loop to access the product fields essentially by number via variables, in a manner not unlike the application array variables in Basic 7.0, instead of having to reference each specific field by name. Lastly, although this example is fairly efficient it is by no means the ultimate solution, time and space permitting, one ought to incorporate error trapping, enhanced user prompts etc., and in so lies your challenge.

Sparrow's Slick Tips *by Sparrow James*

0111011: Detected Devices From Fred Bowen, Commodore Engineering

I thought it might be fun to drop some programming hints, from time to time, on the C64/128 community. Surely there are a few crusty 8-bit hackers still out there who will appreciate them...

One problem with a great many programs I've seen and used stems from the difficulty of finding out if a serial bus device exists. Until the 128's TRAP statement came along, most programs either hung or crashed with a DEVICE NOT PRESENT error. I hate when that happens! And it's fairly easy to test for that situation and handle it in a dignified manner. Try this:

```
C64's:      10 POKE 144,0          :REM CLEAR ST
            20 POKE 780,unit       :REM LDA unit
            30 SYS 65457           :REM JSR LISTEN
            40 POKE 780,1          :REM LDA SA
            50 SYS 65427           :REM JSR SECOND
            60 IF ST<0 THEN PRINT"DEVICE NOT PRESENT!"
```

This can be collapsed into a one liner, of course, as in the 128 version:

```
10 POKEDEC("90"),0:SYSDEC("FFB1"),unit:SYSDEC("FF93"),1:IF ST<0 THEN...
```

But I find the following the best case - it works on either machine:

```
10 T=ABS(PEEK(65533)=255)     :REM T=0 IF C64,  T=1 IF C128
20 A=780:IF T THEN A=6
30 POKE 144,0: POKE A,unit: SYS65457: POKE A,1: SYS 65427
40 IF ST<0 THEN do whatever you wanna do if unit ain't there
```

This works for disk drives, printers, and such. And it's easily converted to machine code. Just remember to do all this in BANK 15 (IO, ROMs) on the C-128.

111100: Handling Interrupts involving the 8563 From Sparrow James, via Transactor and Fred Bowen

For a long time I have wanted to write an assembly language interrupt driven clock routine that works on the 80 column screen. The clock routine itself is a rather simple exercise in polling the CIA registers, but there a rather significant problem in doing this on the 80 column screen because of the way the 8563 interferes with user created interrupts as an normal part of the screen editors operation. However, in a recent issue of Transactor I found the following little routine that plays traffic cop by simply sniffing the stack after an interrupt, and seeing if the IRQ occurred while the processor was executing instructions from the $C000-$CFFF area, roughly where the screen editor lives.

```
        LDA $107,X
        CMP #$C0
        BCC UIRQ
        CMP #$D0
        BCC HIRQ
UIRQ    JMP 'your interrupt routine'
HIRQ    JMP $FA65
```

Be sure when you do your accessing of the 8563 registers and memory you use the routines at $CDCC and $CDDA.

00111101: DIMming Simple Variables for Usage in Long Programs From Sparrow James

When writing long Basic 7.0 programs, especially those which define and utilize a number of variables, you can greatly increase the execution speed of your programs with a simple technique involving the use of the DIM command. Ordinarily the DIM command is used to declare array variables, however it can be used to declare simple variables as well. By declaring often used variables with the DIM command at the beginning of a program Basic will place them at the top of the variable table and therefore find their definitions much more quickly than if they were casually defined. As an example, suppose you have a long program that involves the use of many numeric variables and several nested loops, by declaring your loop variables in advance (i.e. DIM I,J,K) using the DIM statement you can save a great deal of execution time during iteration of each loop.

Rumor/Opinion/Mayhem *by Loren Lovhaug*

During the last few weeks we have witnessed the downfall of Gary Hart and Jim Bakker. As I watched the scrutiny and destruction of these individuals I mused the following: Being the managing editor of a grass roots, 8 bit computer publication is kind of like running for president or being a televangelist. This is because I, like the aforementioned targets of the fourth estate, have been placed on the defensive by an angry mob, waiting with baited breath to devour me. True, my mob is somewhat tamer than the Miami Herald, The Charolotte Observer, and the Washington Press Corps, but in their own way they are just as relentless in their pursuit. I am talking of course about angry Commodore computerists.

So ladies and gentlemen, I will quit ducking those tough questions and answer my critics. Here is the transcript of a mock press conference I had with myself just the other day.

LJL: Good afternoon ladies and gentlemen. I am glad you could all make it. Looks like you folks are going to get the scoop on this one since CNN, NBC, CBS, and ABC did not bother to show up. Ok, let's get this thing under way, Ms. Ross, you have a question?

PAULINE ROSS (Amigo News Service): Yes, Mr. Lovhaug, we have heard you are a critic of Commodore's new Amiga 500 computer: my question is, given the A-500's $650 price tag, and a performance ratio equal to that of the Amiga 1000, how can you justify the existence of the C-128?

LJL: Well Pauline, let me answer your question by first stating that I am impressed with the Amiga series computers and there is no doubt that they are the most technologically advanced microcomputer on the market today. But, let's take a look at things the way they really are.

Mr. Lovhaug walks over to a C-128 hooked up to a projection video system, and reveals:

Amiga 500 computer		C-128 computer	250
w/ disk drive	650	1571 drive	225
Amiga 1080 monitor	300	1902 monitor	280
Scribble	100	Pocket Writer 2	52
Maxiplan Plus	130	Multiplan 128	30
Superbase Personal	100	Superbase 128	70
Deluxe Paint II	120	GEOS 128	70
512K system plus software	1400	128k system plus software	977
A-501 512K expander w/clock	150	1750 512K REU	150
		1351 mouse	45
1 MB system plus software	1550	640K system plus software	1172

(Note: These prices are fairly conservative; by shopping around and making the right software substitutions one can do better on both systems. But in general, Amiga productivity titles run 20-40% more than their C-128 counterparts.)

Now if you examine this chart closely you will see that in reality the basic Amiga 500 system is not $650, but instead is $1400 when you add the cost of an RGBA monitor and four productivity titles (word processing, spreadsheet, database, and graphics). The equivalent C-128 system, costs roughly $420 less. It should also be noted that many Amiga 1000 owners have found it necessary to expand their systems beyond the 512K of random access memory provided in the Amiga 500, while the vast majority of C-128 owners get a long just fine within the confines of their machine's standard 128K. However, if you expand the C-128 by adding 512K of Ram and the 1351 mouse, you greatly enhance the C-128's speed and abilities at about 25% off the equivalently expanded Amiga 500 system. Now granted, the Amiga system is faster, and more powerful, but I submit that for the needs of the average computer user, especially those on a budget, the C-128 system provides an affordable solution.

JANE MEANHAND (TBBS NEWS): In light of the introduction of the new Amiga products, will you consider adding Amiga coverage to Twin Cities 128?

LJL: No. I believe the single most important factor in the success of Twin Cities 128 has been the fact that we cater exclusively to the C-128. Our readers look to us for information they simply cannot find elsewhere, and they appreciate the fact that our publication is not diluted with coverage of other machines. Besides, there are already plenty of publications for the Amigas.

Rumor/Opinion/Mayhem Continued

BELLI HANLEYSKI (Moscow Computer News): In your publication, I fail to notice anything but positive commentary about the C-128 and Commodore. Don't you feel you are doing your readers a disservice by continuing your pro-128 propaganda without balancing it with some objective criticism of the C-128 and Commodore?

LJL: Belli, that is an interesting assertion, one that I will not entirely deny. Yes, it is true that in a typical issue of TC-128 you will not find much complaining about the C-128. But I submit that there is not a great deal about the C-128 worth griping about, and that our pages are much better spent discussing ways to help our readers make the most out of their computer investment. However, you will find that I have been critical of Commodore for various marketing policies and the lack of any tangible user support.

EL KURTO SWANSO (El Commode): Mr. Lovhaug, over the past few months you have reported on items like the 1581 disk drive, and GEOS 128 which have not until just recently been available to the ordinary user. Don't you think most 128 owners would rather not read about things they can't have?

LJL: So under your reasoning: If it is not yet available, then people are not interested in it? I think people are very interested in new products for the C-128, and I am proud of the fact that Twin Cities 128 is often months ahead of the so-called "leading publications". By reporting on items like GEOS 128, BASIC 8, and the 1581 disk drive well in advance of their availability, I think we help our readers make informed decisions about future computer purchases. Also, I always state in any preview article that it is just that, a preview of an item that is yet to be released.

1581 CP/M Modification *by Miklos Garamszeghy*

Older versions of C-128 CP/M will not fully support the new 1581 disk drive. This is a pity because the large capacity of the 1581 combined with its high speed make it an ideal CP/M drive. Of course, you can mail in the coupon which comes with the 1581 along with some of your hard earned cash and get yet another "upgrade" version of CP/M which does support the 1581. I already have three different versions of CP/M for my C-128 (the original plus two "upgrades") so why do I need another one just to use the 1581? The short answer is that I do not.

The first problem involves creating a 1581 CP/M format disk. This is easiest to do with the C-128 in native mode using the burst mode format command since older versions of CP/M's FORMAT.COM utility do not work properly with the 1581. The following command string can be used, assuming your 1581 is device 9:

```
open15,9,15
print#15,"u0"+chr$(134)+chr$(2)+chr$(79)+chr$(10)+chr$(0)+chr$(229)+chr$(1)
close 15
```

This will format the disk identical to a 1581 DOS disk physical structure (i.e. 512 byte/sector, 10 sectors track) but without the directory, BAM, etc. and fill it with CP/M blank disk bytes ($e5 or dec 229).

Next, with a few simple modifications to the CPM+.SYS file, you can take full advantage of the capacity of the 1581 without forking out for the latest upgrade disk. You can still use the disk formatted by the method above with CP/M if you do not make the following mods, but you will only be able to fill it half full. This mod will not allow you to boot CP/M from the 1581, but it will allow you use the 1581 once CP/M has been booted from a 1571 or 1541. (In my present set up, the 1571 is device 8 and the 1581 is device 9. CP/M can only be booted from device 8 anyway). A note in Commodore's favor is that the CP/M upgrade will allow you to boot from the 1581 if it is connected as device 8.

The procedure involves changing a few bytes in part of the CPM+.SYS file know as the "disk parameter block table". This table, which is described in detail in the "CP/M 3 System Guide" under section 3.3 BIOS Data Structures on pages 40 to 44, contains the data for the physical characteristics of the MFM disk formats supported by C-128 CP/M. The location of the table in the CPM+.SYS file depends on the version of CP/M that you are using. The instructions for all three current versions are outlined below. The locations are summarized in Table 1 and referenced in the text.

Before continuing, you will need a formatted CP/M disk (C-128 1541 single or 1571 double sided format) containing the CP/M system files (CPM+.SYS and CCP.COM) and the utility SID.COM (or the older DDT.COM or equivalent debugger utility). You can also include a copy of SHOW.COM to check the results afterwards. The SID.COM program resides on the CP/M additional utilities disk that comes with the Digital Research CP/M plus documentation. Several other public domain debuggers are also available if you do not have SID.

1581 CP/M Modification Continued

Note: Use a back up work disk. Do not do this with your original system disk because it will make permanent changes to the operating system.

After booting up CP/M, note the version date printed on the screen. This will be used as a reference for the addresses in Table 1. Insert your work disk and type in: sid cpm+.sys <return>
where <return> is the key marked "return" at the right of the keyboard. Note that for clarity, I will use lowercase letters to indicate items that you type in and uppercase for prompts made by the computer.

After a few moments, the following status message will be displayed:

```
CP/M 3 SID - Version x.x
NEXT MSZE  PC  END
zzzz zzzz 0100 CEFF
#
```

where zzzz is a hexadecimal number based on the version date, as listed in Table 1. Jot this down for future reference as it will be needed when saving the changes. The "#" symbol is SID's command prompt.

Now we are ready to make the changes. The physical format of a 1581 disk is identical to that of the EPSON QX-10 (10 sectors per track, 512 bytes per sector, 2 sides). The only difference is in the number of tracks per side (80 for the 1581 compared to 40 for the EPSON). It makes things easier to start with the EPSON parameters and change them a bit rather than create a whole new entry. (The modifications will still permit full read compatibility with the EPSON disks and nearly full write compatibility on the 1571. The incompatibility in writing is only that the operating system may attempt to write more to the EPSON disk than it can hold, thus causing a disk error if you try to write to an almost full disk).

Next type in: dyyyy <return>
where yyyy is taken from Table 1. This is SID's d or display memory command. Note that in this, and all other SID commands, there is no space between the command letter and the command parameters. For example, d yyyy would produce an error prompt.

SID will respond with a display similar to:

```
yyyy: 50 00 04 0F 01 BD 00 7F 00 C0 00 20 00 02 00 02 P..........  ....
yymm: 03 0A 45 70 73 6F 6E 20 51 58 31 30 49 A5 00 81 ..Epson QX10I...
```

plus some more rows of similar hexadecimal numbers followed by the "#" prompt. The next step is to change some of the bytes. The ones that are changed are referred to in Digital Research's documentation as EXM (extent mask - 1 byte), and DSM (total drive storage - 2 byte word). These are the two parameters which tell the system how much data it can store on a disk. The change is done with SID's s or set memory command. Type in: sxxxx <return>

where xxxx is taken from Table 1. SID will respond with: xxxx 01

The 01 is the current value of the byte at this location. Change it to the desired value by typing in 00 followed by return. SID will then prompt for the next byte: xxx1 BD

Type in 86 followed by return. SID will then prompt for the next byte: xxx2 00

Change this by typing in a 01 followed by return. That completes the major changes. At the next prompt: xxx3 7F

Enter a period "." followed by return. This should bring back the main SID prompt "#". If you want to change the disk type name from Epson QX10 to say 1581, use the s command again: snnnn

When SID prompts with: nnnn 45

Type in a quote (") followed by 3 spaces then 1581 and 3 more spaces and a return. At the next prompt, type in a period to return to the main SID prompt. Type in: dyyyy <return> again to check your changes and the following should be displayed:

1581 CP/M Modification Continued

```
xxyy: 50 00 04 0F 00 8b 01 7F 00 C0 00 20 00 02 00 02 P.......... ....
xxmm: 03 0A 20 20 20 31 35 38 31 20 20 20 49 A5 00 81 ..   1581   I...
```

The final step is to save the changes. This is done with SID's w or write command:

wcpm+.sys 0100 zzzz <return>

Once this has been done, your new CP/M system is ready for action. Re-boot the computer (you need to boot the modified CP/M system) and turn on your 1581 as drive b: (device 9). You can now PIP some files to a formatted 1581 disk (as outlined above) and use it at will for all file storage. You can check the capacity of the drive by using the SHOW.COM utility: show b:

will display the amount of space currently available on the 1581. If it does not give something like 778k read write space free with a formatted blank disk, then your changes may have gone awry. Double check the changes outlined above and try again.

The procedure outlined above can also be done using the Ram disk (drive m:) as the working drive if you have a 1750 Ram expander. In this case, you would PIP the required files to the Ram disk, do the modifications and PIP them back again to your work disk. The same method can be used to alter the disk parameter table to support other CP/M disk formats on the 1571, such as Televideo, Xerox, DEC, etc which are not normally supported by C-128 CP/M. In this case, I would suggest that you consult the explanation of the table parameters in the CP/M System Guide. You also need to have a thorough understanding of the disk format that you wish to implement.

Table 1: Summary of CPM+.SYS file addresses

	Values by CP/M Version Date		
Parameter*	1 Aug 85	6 Dec 85	8 Dec 85
zzzz	5d00	6400	6400
yyyy	1400	2161	2161
xxxx	1404	2165	2165
nnnn	1412	2173	2173

Note: Refer to text for explanation of parameters

C-128 Graphics *by Loren Lovhaug*

In the dark times, before the age of affordable computing, I spent the majority of my time in hot, poorly lit gymnasiums throwing an orange spheriod at a ring elevated ten feet above the floor. Those were the days when my dreams had very little to do with microprocessors, random access memory, pixel resolution, or money. But even though my life has drastically changed since then, at times it seems as though I am replaying contests of old in a different, but parallel setting.

Like the 6'2 forward who's competitive basketball career ended in high school, the C-128 will never make it to the "big time" of computer graphics. Neither of them possess the size, speed, and muscle it takes to attain the highest level of achievement. But all is not lost just because you will never find my name on an NBA roster or the C-128 in the credits for a feature film, rock video, or episode of Miami Vice. After all, I still get a charge out of driving past an opponent in pickup games, and recent C-128 software developments allow you to achieve some equally satisfying and at times stunning results graphically with your C-128. In this article I will discuss three 80 column graphics products which can give you the kind of performance you thought you could get only from 16 bit micros.

When we discuss computer graphic programs we should be aware that there are really two kinds of computer graphics software. The first division encompasses those programs which are designed to create images that are to be displayed or animated on some kind of video display. The other division involves programs which are designed to prepare graphics for output to a printed page.

C-128 Graphics Continued

Pagemaking

Over the last two years the hottest buzz words in the personal computing industry has been "desktop publishing". Desktop publishing infers the use of personal computing hardware and software to produce "publication quality" text and graphics. The judgement as to what exactly is publication quality, and what is not, in reality depends upon your audience and the amount of time and money you can afford to expend. The undisputed king of the desktop publishing realm has been and still remains the Apple Macintosh, although the rest of microcomputerdom is catching up quickly. The Macintosh's ability to quickly and easily integrate attractive character sets of various sizes and styles with high quality computer illustrations is now being emulated via software on a variety of computer systems including our own C-128s.

Perhaps the best known of these software "Mac-a-likes" in the Commodore world is GEOS from Berkeley Softworks. As regular readers of these pages know, GEOS was originally marketed for the C-64, but now, Berkeley is now marketing a greatly enhanced version of GEOS for the C-128. We have been fortunate to be a part of the pre-testing process for this product and we have come to rely on it for a great deal of our graphics page make-up needs. Although the C-64 version of GEOS brought us many of the user interfaces and powerful text/graphics integration, it could not deliver the speed and flexibility of the Macintosh due to the memory, speed, and display limitations of the C-64. The C-128 version comes a lot closer to the "near-mac" goal. By utilizing the C-128's 80 column display, the user can easily manipulate text and graphics which span the entire width of the document, whereas on the C-64 version of GEOS one was forced to scroll over the width of a page in rather small horizontal increments. While this difference might appear trivial to the casual user, power users (such as myself) immediately realize the increased pagemaking speed this potentially adds, especially when you use photo (graphics) scraps in conjunction with the photo manager.

Besides the added screen width of GEOS 128, other aspects of its operation are making the Macintosh comparison a valid one. By using a 1750 RAM expansion unit in conjunction with a 1571 or 1581 disk drive you will find that the sluggish nature of GEOS on the C-64 is just a bad memory. In fact, as an illustration of just how much like a Macintosh our little C-128's are becoming, you might take note of the pocketwatch and sparrow illustrations on the front cover of this issue of Twin Cities 128. Both are high quality graphic "scraps" converted from Macpaint files for the Macintosh using a little public domain utility you can download from Genie or Quantum Link. There are also many public domain and commercial conversion utilities for transforming graphics from Commodore 64 programs such as The Print Shop, Printmaster, The Newsroom, Doodle and Koala paint. Through the utilization of conversion programs with GEOS 128 one can have an immense amount of high quality pre-drawn graphics at their disposal.

Although I am quite pleased with GEOS 128, it is too bad that Berkeley chose to package the "standard" version of GEOwrite (albeit with the addition of 80 column screen support) instead of the enhanced version of GEOwrite that will be packaged with Writer's Workshop 128. The enhanced version of GEOwrite (version 2.1) in the Writer's Workshop 128 package truly rounds out GEOS 128 as a pagemaking system, because it supports better margin and spacing control, allows for much more accurate positioning of GEOpaint graphics within your text as well as much greater text editing capabilities. In addition, also included with Writer's Workshop 128 is Text Grabber 128, which makes the importation of text created with other word processors a snap, as well as GEOlaser 128 which allows the output of your text and graphics to an Apple laserwriter should you be fortunate enough to beg, borrow, or steal the use of one.

Another fabulous pagemaking package is Fontmaster 128 by Xetec. This program like GEOS 128, allows the use of multiple fonts as well as graphics integration. Fontmaster's approach to pagemaking is from a word processing standpoint, and as such, incorporates all the usual features of a good word processing program. But unlike most word processors Fontmaster 128 allows the user to integrate Print Shop, Printmaster, and Doodle graphics with text. The graphics integration features of Fontmaster are so strong that they allow text to be formatted around the graphic image, set apart from the image, or even the text to be overlaid on top of the image. Fontmaster also supports multiple column output and comes with a font editor so you can design your own fonts. This font editing ability can also be used to create custom graphic logos or symbols for easy integration into your text. Also, like GEOS 128 with Writer's Workshop you can convert text created with other word processing programs to Fontmaster 128 format.

Video graphics

The other side of computer graphics is screen graphics, or as it is referred to in the television industry, video graphics. Video graphics are by nature more flexible than page graphics, since paper is a static media while video displays are dynamic (i.e. they are constantly being changed or refreshed). However, dealing with video graphics is also more complex. When creating video graphics one might be concerned with items such as color,

C-128 Graphics Continued

color resolution, screen scrolling and animation, and appearance on various monitors and situations.

Since its introduction in mid-1985, Commodore's own Amiga computer has been the undisputed champion of microcomputer video graphics. The reason for the Amiga's dominance in this field has been twofold. First, the Amiga's hardware is made up of several custom VSLI controllers which facilitate very high resolution graphics, very high color resolution, and fluid animation. The other reason lies in the fact that several third party software producers have decided to exploit the Amiga's hardware potential by developing incredibly sophisticated video graphics applications. But as powerful and sophisticated as the Amiga's video graphics abilities are, its little brother in the Commodore line, our own C-128 is now making great strides in the Amiga's direction. This is not to suggest that the C-128 has any chance of eclipsing the Amiga's supremacy in the video graphics field, but recent C-128 hardware and especially software developments in this area will certainly have the potential to shock many who don't believe our eight-bits can "cut it".

The software development which has revolutionized C-128 video graphics is Basic 8 by PATECH software, an incredible three dimensional graphics programming language which pushes the C-128's graphics abilities far beyond what most people believed was possible. The hardware development is the increasing acceptance of the (until recently esoteric) addition of 80 column video Ram to your C-128, and the advent of the C-128D which comes with a standard complement of 64K VDC Ram. This procedure involves the replacement of the C-128's 16k VDC Rams with 64K Rams and is detailed in issue #9 of Twin Cities 128. I strongly recommend that anyone who is interested in graphics on the C-128 have this modification done to their machine(s).

As I wrote in a preview of this product in issue #15 of Twin Cities 128, it is very difficult to describe in words what can be done with Basuc 8 without the benefit of a C-128 and RGBI color monitor at your side. Perhaps the two most Amiga-like features of Basic 8 are its color resolution and virtual screen abilities with 64K of VDC Ram. With the extra video RAM installed, Basic 8 can create virtual screens with a horizontal size of up to 2040 pixels and a vertical size of up to 819 pixels.

Using the virtual screen facility and the incredibly fluid 255 speed scroll command you can create spectacular animation effects. These effects range from standard cartoon-like page flipping to precise movement over very large bitmaps. Also aiding Basic 8's animation abilities is a very sophisticated and fairly fast buffering system provided for the movement, manipulation and duplication of screen data and the ability to rotate screen data around a user defined origin in all three dimensions.

And of course all the above can be done in full color. Basic 8 supports a variety of different color resolutions to allow the C-128 graphics artist an immense amount of flexibility. Under Basic 8 you can mix foreground and background colors in the following color matrices: 8 x 16 pixels, 8 x 8 pixels, 8 x 4 pixels, and 8 x 2 pixels. In addition, through a sophisticated technique called dithering which involves the mixing of specific colors with various background data you can achieve 128 different "shades" of color. Frankly, I cannot describe how stunning the resulting combination of Basic 8's array of colors used with the right artistic insight in the 8 x 2 color mode, you will just have to experience it yourself.

Conclusions?

So after reading this two page diatribe what thoughts would I like to leave with you? Simply this: The graphics potential of the C-128 is just now being fully understood and exploited. In many respects these new found abilities can compare favorably with the results found on much larger, and much more expensive (especially when you consider the price of both hardware and software) computers. Personally I gain a certain satisfaction from the exploitation of the C-128 in a domain where it is most certainly an underdog. In the near future, it is my belief that we are going to see even better applications of the aforementioned hardware and software, both unilaterally, as well as together as talented programmers devise conversion utilities for the transfer of pictures, fonts, and fill patterns between various software formats. And rest assured, Twin Cities 128 will remain at the forefront of these developments, helping to nurture them into reality and letting you know about them first. Anyway, I'm off to practice my jumpshot and...maybe its not too late after all.

Sparrow's Slick Tips *by Sparrow James*

00111110: DIRECTORY by a shorter name From Ray Bryan, Saint Paul, MN
A little known fact about Basic 7.0 is that it actually contains two commands for getting a disk directory, the standard DIRECTORY command, as well as the CATALOG command. The CATALOG command functions identically to the DIRECTORY command, but has one advantage, namely its keyword abbreviation is shorter as demonstrated below:

```
Command          Abbreviation
DIRECTORY        diR (di [shift] r)
CATALOG          cA  (c [shift] a)
```

Keep in mind that both the CATALOG and DIRECTORY command default to unit 8, drive 0, so in order to access any other drive or device you must specify them with the ON extension. (example: cA on u9 retrieves a directory from device 9)

00111111: Creating and using partitions and subdirectories with the 1581 From: Loren Lovhaug
One of the nicest thing about the 1581, besides its speed and storage capacity, is its ability to create partitions and subdirectories. This ability makes the organization of files on a high capacity drive like the 1581 much more manageable. Partition's on the 1581 are created through the use of the command channel. Partitions show up on your directories as "CBM" files, a new filetype implemented in the 1581's upgraded version of Commodore DOS. Here is how you create a partition:

```
Open 15,8,15,"/0:par-name,"+chr$(starting track)+chr$(starting sector)+chr$(low byte of # blocks)+
chr$(high byte of # blocks)+",C":Close 15
```

Partitions are nothing more than protected areas of disk space. Once a partition has been created the specified area on the disk is protected from BAM allocation changes and validates (COLLECT). One must be very careful not to create a partition that includes the 1581's directory track, track 40. Doing so will make your disk somewhat useless although I am convinced that something along these lines is going to be used as a copy protection device if commercial software vendors ever decide to market software in the 3.5 inch format. In addition, you must be careful if you have already saved some data on your disk that you do not overwrite it with a partition. The SHOW BAM utility on the 1581 test/utilities disk can show you what areas of your disk have not been previously allocated for data storage. Partitions are handy things because it makes it possible for foreign formats and bizarre stuff to "live" on the same disk as normal formatted data. For instance you can have a disk with both CP/M and native mode data on the same diskette (in addition it is looking quite possible that many really foreign MFM formats..like Atari ST may be able to live and even be read with the 1581.) In addition, provided your partition is created in such a way that it meets the following criteria you can format the partitioned area and thereby create a "sub-directory" (actually a directory of files that lives inside of the main or ROOT directory).

In order to create a sub-directory inside a partition the partition must be created as follows:

1. The partition area must be at least 120 blocks in size.
2. The starting sector must be 0.
3. The ending sector must be a multiple of 40 (and thus the total number of blocks are as well).

If your partition qualifies you can create a subdirectory as follows:

1. Select the partition via the command channel using: Open 15,8,15,"/0:partition name"
2. Format the area using the normal HEADER command or OPEN 15,8,15,"N0:name,id".

Now you have a nifty directory inside of your main directory. You can return to the main directory by either issuing a DCLEAR or OPEN 15,8,15,"I0" or selecting it via the command channel with Open 15,8,15,"/"
Note that DOS will chew up 40 blocks in your sub-directory for BAM, directory storage, etc.

You can access subdirectories from within any application program that allows you to send a command via the command channel. Most of the best packages allow this, such as Bobsterm, Pocket Writer, Paperclip, etc.

01000000: Advanced pattern matching on the 1581 From Fred Bowen, Commodore Engineering, via USENET
The latest version of Commodore DOS, version 3D, found in the new 1581 3.5 inch disk drive has some useful additions to the powerful filename pattern matching features. The new routines inside the 1581 now accept filename extensions delimited with a period as valid matching criteria for use with the wildcard character (the asterisk). Here are some examples:

Sparrow's Slick Tips Continued

```
(RECTORY"*.work"          Displays a directory of all files with the extension '.work'
 (RECTORY"*.work=p"       Displays a directory of all PRG type files with the extension '.work'
DIRECTORY"*.work=s"       Displays a directory of all SEQ type files with the extension '.work'
DIRECTORY"*.work=c"       Displays a directory of all CBM type files (partitions) with the extension '.work'
DIRECTORY"*.work=u"       Displays a directory of all USR type files with the extension '.work'
DIRECTORY"*.work=r"       Displays a directory of all REL type files with the extension '.work'
DIRECTORY"*.dir,*.work"   Displays a directory of all files with the extension '.dir' and '.work'
```

These pattern matching techniques will work on the DIRECTORY, CATALOG, and SCRATCH commands. Note that unlike some disk operating systems like CP/M and MS-DOS, Commodore DOS does not reserve any file extensions so you are free to create files with any extensions you like. Also note that the above file extension pattern matching abilities are only found in the 1581's version of DOS thus far, so you should not rely upon them if you are writing a program that might be used on any earlier Commodore drive such as the 1571 or the 1541.

01000001: BASICPAINT on monochrome monitors From Louis Wallace, BASIC 8 co-author
Some of you may have noticed some problems using BASICPAINT on monochrome monitors. David Darus and I developed Basic 8 specifically for use with RGBI monitors, and you are missing a lot if you are doing any Basic 8 hacking on a green screen, but there is no reason why you can't use a monochrome monitor, provided you make a few alterations to BASICPAINT. You see, since David and I felt that this package was meant to be used with RGBI monitors we forgot to test it with monochrome monitors. As you may (or may not) have noticed the black and dark grey colors of BASICPAINT are impossible to read on a monochrome monitor, but fortunately this is a small oversight that is easily fixed. Here is what you must do:

Line #	Fix
72	Change the definition of the variables FC, BC, and OC to: FC=15:BC=0:OC=0
160	Change the parameters in the @CLEAR and @COLOR commands to: @CLEAR,0,0,15:@COLOR,0,15,0
1573	Change quote mode colors to CTRL-C, CTRL-white (the 2 key), CTRL-black (the 1 key). This will appear as a reverse CEP.
1796	Same change as in line 1573
2006	Same change as in line 1573
9255	Change the parameters in the @CLEAR and @COLOR commands to: @COLOR,0,15,0:@CLEAR,0,0,15
9510	Change the parameters in the @CLEAR and @COLOR commands to: @COLOR,0,15,0:@CLEAR,0,0,15
9520	Same change as in line 1573
10037	Same change as in line 1573

These nine lines will change Basic Paint from the default grey and black to white and black, providing the contrast necessary for using monochrome monitors.

01000010: Easy documentation saver From Sparrow James, Sitting in the midst of too many manuals
If you are like me, in the midst of programming or using a complex piece of application software you are constantly finding yourself consulting the program's or language's documentation. Perhaps the most time consuming and frustrating aspect of this procedure is the "search syndrome", especially in manuals that do not provide indexes and tables of contents. In addition, the search syndrome also has a detrimental effect on the pages of your manual as they become, worn, torn, and dog-eared. Some have suggested as a solution to the "search syndrome" the use of handy command summary reference cards. But for me this solution is not a useful one, because given the state of my office I would be forever searching for the reference card itself. Also often in complex operations a short command summary is just not a replacement for detailed explanations. But recently I have come upon a solution that promises salvation for my Superbase, Basic 8, and COMAL manuals. This solution involves the placing of colored and marked index tabs (the kind you can get from your local stationary store) at various vertical positions on the outside edges of your manual pages making it much easier to locate key sections of your documentation. Of course, none of us really has the time or desire to go through the drudgery of doing this to each of our manuals ourselves, so go and enlist the help of your spouse, children, or friend the librarian down the street. Note: When dealing with your spouse or children, your selling of this project is crucial. Be sure to use terms like, "family togetherness", "interesting diversion from the tube". If all else fails, bribery also works well. As for librarians, don't worry, I have found that librarians don't seem to mind doing this kind of "busy work" since they do it all day long anyway. In fact some of them seem to enjoy it.

Rumor/Opinion/Mayhem *by Loren Lovhaug*

Well at long last here it is, issue 17 of Twin Cities 128. Six weeks in the making, the look and the content of this issue reflects the immense amount of change and progress that is taking place in the Commodore 128 world, and indeed within the entire spectrum of microcomputing. If this is not your first issue of Twin Cities 128, by now you have no doubt noticed we definitely have a new look. The acquisition of the Okidata Laserline 6 laser printer represents a major step in the continued evolution of Twin Cities 128. The Laserline 6 gives us the ability to produce near typeset quality text with our lowly Commodore 128s. For months we have been investigating the possibility of purchasing a laser printer to improve the appearance and readability of Twin Cities 128, but one glaring reality has always prevented us from doing so: cost.

Even as late as six months ago it was nearly impossible to find a laser printer selling on the open market for less than $2500, and most were in the $3500 to $7000 range. But just within the last six months radical technological changes, combined with increased demand and competition in the laser printer market have caused incredible price reductions which have left the price of many laser printers at under $2000. And now there is serious talk of an under $1000 laser appearing within the next two years.

Now I realize that one or two thousand dollars is much more than most Commodore 128 owners are willing or able to spend on their computer systems. In fact, we have taken a big risk ourselves in investing such a large amount of capital in our new printer. But I think the method behind our madness is worth considering.

In these days of shiny new sixteen and thirty-two bit micros which boast superior facilities for graphics and desktop publishing, it is indeed interesting to note that none of them can touch the quality of the page you are reading unless they themselves are connected to a laser printer or phototypesetter. My point here is that it is not the processing engine, the microcomputer, that creates these wonders, but the printing engine, the laser printer. And to that end, it does not matter what kind of computer you have hooked up to the laser printer, just that you have one.

Now it is true, that as of this writing, the owners of sixteen and thirty-two bit microcomputers like the IBM PC, Macintosh, Amiga, and Atari ST do have better software support for laser printers and advanced desktop publishing applications, but this advantage is rapidly shrinking. And as laser printing technology becomes more and more affordable, and as more Commodore computerists begin gaining access to this technology it is likely that other software companies will follow suit.

In addition, I have found that it is not nearly as difficult as some computer salespeople would lead you to believe to connect your Commodore 128 to a laser printer and adapt your existing software to communicate with it. In fact, with a little study and effort, I have found that just about any software package that will allow you to create a custom printer driver can in fact drive the laser printer. For instance this article is being composed and printed with Pocket Writer 2, a word processor that does not include a printer driver for either the Okidata Laserline 6 or the HP Laserjet (our Laserline 6 is HP Laserjet compatible) but does give you the option to create your own printer driver. The only difficulty we have run into involves the usage of proportional text, such as the Times Roman Compressed font I am using on this page. Proportional fonts, unlike more traditional computer printer fonts, do not allocate the same amount of space on the page for each character, thereby making layout manipulations such as right justification and multiple columnar output much more difficult. This issue reflects our first attempt at using the laser printer for the production of our publication and I am confident as we continue to use it we will develop even better strategies for using the Commodore 128 to do what others have deemed impossible.

Since our inception in January of 1986, I believe we have been at the forefront of C-128 usage. Now I want to expand our excellence beyond the simple reporting of C-128 application ideas, programming concepts, evaluation of hardware and software tools. I want Twin Cities 128 to be the driving force in the C-128 community. The laser printer is just one example of where we are making inroads into areas that had previously not been made with the C-128. We, like you, are interested in getting the most out of our computing investment. I realize that the C-128 is by no means the perfect microcomputer, and experience has taught me that even the paramount achievement in computing technology is not going to be without its flaws. But when dealing with imperfection one can take one of two approaches: 1. Abandon the source of the imperfection in search of perfection or 2. Adapt to and/or conquer the imperfection and create ones own version of perfection. I have chosen the latter.

It is been a long time since I have passed along, and passed judgement on, any of the hordes of rumors that pervade the Commodore computing world so I'll do my best to tantalize you this time. Please remember that information that appears on this page, unlike that which appears on the next page under the C-128 Price and Progress report is far from a bona fide reality so please keep what is printed here in its proper perspective.

Rumor/Opinion/Mayhem Continued

KEEPING OUR FINGERS CROSSED:

I keep hearing that the prospect of real time spell checking is about to become a reality for the C-128 community. With real time spell checking a word is automatically compared against an electronic dictionary immediately after the last character in a word is typed (i.e. when the space bar is pressed) and the user is informed immediately thereafter whether or not that word was found in the dictionary. Until recently, this facility was available only on large mini and mainframe computer systems and the most expensive word processing programs in the 16/32 bit world. As the story goes, supposedly updates of both Fontmaster 128 by Xetec and Paperclip now owned by Electronic Arts would use the 1700/1750 Ram expansion units for dictionary storage. The speculation is that both programs will be available before Christmas and may also include as part of this real time spell checking ability the ability to "look up" words and the presentation of alternative (correct) spellings for words not found in the dictionary.

Lou Wallace & David Darus, the masterminds behind what has to be one of the greatest C-128 programming accomplishments of all time are contemplating a sequel to Basic 8. In a recent telephone conversation with Lou, he indicated that he and David were considering doing a 100% machine language drawing and CAD package for the C-128's 80 column screen, if Basic 8 sales reach a certain level by late this year. Lou said that both he and David were very concerned about the amount of piracy that the unprotected Basic 8 seems to be falling victim to. In summary, Lou indicated that they would like to create the 'ultimate drawing package ever created for an 8 bit microcomputer' but neither he nor David felt like spending the hundreds of hours such an endeavor would require if the C-128 user community could not demonstrate that they can be trusted not to steal their efforts. Editor's Note: Please refrain from the illegal duplication and distribution of commercial software and refuse to tolerate those who condone such illegal acts. It is their greed and thoughtlessness which drives talented developers like Lou and Dave away from Commodore computers and consumer computing in general.

When I was in Chicago a few weeks ago at the Fox Valley Commodore Userfest, I spoke with the Timeworks representative manning their booth. Since Timeworks has been an avid support of the C-128 I inquired whether they plan on developing a C-128 version of their recently announced desktop publishing package for the C-64. He indicated to me that preliminary work on a C-128 version was under way and that it was very likely that a finished product could be on the market as early as February of 1988. Meanwhile, encouraged by brisk sales of the C-128 version of GEOS, Rob Siegel of Berkeley Softworks told me that a C-128 version of their soon to be released desktop publishing programming program called Geopublish was being considered, and that the C-128 version of Writer's Workshop should be shipping soon. Meanwhile Rob said, he believed that the C-64 version of Geopublish would function under the 128 version of GEOS in 40 column mode.

A third party developer is considering marketing "smart" replacement keyboards for the C-128 and C-128D. Supposedly these keyboards would be detachable and feature an industry standard layout with the four basic arithmetic operators and parentheses on the numeric keypad, 32 battery-backed programmable function keys, and a true four direction diamond (no, not that sissy inverted T layout on the Amiga 500 and the IBM PS/2 machines) cursor key layout. Additionally, this keyboard would also have a 5 position angle adjustment and retail for under $100.00.

TOTAL FICTION WITH NO BASIS IN REALITY:

Cheryl Peterson formally of Ahoy magazine and now of Computer Shopper fame will be joining the Twin Cities 128 staff. INFO has made a $200,000 bid to buy out Twin Cities 128. Steve Jobs, co-founder of Apple Computer, has asked us to dump Twin Cities 128 in favor of a new publication called Twin Cities NeXT. There was a chance of getting this issue out on schedule.

An Evaluation of CP/M Kit *by Randy Margolis*

Not long after the C-128 came out there started appearing article after article on 'Introductory CP/M' in every major Commodore magazine. It brought to mind an old saying about the weather: 'Everyone talks about Introductory CP/M, but nobody ever does anything about it'. Well, a new product from INCA (Innovative Computer Accessories) is attempting to put this situation to rights. Called CP/M Kit, this $29.95 package consists of a forty page manual and two disks (one has programs on both sides) full of public domain CP/M programs and utilities.

The manual is more of an introductory CP/M primer than an explanation of the included software, but all of the supplied programs are fully documented on the disks. As a CP/M tutorial the manual is actually very good, if one understands that it is targeted at the novice CP/M user. The nice thing about it is that its approach to the subject is from a C-128 perspective. There are a lot of little things that make Commodore's implementation of CP/M quite elegant, and the first twenty pages of this booklet do a nice job of explaining the basics from a beginner's point of view. The text is not extensive but should suffice in getting one started.

An Evaluation of CP/M Kit Continued

As previously stated, the manual does not go into lengthy explanations about the supplied programs, instead forcing the user to print out his own documentation material from files provided. Several utilities are provided for this purpose requiring no special knowledge to use. The exception is a chapter in the book on the MEX terminal program. In case you have an early version of the CP/M operating system (releases dated prior to 6 Dec 85 were not capable of supporting the modem port), INCA has provided NEWSYS.COM, a utility to transform your software to the newer implementation. Nine pages are set aside for explaining how to use MEX. This is probably the best section of the manual, although the entire book is well done, with no inaccuracies.

In issue #13 of Twin Cities 128 I wrote an article describing several essential public domain CP/M programs, and most of them are included in the CP/M Kit. The following is a listing of the programs contained in the CP/M Kit collection:

```
C1571    - Speeds up 1571 disk operations
CONF     - Sets system parameters
DD       - A gem of a directory lister
DE-LBR   - Dissolves library files quickly
EDFILE   - Easy to use (unlike ED.COM) text editor
LDIR     - Lists files inside a library file without having to dissolve it.
LRUN     - Allows the execution of a program contained inside a library!
LTYPE    - Lists a text file contained within a library, squeezed files.
MCAT43   - Still the best CP/M disk cataloger
MEX128   - CP/M terminal program par excellence
NEWSWEEP - Multi-purpose file and disk handler (absolutely essential)
NEWSYS   - Updates early versions of CP/M
NULU12   - All purpose library utility
SCAN12   - Allows viewing of text files forwards or backwards
SD-50    - Similar to DD, but will list files within a library, too
SQ       - File compression utility, makes library files more compact
USQ      - Unsqueezes files created with SQ
VDE13    - Full featured PD mini word processor
XCAT43   - Used with MCAT43 to print out disk catalogues
ZCHESS   - Surprisingly competent CP/M chess program
```

Some may wonder why they should pay $30 for programs in the public domain. Most, if not all, of these programs will be found in a well stocked user group library, or on the national telecommunication networks. However, if you don't have a nearby user group, I guarantee that it will cost you at least as much as this package costs to download all these items from Compuserve, GEnie, etc. In addition, these utilities have all been tested to work on your C-128 without any modification or 'installing' on your part. Also, the booklet is a good resource for getting started fooling around with CP/M.

My only complaint with the package is that a couple of the programs have had several updates since the versions provided on the disks. They still work perfectly, but some new features have been added since these came out. Specifically, VDE13 and NULU12 are somewhat elderly. I hope that INCA keeps up with the latest versions in future releases of this product. In their defense, however, updates of public domain programs do come out with surprising frequency, and lead times for producing a product of this type are large. This minor complaint aside, you will not be making a mistake buying CP/M Kit if you are an inexperienced CP/M user. The applications and utilities in CP/M Kit will become staples of your CP/M collection immediately.

At Home With the Commodore 128D *by Loren Lovhaug*

If you are a regular reader of Twin Cities 128 you know that I have been a C-128 owner since the C-128's introduction in the late Summer of 1985. For months prior to that time, I had been gathering and digesting just about every piece of information concerning the imminent release of the Commodore 128. But you might be surprised to know that the C-128 was not the computer that I had originally decided I wanted to purchase during the summer of 1985. You see, after reading the CES Report in issue #6 of INFO magazine, I was convinced that I wanted a Commodore 128D and not the standard Commodore 128 that I had read so much about. According to that issue of INFO, the Commodore 128D was a special version of the Commodore 128 that featured a built-in 1571 disk drive, a detachable keyboard that could be stored beneath the system unit, and a suitcase like handle that made it semi-transportable which would retail for around $750. This more professional looking model of the C-128 was supposed to be sold only at computer specialty stores and not by mass merchants or mail order houses, hence its moniker, C-128D, the 'D' supposedly standing for dealer. The idea was, or so it was reported, that this special version of the Commodore 128 would keep local dealers happy since with the Commodore 128D they would not have to

At Home With the Commodore 128D Continued

mpete with the volume discounters and could attract a more sophisticated buyer (supposedly because of the ommodore 128D's more professional appearance).

However, two years, almost to the day, after I bought my first C-128 because I decided I just could not wait for or afford the Commodore 128D, a C-128D has finally graced my desktop. This C-128D however is vastly different from the one pictured and described in INFO two years ago. The C-128D they were writing about did appear, in fact it is sold in Europe along with our traditional "flat" C-128, but it is definitely not the same machine that I am typing on right now.

First, the North American version of the C-128D does not have a handle or a place beneath the system unit to store the detachable keyboard like its European predecessor. This is truly a shame, because I know I am not the only C-128 owner who wishes their computer was more transportable. The system unit, like the European version still houses a built-in 1571 disk drive, but it is not nearly as tall as the 128D in Europe and is made of metal except for the front plate which is made of an extremely thick plastic. The metal case makes the machine more solid, but adds a lot of weight, this machine is definitely <u>not</u> transportable.

The inclusion of a metal case on the C-128D was not for cosmetic purposes. The power supply for both the computer and the 1571 disk drive lies within the system unit and produces a great deal of heat which could have a detrimental effect on the operation of the system. The metal case is designed to conduct the heat out of the system unit and to serve as a radio frequency interference shield. I am pleased to report that the metal case performs adequately at both tasks. I leave my C-128s on almost constantly, and the 128D is not affected at all, although the metal case does get warm (as it should). The metal case also seems to do a good job as an RF barrier. We use a cordless phone at the Twin Cities 128 international headquarters and I have always noticed that it begins to buzz and crackle a bit when I am near the computers. However, in purely unscientific testing the C-128D seems to affect the cordless phone less than my traditional C-128s.

I have but one reservation about the metal case on the system unit and that has to do with the popular notion that it makes a good monitor stand; in my opinion it does not. Sure, all the pictures in the system guide show a monitor placed upon the system unit, but pictures are one thing, practical usage is quite another. You see, nitors, especially RGBI monitors like the Commodore 1902 and 2002 are heavy and give off a great deal of heat. ..s my suspicion that although the metal case on the C-128D appears to be quite sturdy, that over time with a heavy monitor placed upon it, it might begin to bow in the middle. Also most monitors have heat vents on their undersides. Think about it. Do you really want to bet your computing investment on the hunch that the metal case can conduct away all of the heat from the power supply and the integrated circuits inside the C-128D, with a monitor sitting on top of it? Besides, for me the viewing angle of a monitor perched upon the C-128D's system unit is too low anyway. However, all is not lost! The C-128D's system unit still makes an excellent shelf. I simply slide the system unit beneath a monitor stand and put my 1581 disk drive (the 1581 stays very cool since its power supply is external) and a small copy holder on the system unit making the C-128D set up quite functional as well as attractive.

The keyboard on the C-128D is laid out exactly as the traditional C-128 keyboard. The only difference is the obvious fact it is detachable, and therefore provides for more flexibility as far as positioning on your desktop and one other very subtle difference. The angle of assent on the C-128D keyboard is slightly shallower than that of the traditional C-128. I noticed this right away, though I doubt that most people, unless they do as much work on their C-128s as I do, would notice the difference. Although I have been using the C-128D for over two weeks now I have not yet determined if the shallower angle is any better or any worse to work with, though I still notice it when I move from machine to machine. The keyboard is attached to the system unit via a two foot cylindrical cable which is approximately a quarter of an inch in diameter. This cable cannot be removed from the keyboard unit. This is unfortunate because it prevents someone from adding a longer keyboard cable, without actually tearing apart the keyboard itself. This prevents side-mounting the system unit as is popular in the IBM world.

Internally the C-128D is the same old C-128 you know and love, except for two important differences. The C-128D comes complete with 64K of VDC Ram and the latest version of the C-128 kernal and 1571 Roms. The 64K of VDC Ram allows for spectacular 80 column graphics, with packages like Basic 8, which use the extra memory for color attributes, multiple screens, and virtual screens. The updated C-128 Roms correct some of the minor flaws which were part of earlier C-128s, such as the infamous CAPS Q bug, problems with the graphics commands when using relative coordinates, and round off errors with some of the math routines. The updated 1571 Roms make the 1571 disk drive perform much better when reading and writing direct access and relative files, single sided disks, and gn formats from CP/M. It should be noted that the 1571 inside of the C-128D is different from the external 1571 you probably own. The internal 1571 drive in the C-128D is the new "cost reduced" version of the 1571 which

At Home With the Commodore 128D Continued

does not incorporate the Western Digital 1770 controller. I have not found this to be a problem, although I am sure, given the wacky world of software copy protection somebody out there is making software that will not run on the new implementation of the 1571 drive...it is inevitable. By the way, all future external versions of the 1571 will also be of the "cost-reduced" variety, for more information on how to tell what version your drive is see the C-128 price and progress report in this issue of TC-128.

Externally, besides the differences already mentioned, there are a few other physical differences worth noting. The port configuration of the C-128D is similar to that of the "classic" C-128, with the expansion, user, serial, and video ports all in their familiar spots in the rear of the machine. Of course, since the C-128D features a built-in disk drive, the purpose of serial connector on the rear of the machine is now for chaining external peripherals and not for connecting your first disk drive as was previously the case. Moved to the rear of the machine is the main power switch (Does anybody besides me long for the day when computer manufacturers will opt for putting switches in the front of their machines instead of the rear?) which is located on the left side of the machine just above the user port. Gone from the internal 1571 are the useful device setting switches. This means that you must use the internal 1571 disk drive as device 8 unless you software select it as a different device. Unfortunately many commercial software packages are not compatible with the software selection method. It is possible to create your own hardware device setting switches and it is likely we will outline this procedure in a future issue of Twin Cities 128 (you can bet our hack will put the device switches in the front and they sure won't be those tiny darn dip switches). On the right side of the computer we find what is perhaps the most drastic change in port configuration on the C-128D. The cassette port as been moved to the near right side of the machine. Although originally designed to facilitate the 1531 datasette a device that used cassette tape as a mass storage medium, this port is used primarily to power many popular printer interfaces. The movement of this port to the side of the C-128D is a mixed blessing, depending on your system setup. For people who place their printers to the right of their computers, the cassette port placement makes the interfacing much easier, those who place their printers to the left of their machines, may find that the new cassette port position renders their cables too short. This may also lead to some inconvenience as a result of bringing a cable to the forefront of your machine that previously resided in the background. Just behind the cassette port lies the keyboard connector and the mouse/joystick ports. Finally, better than 12 inches back from the front of the system unit, lie the computer reset switch and a drive reset switch. Since most people are going to place their keyboard two to nine inches in front of the system unit, this makes for a pretty difficult reach when you want to reset the computer or disk drive (these switches belong on the front too). The drive reset switch has been added since with the C-128D you are unable to turn off the internal 1571 drive without turning off the computer and at times catastrophic may require a cold reset of the drive. Unfortunately, some hardware design engineer with a flair for masochism recessed the drive reset button into the case so that it can only be pressed by inserting a pencil, thereby making a somewhat difficult reach into an impossible one.

Okay, so what is the bottom line on the C-128D? Well the popular marketing hype is being offered because it is better ergonomically designed and more space efficient. That is horsehockey. The reasoning behind this incarnation is plain and simple: it is cheaper to build and squares better with Commodore's long term plans for the 128. But that does not mean that it is not a nice computer. I like the C-128D, despite my physical criticisms it is a good value at $425 - $475, especially when you consider it comes complete with the upgrade C-128 and 1571 Roms and 64K of VDC Ram. The C-128D represents an excellent choice for a C-128 owner looking for a second machine.

Machine Language Routines for the C-128 *by Miklos Garamszeghy*

Have you ever thought that you have been re-inventing the wheel every time you write an assembly language program? Would you like a large library of documented assembly language source code routines? Or, are you merely curious about how assembly language is actually used in practical applications?

If you answered yes to any or all of the above questions, then you should consider acquiring a copy of "Machine Language Routines for the Commodore 64/128" from COMPUTE! Books, by Todd D. Heimarck and Patrick Parrish. The 580 pages of this recent release contains the assembly language source code for some 200 fully documented, commented, and tested routines for performing a wide range of tasks from file input/output to floating point math to programming the CIA TOD clocks to alphabetizing and searching lists to ASCII <> PETSCII conversions, to name but a few. Although they are not intended to be used as stand alone programs, many can be used with little or no additional programming by incorporating the additional coding provided with each routine to demonstrate its use.

Now that I have tweeked your interest, let me start at the beginning. The first few chapters of the book are devoted to a description of the 6502/6510/8502 type mnemonics instructions such as LDA, STA, etc. and their associated op-codes, and the KERNAL function calls such as CLRCHN, SETLFS, etc. The mnemonic descriptions are similar in style to those found in other COMPUTE! books on machine language programming. The descriptions are

Machine Language Routines For the C-128 Review Continued

short but clear and to the point. References are made in the introduction to several other more comprehensive works on machine language and assembly language programming, including some non-COMPUTE! publications. The descriptions of the KERNAL function calls, which include both standard ones as well as the new C-128 entries, are also short but clear. No real examples of how to use either the KERNAL or the op-codes are given at this point, but all are used extensively in the remainder of the book.

The meat of the book is some 500 pages of carefully explained and documented assembly language routines arranged in alphabetical order by an arbitrarily assigned program name such as ALARM2 (for setting up TOD clock#2 as an alarm clock) or BORCOL (for changing the border color). The routines are presented in a format suitable for a PAL or Buddy type assembler, but the introduction gives tips and guidance for converting the listings to other assembler formats. If you prefer to do it the hard way, the routines can also be entered using a monitor such as Basic 7.0's MONITOR command. However, this approach does require much more manual labor on your part for calculating absolute addresses and label references. Tedious, but possible.

The routines are presented in true assembly language format with address and data labels. (Despite the reference to machine language in the title, the book actually deals with assembly language. There is a small but definite difference between the two. Strictly speaking, machine language deals with numbers only while assembly language deals with a set of arbitrarily defined mnemonics used to represent the numbers. For example, $AD $FF $8D $00 $D0 in machine language is equivalent to LDA #$FF : STA $D000 in assembly language). Unlike an earlier COMPUTE! book with a similar title ("Machine Language Routines for the Commodore 64"), BASIC loader DATA statements are not provided for the routines. If you want to SYS to any of them from BASIC, you must create the DATA statements and POKE it in. An assembly language subroutine for creating DATA statements from object code in memory is provided as one of the routines in the book.

Most of the routines are shown as being located in high memory. This follows from the C-64 tradition of putting such things in the unused chunk of Ram at $C000. However, with an assembler, the code is fully relocatable by changing the assembly origin statement. If you want to incorporate more than one of the subroutines or mix them with your own assembly code, relocation is inevitable. It should also be noted that on the C-128 the routines could be placed in low memory, say at $0B00 or $1300, to avoid having to deal with bank switching for accessing the KERNAL and Basic Rom's as well as the I/O block. Remember that on the C-128, the Ram below $4000 is visible in both BANK 0 and BANK 15 and is an ideal spot for code needing both banks. The introduction discusses these requirements for the C-128 but makes no attempt to explain C-128 bank switching in assembly language (it is very simple) and refers the reader to other sources for an explanation.

Because of the similarity between the two machines, most of the routines will work without modification on either the C-128 or C-64. Where differences exist, due mainly to different addresses for Basic Rom routines called as subroutines, the documentation for that routines clearly explains the required changes for each machine. Most of the listings are given for the C-64 format with C-128 modifications listed as comments. It should even be possible to adapt the routines to work on other Commodore machines, such as the VIC-20, PLUS/4, etc. Routines designed specifically for one machine or the other, such as accessing the 80 column chip or Ram expander on the C-128, are clearly noted as being such, although I suspect that they would work quite well on a C-128 in C-64 mode with very minor modifications. The Ram expander routines should also work on the C-64 with the new 1764 expander.

The collection of routines provided in the book gives something for everyone: from some quite simple routines to very complex ones. A wide range of topics is provided from math and conversion routines to graphics and sound to chip register programming and many more. In fact it covers enough ground to do almost all of the mundane and boring tasks that many programmers don't like to waste a whole lot of time developing for themselves. Face it, most people are lazy, so why re-invent the wheel? If you are really lazy, you can mail in the coupon provided at the end of the book with an extra $12.95 (plus postage, handling and applicable taxes) and get a disk containing the source code for each of the routines in the book in PAL/BUDDY format. Even if you choose not to use the routines exactly as listed, they can serve as a very useful starting point for your own custom routines.

Although the introduction states that the book is not intended to be an introduction to assembly language programming, I would recommend this book most strongly for beginner to intermediate programmers. These are the people who would benefit most from the clean programming style and clear documentation of how each routine works. Despite the above statement, the bottom line is that "Machine Language Routines" will be a welcome addition to the library of virtually any assembly language programmer. (Remember, Christmas is not too far away).

Multiple Column Output With Superbase 128 *by Loren Lovhaug*

Superbase 128 is a truly useful program for all manner of database management. However, one area that I (and, I suspect, some others) sometimes have difficulty with is getting all that information stored in databases on paper exactly the way I want it. Recently, I decided that it would be nice to output a listing of my library of video taped movies in two columns, alphabetically down the first column, and then continuing in the second. Since there is no built-in facility for doing this, it necessitated writing a small program. I hope that this program will help you with your Superbase output programming.

My file setup includes fields for [title] (27 characters), [no.] (5 characters), and [year] (4 characters numeric). Using compressed print, there is plenty of room across the sheet for two columns. The approach here is to read all the records into array variables before printing. Line 30 is a trap statement which passes control to line 280 in case of any difficulty. It's a very good practice to include a TRAP statement in all your Superbase programs, especially during the program testing and debugging stages. After choosing the file in line 40, (just in case you were previously working with another file), you need to find a record count in order to DIMension the arrays. There are several different ways to approach this. The method presented in the program listing (line 70) requires no user input, but causes the database to be read from the disk twice. Another way could be to substitute this line: 70 catalog:ask "Number of Records";n In this case the user is required to type in the number displayed next to the proper file on the catalog display. This is much less elegant, but quite a bit faster, especially if your file contains many records. A third approach is to hard-code the number of elements in the array. The only problem with this method is that you have to change the program every time you add new records. Line 80 tests the number of records read in and changes it to an even number if it's odd. Line 100 DIMensions the array to the proper size. Line 120 is the start of the printing routine. My printer requires a CHR$(15) to set compressed print. Substitute whatever code your printer needs in line 130, and don't forget to set the right margin to 136. The DO LOOP in lines 150 to 200 reads the records from an alphabetically sorted list into the string array elements. Since my [year] field is numeric, I have to use the STR$ function to change it to a string as it's fetched from the records. Lines 210 and 220 print a two line underlined title including the date asked for in line 110. Including the date of the list helps you identify the most current printout when there are changes in your database. The second DO loop (lines 240 - 270) outputs the two columns of variables to paper. The variable B is defined as half of the number of records since we are outputting using to columns (see line 90). Finally, line 280 cleans everything up, sending a form feed to the printer, resetting the pitch to pica, the right margin to 80 and returns to the main Superbase menu. This is also the line indicated in the TRAP statement. I'll admit to being surprised by the small number of program lines required to accomplish a task I thought would take considerably more programming. You can modify this program to use as a subroutine of a larger program. With Superbase 128 you are limited only by your imagination.

```
 30 TRAP 280
 40 FILE "FILMS"
 50 SCNCLR
 60 N=0
 70 BATCH ALL N=N+1;@0@10,4"COUNTING RECORDS:"&3,ON
 80 IF N/2=INT(N/2)THEN Y=N:ELSE Y=N+1
 90 B=Y/2
100 DIM A$(Y),B$(Y),C$(Y)
110 ASK "DATE";D$
120 PRINT ACROSS:RMARG 136
130 PRINT CHR$(15)
140 I=0
150 DO
160 SELECT FROM "FILMSORT"
170 EOL EXIT
180 A$(I)=[TITLE]:B$(I)=[NO.]:C$(I)=STR$([YEAR])
190 I=I+1
200 LOOP
210 PRINT @48"FILMS-ALPHABETICAL ";D$
220 PRINT @-@5,0"TITLE"@40"NO." @50"YEAR"@63"TITLE"@103"NO."@113"YEAR"@-
230 F=0
240 DO UNTIL F=B
250 PRINT @5A$(F)@40B$(F)@50C$(F)@63A$(F+B)@103B$(F+B)@113C$(F+B)
260 F=F+1
270 LOOP
280 PRINT @0,12CHR$(18):RMARG 80:DISPLAY:MENU
```

Basic 8 Unreview *by Loren Lovhaug*

I hate software reviews. Not reading them, I hate writing them. Why? First, because I would much rather create or do something original with my computer rather than critique someone else's work. And second, because the staple of every other Commodore computing publication is its software reviews, and I want Twin Cities 128 to go beyond what the others do in every facet of Commodore 128 coverage. Take, for instance, a product like Basic 8. In the coming months you probably will see every major Commodore rag review Basic 8. They will summarize the basic story about Basic 8, namely that it is a revolutionary graphics programming language that unleashes the previously untapped 80 column graphics potential of the C-128. The better reviewers will go on to outline the product's specifications and potential uses. And that is where they will stop, relating the obligatory who, what, why, and where plus their own opinions on the product's worth and implementation. I am not going to do that. I am more interested in helping you do something with Basic 8. Besides if you are a regular reader of Twin Cities 128 you have already read the obligatory information anyway. (See issues 15 & 16) And as far as an evaluation goes, I think the following will suffice: Basic 8 is excellent. It is a must for every C-128 programmer, expert and novice alike. Both programming and non-programming C-128 users will benefit from the innovative applications that have been and are being produced each day. Enough said. We feel Basic 8 is so important to the C-128 community that we are going to try to feature an article on its use in each and every issue of Twin Cities 128 from now on.

Basic 8 Programming Strategies

Over the past couple of weeks I have talked to a lot of C-128 owners who have purchased Basic 8. All of them tell me they are impressed with Basic 8's abilities. But many of them also tell me that sheer vastness of Basic 8's command, combined with the already extensive Basic 7.0 command set makes it difficult for them to get started writing programs, especially larger programs that involve a whole host of operations. So here are a few ideas that might make the whole process of starting a Basic 8 project a bit easier.

Being an extension to the C-128's built-in Basic 7.0 programming language, Basic 8 suffers from the same fundamental flaw that the Basic programming language in general suffers from: namely it is too easy to hack out code. Don't get me wrong, programming should not be painful. But the same features that make Basic so attractive and easy to use, such as its interpretive nature and its lack of rigid structural requirements also lead to sloppy programming which in many instances makes programs harder to efficiently write and debug. Most of you veteran programmers probably know what I am getting at. Basic really does not require you to plan your work. In fact, most people when they program in Basic simply sit down and do it. They begin writing their program with just a vague idea of the goal they wish to accomplish, instead of breaking the task into smaller objectives and developing schemes for integrating the solutions to those smaller objectives into a program that ultimately accomplishes the desired goal. The problem with the free form approach that most people take to Basic programming is that it is almost always more time consuming and frustrating. This is because you often find yourself sitting in front of the keyboard trying to figure out what to do next, deciding how to make your previous work mesh with your latest or sitting with your face buried in a manual trying to determine what command or syntax you should be using.

By organizing your program before you go to the keyboard into smaller component tasks that work together as a whole you can save yourself a lot of grief. Essentially this is all modular or structural programming entails, although many people try to make it much more than that by enforcing rigid notions about what your planning stages should entail (i.e. requiring the construction of elaborate flow charts, or extensive outlines and variable charts etc.) Personally, I think these people are missing the point, a modular approach is supposed to make programming easier instead of creating a lot of ancillary work. Before I begin a Basic 8 program I usually go through the following checklist either mentally or on a piece of scratch paper.

Basic 8 Pre-programming checklist

A. General Questions - These questions are common to all programming

1. What is the intent of the program? Simply speaking, what should the program do? I try to imagine what options the program should include, and perhaps even how the user will be presented with those options. The more specific ideas you have about what you want, the easier it will be when it comes to actually making it happen. However, at this point it is not necessary to actually do any coding.

2. Who are you writing this program for? In other words, are you writing this program for yourself or do you want others to be able to use it? The answer to this question is very important. Many of your design considerations will hinge upon the characteristics of your target audience. Good programmers take into account things like the age, computer literacy, dexterity, etc. of their target audience.

B. Basic 8 Considerations - These questions pertain to design considerations specific to Basic 8

1. Will your program be designed to run using 16K or 64K of Video display Ram or both? Obviously, if you have only 16K of video Ram installed in your C-128 then you don't have a choice. However, if you do have 64K of video Ram in your C-128 you have a major decision. Using 64K of video Ram allows you to utilize many capabilities of

Basic 8 Unreview Continued

Basic 8 that simply are not available with 16K. These include multiple screens in memory, virtual screens (those larger than the 640 x 200 viewing area), and multi-color graphic screens without shrinking the vertical resolution of the viewing area. Given these advantages many of you might wonder why anyone who has 64K of VDC would want limit themselves to 16K. Well the answer is simple, the vast majority of C-128 owners only have 16K of video Ram in their machines. If you want your program to function on any C-128 you must make concessions and write your program with 16K of video Ram in mind. There are two approaches you can take if you are a 64K VDC Ram owner and wish to write programs that will run on 16K machines. The first approach is to simply write the program completely from a 16K standpoint. This is what Lou Wallace and David Darus did with their workbench program. This approach works well for simple programs. The second approach is much more complicated and involves writing your program so that it can use 16K VDC machines or 64K VDC machines. This is what Wallace & Darus did with BASICPAINT. The most common way of doing this is simply asking the user to indicate whether he/she is using a 16K or 64K VDC Ram machine. There is one other more radical option, one, in a way, I kind of embrace: namely write your programs for 64K only and make the rest of the world adapt. The @WALRUS command is used to select whether you are using 16K or 64K of VDC Ram.

2. What screen mode and color cell resolution are you going to use? Basic 8 has four distinct screen modes and five color cell resolutions (counting monochrome). The screen modes and the respective color cell resolutions available in each mode are outlined on page 90 of the Basic 8 manual. Which screen mode and color cell resolution you use depends upon your application and your hardware. If you are programming for 16K video Ram machines only one screen mode is available, although you can use all five color resolutions. However, on 16K video Ram machines, the higher the color resolution the more vertical bitmap (plotting) resolution you must sacrifice (e.g. if you use an 8 x 8 pixel color cell, you can utilize a plotting area of 640 x 176 pixels, however if you want to have an 8 x 2 pixel color cell you can utilize only a 640 x 104 pixel plotting area). With 64K of VDC Ram this is not a problem. With 64K of VDC Ram, depending on which color cell resolution you choose, you can maintain multiple screens of various sizes in VDC memory. This technique is useful because it allows you to utilize a powerful technique known as double buffering. In its simplest form, double buffering is viewing one screen while another is being created or manipulated. A more complex application of double buffering involves moving images to the current screen that is being viewed from another which is not being viewed with the @COPY command to create animation or the illusion that images are being drawn instantly on the screen. Also with 64K of VDC Ram, you can tailor your own screens to the plotting size and color cell resolution you desire with the @SCRDEF command. The @MODE and @SCREEN commands are used to choose your make your screen and color cell resolution choices.

3. Is your program going to involve two, or three dimensional manipulations? Basic 8 supports both two and three dimensional graphics regardless of how much video display Ram you have. If your program will require three dimensional graphics you have lots to consider, such as: the viewer's vantagepoint in space, whether images which are along the perceived z axis are viewed as growing & receding or whether they are simply offset on the viewed two dimensional plane. With the power of three dimensional manipulations comes complexity.

4. What are you going to use as your data entry device(s)? The traditional computer data entry device is the keyboard, but there is no reason why you can't use a mouse or a joystick to help the user communicate his or her desires to your program. Unfortunately using the keyboard for user input becomes slightly more complicated when using Basic 8 than it is normally with Basic 7.0. This is because most programs running under Basic 8 are utilizing 80 column graphics screens instead of the C-128's normal text screen. The INPUT command, when being used on the 80 column text screen, places its prompts and the characters you type on the text screen, which of course is not available when you are displaying an 80 column graphics screen. The solution to this dilemma is to either use the 40 column text screen for user input using the INPUT command, or to write your own custom input routine using Basic 7.0's GETKEY command. I usually use the latter, but there is nothing preventing you from using the former. Utilizing the mouse or a joystick from Basic 8 is an exercise in event programming. If you are using a mouse or a joystick you use the @MOUSE function and the @PTR command to read the position of or move an on screen pointer and then depending upon conditions you establish involving pointer position and/or button presses allow the user to choose options or respond to questions. Of course, here again you should design your program according to the needs and characteristics of your program's goal and target audience. For instance: All C-128 owners have a keyboard, but does everyone in your target audience have a mouse? Or, does a certain characteristic about your program or its target audience lend itself to a mouse or a joystick rather than the keyboard?

5. Is your program going to depend upon color as more than just an aesthetic enhancement? Although Basic 8 has fantastic facilities for color, you must decide whether or not the operation of your program is going to depend upon the user having a color RGBI monitor. Many C-128 owners use monochrome monitors (green or amber

Basic 8 Unreview Continued

screens). If your program depends upon color for indications of status or conditions, C-128 owners who do not have RGBI monitors may have a difficult, if not impossible time utilizing your program. Again, this is another question that has to be evaluated on the basis of whom you are designing your program for.

6. Are you going to depend upon graphics primatives to create all of your images or are you going to "pre-create" some images using BASICPAINT? Wallace and Darus included the BASICPAINT graphics editor both as an example of what could be done using Basic 8 and as a tool. Do not overlook BASICPAINT as a valuable tool for image creation. There are times when it is much easier and more efficient to use BASICPAINT to create images for use in your programs rather than using the graphics primatives such as @LINE or @BOX. This is especially true in cases of complex images that do not involve mechanical patterns that can be easily described using Basic. In these cases it is much better to pre-draw images with BASICPAINT and load them into memory as structures or display them directly to a specific graphics screen. However, in cases where your program requires mechanical repetition (e.g. a series of grid patterns) or precise mathematical placement you are probably better off using graphic primatives and the aspects of the language such as mathematical functions, loops, or conditional statements to construct the image.

7. Where are you going to store images once they are created? Basic 8 offers a great deal of flexibility as far as data storage. You are allowed to store images on disk, in VDC memory, or in system memory. The best place to store a specific piece of graphics data depends mostly upon what you want to do with that data within the context of your program. Storing image data on disk allows you to access vast amounts of graphics data but manipulation of that data is hampered by the speed limitations of disk storage. Storing graphics data as structures in either internal system RAM or external system Ram (i.e. in a 1750 or 1700 Ram expansion unit) allows you the most flexibility as far as data manipulation (structures can be cropped, flipped, rotated, inverted, or used as fill patterns etc.) but you are limited in the amount of data you can store and manipulate by the amount of Ram you have in your system. In addition, because of the architecture of the C-128, image data stored as structures, especially large structures, are fairly slow when it comes to transferring them to VDC memory for display or storage. Images stored in VDC memory, however, can be moved or copied almost instantly, however you are severely limited as far as storage space and complex manipulations are concerned.

Once you have answered the above questions you can begin to pseudo code. Pseudo coding is nothing more than outlining the program's operation in English, rather than using the commands and syntax of your programming language. You can be as vague or as specific as you wish, but try not to get bogged down in the actual coding of specific operations. The following is an excerpt of pseudo code I wrote for a Basic 8 game I wrote recently:

```
GETKEY UNTIL KEY PRESSES RETURN
IF KEY PRESSED IS R,G,B,Y,W,P OR UP/DOWN THEN
  CHANGE APPROPRIATE WINDOW COLOR TO REFLECT KEYPRESS
  GET MORE KEYS IN SEARCH OF RETURN
ELSE GO BACK AND GET ANOTHER KEY
ONCE KEY IS RETURN COLOR COUNT=COUNT+1
IF COUNT < 4 THEN GO BACK AND GET MORE CHOICES
ELSE DISPLAY RESULTS OF GUESS
```

Once you have constructed the pseudo code for most of the program, examine the pseudo code and look for patterns or phrases that appear over and over again. These patterns or phrases most likely represent routines or modules that will be called upon constantly throughout the program to perform specific tasks. Here is where I start actually writing my code. Try to make each module perform one and only one specific "low level task", but try to design these modules in such a way that they can be called often in a variety of situations. Of course this sounds easier than it is, but with practice you will begin to develop combinations of Basic 8 commands that you will constantly rely upon in program after program.

Sparrow's Slick Tips *by Sparrow James*

01000011: As noted in the Programmers Reference Guide, interrupts to the 80 column screen should never be attempted as registers programmed before the interrupt cannot be saved nor restored. Our slick tips in issue #15 outlined a method for creating interrupt driven routines when using the 80 column screen. Unfortunately the method as reported fails occasionally. Here is a much better answer to the 80 column screen interrupt problem.

```
      TSX           ;transfer stack pointer to .X register
      LDA  $107,X   ;get the high byte of the return address
      CMP  #$C2     ;is it $C2?
      BNE  OUT      ;if not exit to remainder of system IRQ
      LDA  $106,X   ;get low byte of return address
      AND  #$F0     ;Clear lower 4 bits
      CMP  #$60     ;is it now $60
      BNE  OUT      ;if not exit to system irq
      >>>>>>>>      ;put your irq routine here
OUT   JMP  $FA65    ;finish normal IRQ service
```

This routine should be placed at the beginning of your interrupt wedge. See also the programming example in this issue's installment of the Assembly Line for more details and a programming example.

01000100: Lately I have been toying with the idea of writing my own outline processor for the C-128. One of the things I wanted my outline processor to do was automatically generate roman numerals for topics and details. Unfortunately, the C-128 was designed in West Chester during 1984 and not in Rome during the first century A.D., so I had to teach my C-128 to convert arabic numbers to their roman numeral equivalent. The following Basic 7.0 will accomplish the task for values from 1 to 3999. Numbers greater than 3999 would require characters that are not available within the C-128s normal character set.

```
100 DIM D(13),D$(13)
110 DATA 1,"I",4,"IV",5,"V",9,"IX",10,"X",40,"XL",50,"L",90,"XC",100,"C"
120 DATA 400,"CD",500,"D",900,"CM",1000,"M"
130 FOR I= 1 TO 13
140 READ D(I),D$(I)
150 NEXT I
160 INPUT"ENTER NUMBER BETWEEN 1 AND 3999 TO CONVERT TO ROMAN NUMERALS";N
170 IF N<1 THEN PRINT"PARVUS EST":GOTO 160
180 IF N>3999 THEN PRINT"MAGNUS EST":GOTO 160
190 RC=N
200 FOR I= 13 TO 1 STEP -1
210 IF INT(RC/D(I))=0 THEN 220:ELSE R$=R$+D$(I):RC=RC-D(I):GOTO 210
220 NEXT I
230 PRINT R$
```

01000101: Many new owners of 1581 disk drives have had a tough choice to make: Whether to use the 1581 as device 8 or device 9. Using the 1581 as device 8 allows you to have your fastest and largest drive (in terms of storage) ready and waiting as your default drive, however, it does lead to some difficulty when trying to boot copy protected commercial software which cannot be transferred easily to a 3.5 inch disk. Here is one method that will allow you to "autoboot" some commercial software with your 1581 as device 8 and your 1571 as drive 9. Save the following program to a 3.5 inch disk, preferably a data disk that you use with the application you wish to "autoboot".

```
10 BOOT ON U9
```

Then create an autoboot sector (a utility for creating autoboot sectors can be found on the 1581 test/demo disk) that automatically loads and RUNs the Basic program. This method works with several packages including the Pocket 2 productivity series from Digital Solutions.

Rumor/Opinion/Mayhem *by Loren Lovhaug*

Talk is cheap...ACTION is where it is at!

It started, believe it or not, just eight months after the initial release of the Commodore 128 personal computer in August of 1985. Yes, as early as May of 1986 the first rumors began circulating that Commodore was preparing to dump the C-128. At that juncture, those prolificating the story were convinced that Commodore's decision to begin marketing a slightly cost-reduced version of the venerable C-64 (the C-64c) with a prettier case and GEOS signalled the end of the 128. Of course they were wrong, and Commodore has kept on selling (and selling... and selling) C-128s. But the rumors that the end is near just have never left us.

This is not news to you. After all, as C-128 owners I am quite sure you all have heard the gossip, read the messages on the telecommunications networks, and even studied the utterances of the self-proclaimed prophets who spread dissent and disinformation with little regard for the facts or journalistic integrity in a variety of Commodore publications. But in this column I am not going to take up any space refuting the rumors as I have done before. Why? Because I don't really think it is that important whether you believe those rumors to be true or false, although I do believe that Commodore intends on offering the C-128 (when I say C-128, I mean C-128 and/or the C-128D) for quite sometime. After all whether Commodore stops selling C-128s tomorrow or continues to sell them into the 1990s does not change the fact that you and I still own them. And the very fact that you are reading Twin Cities 128 indicates to me that it is your goal to get the most out of your C-128 investment.

So how do we as C-128 owners get the most out of our investment? Well the first and perhaps foremost way of getting your money's worth out of the C-128 is to use it! A C-128 that sits idle on a desk or worse in a closet is not doing you any good. So get busy! Write some letters, get your budget in order, organize your belongings on your database in case of fire or disaster. DO SOMETHING!

This especially holds true for those of you who have recently purchased other machines such as clones or Amigas. Just because you might have a new toy does not by any stretch of the imagination mean that you should stop exploiting the potential of a solid machine like the C-128. In my mind, the only places anyone can really justify throwing away a couple hundred bucks is when you give it to a worthy charity or you blow it at Vegas; and in this case the C-128 is really a "sure thing". In fact, in some ways the demotion of your C-128 from the role of primary personal computer to a secondary role opens up some exciting possibilities. Consider setting up your C-128 to handle background tasks such running a BBS, or protecting your home against theft, or becoming the center of a home-made robot or just simply managing a continuously available database. Think about it and challenge yourself not to let any piece of computing hardware that you spent your hard earned money on go to waste.

Support those who support you...

Another way to insure that you will continue to reap long term benefits from your C-128 investment is to encourage talented commercial and public domain developers and writers to continue to produce applications for and articles about the C-128. This of course sounds easier than it is in practice. After all, we as Commodore 128 users can't inspire those individuals with talent and imagination to work magic for us? Or can we?

Of course the primary motivator for commercial developers is profit. We have to make sure that it is profitable for commercial developers to expend their time, energy, and money on us. Commodore has sold over 1.5 million C-128s over the past 2.5 years so there is most definitely a healthy installed base of machines to be catered to. But developers need to be made aware that we want their products, and we are willing to pay a fair price for them, and you can only do that with your words and your dollars. In spite of the large number 128s in the marketplace many developer's attitudes are jaded about the Commodore 8 bit market. The reason I hear quoted over and over for their lack of enthusiasm? Software theft. Stated simply, many of them have grown very weary of spending months, and in some cases years working on software only to have the paychecks taken literally away from them by thoughtless individuals. I plead with you, do not pirate software or tolerate others who pirate software. I can't stress this enough: There is no justification for software piracy except evil greed: greed which ironically turns out to be self destructive.

But beyond avoiding the temptation to steal, there are other ways to help convince talented individuals to continue helping us get the most from our 128s. I think one important, and often overlooked method is through open demonstrations of support for those who you feel are doing a good job. If you feel a certain company, developer, or public domain author is doing a good job supporting the 128, I'll guarantee you that it will be worth your while to spend a couple of minutes with your word processor (or sending e-mail on your favorite telecommunications network) telling them that you appreciate their efforts. I'll tell you quite honestly that there have been many times when encouraging letters from readers have chased away my thoughts of "chucking the whole darn thing".

In the final analysis, it is the man (woman) not the machine that achieves. By making your machine work for you, and supporting those who help you make it work you will get your money's worth from your 128 and much, much more!

An Evaluation of Beyond Zork *by Avonelle Lovhaug*

Because our computers get such heavy "practical" use, it is rare that my husband and I take advantage of another aspect of our computers: the entertainment possibilities. However, when our local computer dealer asked me to take a look at a new Infocom game, I decided it would afford us a little fun, and also provide me with some interesting review material.

Beyond Zork is a text adventure game by Infocom, the folks who are famous for this type of game. The storyline is reminiscent of the medieval themes this fare often entails: The Southlands of Quendor are experiencing a dark age, complete with monsters and other menacing creatures, with only one hope in sight, that the fabled "Coconut of Quendor" will fall into the right hands. Your task is to work your way through the adventure, trying to find treasure, avoid being eaten or otherwise destroyed, and solving mysteries. In this text adventure game, instead of using your joystick to win, you guide your character across the Southlands of Quendor, ala Dungeons and Dragons style: following maps and passages. Each time you move, the game tells you exactly what you "see", for example: the inside of a tavern, a precarious ledge, or a mystical castle. In addition, the computer describes the characters you encounter, any distinguishing surroundings, etc. From all of this information, clues are provided. Unfortunately, not all the clues are handed over so easily. Often, the most important information has to be obtained, generally by asking the right questions of characters you encounter. For instance, if I find a scroll which appears to have a magic spell inscribed upon it, not only do I have to remember to ask about it when I encounter someone, but also I have to ask the right person, a magician, or someone who owns a magic shop. Also, sometimes it is important how you word the question.

Beyond Zork includes some interesting features. One of these is the ability to create your own character. You are allowed to create your own mix of six characteristics: endurance, strength, dexterity, intelligence, compassion and luck. Since each situation in Beyond Zork is different, you will find there is an optimum mix for each situation. For me, high intelligence did not work nearly as well for me as strength and endurance; but this may be due to the fact that I never got very far before something terrible ate me.

Another interesting feature of this game is a small section of the screen devoted to a map of the immediate area your character is located. This map gives you instant information on which directions you may move in by showing you available exits. This map is generally quite small, showing only the immediate area, but you can enlarge it some by using the zoom command, which will make each individual map less detailed, but then it will cover a greater area. You may also play the game in standard mode, which make this game like all other Infocom games, without a map, and without many special features. Why anyone would want to play this way is a mystery to me.

I found the vocabulary of this game to be excellent. If I was engaged in a fight, the game understood most every violent attack phrase I could render such as: attack, stab, kill, wield. Also, if I didn't quite complete a command, the game would try to guess what I meant, and if it couldn't, would prompt me for more information. If you make a typo, you are allowed to simply type "oops" and the correct spelling without retyping a line. And certain phrases can be defined on the function keys so that repeat typing can be kept to a minimum. Finally, abbreviations can be used and are very logical. For instance, a "$" will give you your total cash available.

Beyond Zork has several "modes of play" available. If you play in normal mode (not be confused with standard mode alluded to above), each time you enter an area you will see a full description of the place. In brief mode, you only receive a full description the first time you enter an area. Superbrief will only tell you the name of a place, even if you have never been there before. Although I was very challenged by normal mode, apparently some people need even more of a challenge. The program will also send the story as it happens to your printer, if you tell it to "script".

Beyond Zork comes complete with disk, Infocom catalog, The Southland of Quendor map, a book entitled "The Lore and Legends of Quendor" and an instruction manual. Although I never could make heads or tails of the map, there are some fascinating tidbits in the book on legends. Actually, the book is more of a "manual of monsters and bad guys" in Quendor. Some of these monsters have quite impressive names, such as the eldritch vapor. According to the book, these creatures snatch away the possessions of unsuspecting persons. Visitors with no possessions are "snatched" away themselves. Some of these entities do not have such fright inspiring names. The "Christmas Tree Monster" and "Dust Bunny" were my personal favorites. The book also mentions a few good creatures, such as unicorns and also some important places and items. Besides providing the user with background on the story line, the instruction manual also gives information on special commands, and a sample transcript for those persons who are not entirely familiar the text adventure format. There are also some suggestions for troubleshooting possible problems which might occur.

As with many of Infocom's games, a "hints" for Zork can be ordered from Infocom. I found that I enjoyed the game without one, but someone who played it more frequently might desire some help. Either way, the game is a very entertaining addition to the growing selection of C-128 game software. I highly recommend this game.

Dipping Your D *by John D. Clark*

As most of you know, the 128D has a built-in 1571 disk drive. However, Commodore never added dip switches to the 128D to allow the user to change the device number of the built in drive like those found on the standard 1571. This has a limits multiple drive owners. TC-128 to the rescue:

In a standard 1571, two circuits have switches which are normally on (connected) to provide device 8. When the first switch is off, you get device 9, when the second is off and the first is on the drive is device 10 and when both are off you have device 11. In the 128D, these circuits are permanently connected with no switches installed. The problem is finding where these circuits are and cutting them. then connecting a switch in between each end.

To the best of our knowledge the information presented below is correct. Neither Twin Cities 128, nor the author of this article assumes any responsibility for any damages either to you or your computer resulting from any attempts to make the modifications to your hardware outlined below. Also note that effecting this modification on your computer will most certainly invalidate any and all warranties from Commodore.

I want to you to know that this project turned out to be the most difficult hardware modification I've done to date. I have replaced the ROMs in my 128 and 1571, and also upgraded the 128's VDC Rams to 64K. This job is not for the unexperienced or faint-hearted! Please do not try this unless you have confidence in your ability with a soldering iron. But if you I have not scared you off yet, here is what you need:

Parts:
1. A two-position dip-switch or two toggle switches
2. An eight pin socket
3. Some 24 gauge wire
4. A small perfboard (Radio Shack #276-148)

Equipment:
1. A 30 watt soldering iron and solder
2. A knife or sharp implement
3. A continuity tester

First unplug your C-128D from the outlet socket. Remove the cover by first removing five screws, three in back and one on each side facing downward. Slide the metal cover back an inch and lift it off. Looking toward the back, just to the right of the connectors plugged into the center of the motherboard, just beneath the left side of the built-in drive you'll see two pair of metallic spots near PCB locations u112 and u114. If you examine these spots carefully you will notice that each metallic spot is made up of a pair of semi-circles connected together by a thin metal strip.

To install the dip switch, first scrape away the small metal connection between the two semi-circular sections of one metal spots. (If you scrape away the connection between the semi-circular sections of the spot nearest the front of the computer you will make the drive unit 9. This simple fix will allow you to have an external drive as the default drive and require no soldering.) To insure that the connection has been properly broken, use the continuity tester to test your work by placing one probe on each semi-circle. Then if you desire, scrape away the connection between the second set of semi-circles.

You must decide for yourself where you want to place the dip switch. I considered gluing the switch to the top of the expansion port connector where it could be accessable from the back. I decided to simply mount the switch just above the connections, screwing the small portion of perfboard to the screw hole on the corner of the drive. Once you decide where you want to put yours, cut three pieces of wire to the appropriate length with an extra couple inches to spare. With a pair of wire cutters, cut the small perfboard in half and then in half again. You'll end up with a board about 3/4 inch square. Insert the socket's pins into the perfboard and bend them back. Solder two wires to two adjacent pins on one side of the socket, and the remaining wire to one pin across from one of the other wires. Now cut a small piece of wire at one inch long, and solder one end to the remaining pin and the other end to the other pin on that side of the socket (i.e. the one with the wire already attached). The semi-circles on the motherboard next towards the plug-in cables go to ground, so it's not necessary to solder both to the motherboard.

Before soldering anything to the motherboard it makes life a bit easier if you disconnect the power connector cables and lay them aside. Make sure you jot down how the plugs must be oriented so that you can re-plug the connectors properly. Take one of the wires on the side of the socket that has two long wires and spot solder it to the back-most right semi-circle. The other wire on that side of the socket should be soldered to the front right semi-circle. Make sure you have not accidentally soldered across the gap left by scraping away the connector by testing with the continuity tester. I found it was easiest to solder these wires by first melting a bit of solder onto the spot and then immobilizing the wire on the solder and heating with the iron. The last wire should be soldered to either of the left-hand semi-circles (these are the ones which go to ground).

Once the soldering is complete, place the perfboard where you like and insert the dip switch into the socket. You can then test your solders with the continuity tester with the switches both on and off.

In Search of 64K VDC *by Fred Bowen*

Every 8563 (the C-128's 80-column display controller) can access up to 64K of DRAM. Until recently the systems were built with only 16K, more than enough for text purposes. But quite a few C-128 owners have added more display memory to their systems, and as you have read in TC128 newer C-128's (such as the US version of the 128D) have a full 64K of 80 column display Ram built-in. Some folks like the additional memory for the bitmap graphic capabilities (as with BASIC-8 from PATECH), others have used it for Ram disks that have been designed to use it, or for the windowing goodies.

The intent of this article is not to tell you how to get 64K display Ram nor what to do with it. Rather, I want to describe how to tell from a programming viewpoint if a C-128 has it. Many people seem confused regarding this, and it's not surprising, it is not easy unless you know what's going on in that piece of silicon.
Depending upon the type of DRAM used, the 8563 calculates addresses differently. You must tell the 8563 what kind of DRAM you're using. The Kernel does this in its init routine, IOINIT, which is called at power up, reset, and Stop/Restore. It always assumes 16K, and sets bit 4 of register 28 to zero. This works just dandy regardless of the actual amount of memory. This bit tells the 8563 whether the system has bit-wide DRAM or nybble-wide. The chip addresses each type differently, as described below:

```
bit (4116):      A7/A15 A6/A13 A5/A12 A4/A11 A3/A10 A2/A9  A1/A8 A0/A8
nyb (4164/4464): A7/A15 A6/A14 A5/A13 A4/A12 A3/A11 A2/A10 A1/A9 A0/A8
```

Confused? Here's what it basically means. You cannot POKE to a location above $3FFF (i.e. beyond the first 16K) to test for memory higher than that and expect to see it show up someplace obvious, like wrapping. So, how to tell how much memory there is, short of asking the user (who might not know)?

The program below makes use of the chart above, it selects an address that affects more that one Ram location if the memory is only 16K. We simply write to one cell, and see if another cell changed, a cell which we determined should change based upon the screwy 'bit' addressing scheme outlined above. Still confused? Pretend it's magic and just use the program anyway.

```
1 REM  FRED'S NIFTY PROGRAM TO DETERMINE SIZE OF 8563 DRAM - 09/28/87
10 BANK15: AD=DEC("D600"): DA=AD+1: GOSUB900              :REM  SETUP ML
20 POKEAD,28: S=PEEK(DA): POKEDA,63                       :REM  SELECT 64K
30 I=16896:SYSW,I/256,18:SYSW,IAND255,19:SYSW,85,31       :REM  WRITE $55
40 I=16896:SYSW,I/256,18:SYSW,IAND255,19:SYSR,,31:RREGC1  :REM   READ HERE
50 I=17152:SYSW,I/256,18:SYSW,IAND255,19:SYSR,,31:RREGC2  :REM   AND HERE
60 I=16896:SYSW,I/256,18:SYSW,IAND255,19:SYSW,170,31      :REM  WRITE $AA
70 I=16896:SYSW,I/256,18:SYSW,IAND255,19:SYSR,,31:RREGC3  :REM   READ HERE
80 I=17152:SYSW,I/256,18:SYSW,IAND255,19:SYSR,,31:RREGC4  :REM   AND HERE
90 POKEAD,28: POKEDA,S: SYSDEC("FF62")                    :REM  RESTORE 16/64K
100 IF C1=C2 AND C3=C4 THEN PRINT"16K": ELSE PRINT"64K"   :REM  DID IT ECHO?
110 END
900 FORI=0TO13: READA$: POKEDEC("1800")+I,DEC(A$): NEXT
910 R=DEC("1800"): DATA 8E,00,D6,AD,01,D6,60   :REM STX $D600 :LDA $D601: RTS
920 W=DEC("1807"): DATA 8E,00,D6,8D,01,D6,60   :REM STX $D600 :STA $D601: RTS
930 RETURN
```

Of course, this is a bit longer than it has to be. I'm just trying to be neat. To those of you who never diddled with the 8563 directly, well, as you can see you cannot simply PEEK and POKE registers like the VIC chip. I refer you to the programmer's reference manual. If all you want to do is visually tell what's inside a C-128, perhaps in a store or something, try this. In 80-column mode type:

```
POKE DEC("D600"),28:POKE DEC("D601"),63:SYS DEC("FF62"):SCNCLR
```

If the screen looks normal and says READY. and all that, you've got 64K display memory. If the screen is full of 0's and looks like a programming nightmare, you've got 16K (just hit Stop/Restore and let Mr. Kernel fix it). Well, enjoy! I gotta get back to work.

Doing It In Basic 8 *by Loren Lovhaug*

In our last issue I discussed strategies for getting ready to write Basic 8 programs. In this column I am going to take a more utilitarian approach.

BUG SWATTING

The first routines I am going present are fixes to little bugs that made their way into the final release of Basic 8. Actually considering the complexity and scope of Basic 8 it is amazing that more problems have not manifested. I also must report that each of the following "bug fix" routines were developed within 1 hour of the reports the problems and were immediatcly posted on the Quantum Link telecommunications network. This is indicative of the support Lou Wallace and David Darus are providing Basic 8 users and they are to be commended for it.

1. @ANGLE fix: The following subroutine should be called at the beginning of any Basic 8 program which is uses the @ANGLE command to alter perception and view of a three dimensional objects.

```
1000 REM FIX ANGLE BUG
1010 POKE 9653,190:POKE 9657,189
1020 FOR T= 0 TO 10: READ A: POKE 6860+T,A:NEXT
1030 FOR T= 0 TO 9: READ A: POKE 11958+T,A:NEXT
1040 DATA 170,173,0,255,72,173,11,19,76,185,46
1050 DATA 76,204,26,141,0,255,138,234,234,234
1060 RETURN
```

The code above places an assembly language patch routine in RAM that fixes the @ANGLE problem.

2. @ORIGIN fix: The following subroutine fixes a problem with the @ORIGIN command which defines the center of rotation and the vanishing point (in perspective mode) for three dimensional graphics. This subroutine must be called immediately after the @ORIGIN command is used, every time it is used.

```
1100 REM ORIGIN PATCH
1110 POKE 5060, PEEK (5048)
1120 POKE 5061, PEEK (5049)
1130 RETURN
```

The code above adjusts values in the @ORIGIN routine which are not being set correctly as the code stands. Both Lou and Dave apologize for any inconvenience the above problems may caused. Both of these errors have been corrected in an upgrade which will soon be made available to existing Basic 8 owners. Programs written with these patches will function with the new release without modification. Look for specifics on how to obtain upgrades in our next issue.

As discussed in our last issue, perhaps one of the most important decisions a Basic 8 programmer has to make is whether or not his or her program is going to take advantage of the vast potential that 64K of VDC RAM has to offer or not. Of course, most C-128 owners and programmers who have expanded video RAM in their machines will want to make use of the extra display memory. However, you can't utilize 64K of VDC RAM if it is not present. Therefore it is important if you are going to write Basic 8 programs for other C-128 owners to use to determine how much VDC RAM they have in their machine. The most common method for doing this has been to literally ask the user how much VDC RAM he or she has in their 128. This is satisfactory, but it is more efficient to "ask the computer" how much VDC RAM is present instead of troubling the user. Fred Bowen details how to accomplish this outside of the Basic 8 environment elsewhere in this issue, but here is a method to do it in Basic 8:

```
1200 REM BASIC 8 ROUTINE TO SNIFF FOR VDC RAM
1210 POKE 23013,0:POKE 23026,0
1220 @WALRUS,1
1230 @COLOR,0,0,0:@CLEAR,0
1240 @SCRDEF,1,0,0,640,400,16000,0
1250 @SCREEN,1,0:@CLEAR,255,0,0
1260 @SCREEN,0:VR=0
1270 TQ=@PIXEL,0,0,0
1280 IF TQ=0 THEN VR=1
1290 @WALRUS,VR
1300 POKE 23013,134:POKE 23026,134
1310 RETURN
```

Doing It In Basic 8 Continued

Simply call this subroutine at the start of your Basic 8 program and it will set the variable VR=0 if there is only 16K of VDC RAM available or VR=1 if there is 64K of RAM available and perform the appropriate "WALRUS" command for the system. Another item a Basic 8 programmer's might need to determine is whether or not the user has expanded their system RAM using either the 1700 or 1750 RAM expansion units. Here is a routine for doing just that:

```
1400 REM SNIFF FOR RAM EXPANSION
1410 AR=57094:POKE AR,255
1420 IF PEEK(AR)<>255 THEN XR=0:GOTO 1450
1430 POKE AR,0:AR=57088:XR=1
1440 IF (PEEK(AR) AND 16) THEN XR=2
1450 RETURN
```

The above routine uses two locations which are gateways to the RAM expansion controller chip to test whether or not a RAM expander is present or not and if it one is present, how big is the expander. After this subroutine is called, if there is no RAM expander connected to the 128, then XR=0, if a 128K expander is attached then XR=1 and if a 512K expander is present then XR=2. Incidentally, this routine does not use any Basic 8 commands so it could be used in a normal Basic 7.0 program too.

Fancy stuff...

Okay now that I have presented you with a few important, but somewhat blase' tools to get some important Basic 8 programming tasks out of the way, let's explore some sparkle and flash. A lot of people vastly underestimate the power and sophistication that can be obtained with the C-128 if you know how to coax it out of her. Yesterday, I had a C-128 owner over who was considering the purchase of a new computer. As we discussed the matter he said, "The C-128 has all the power I really need, but I can't do some of the really fancy stuff I want to.", as I probed further he confessed, "I'm impressed with GEOS and Bobsterm Pro because of the things they do on the screen, but those programs are written in assembly language, way beyond my grasp, and that is why I am considering the new machines. I want to get at that kind of stuff without having go bug-eyed staring at hexadecimal into the wee hours." Now I was challenged. I sat our friend down in front of the C-128 and said, "Listen, there are legitimate reasons for getting the new big bright machines, but your's is not one of them. I am going to show you how easy it is to get at your dreams with just Basic 8." As I began typing our friend reminded me, doubtfully, that he had only 16K VDC RAM. I said, "Don't worry this will run on your machine". Here is what I typed:

```
100 REM ** SETUP **
110 @WALRUS,0
120 @MODE,0
130 @SCREEN,2
140 @COLOR,2,13,0:@CLEAR,0
150 @BUFFER,1,1024,16384
160 POKE 47,0:POKE 48,68:CLR
170 DEF FNR(X)=INT((RND(1)*X)+1)
180 REM ** DRAW SOME STUFF **
190 @BOX,10,10,0,500,30,0,0,0,1
200 @CIRCLE,320,100,0,75,1
210 @ARC,70,100,0,30,40,-360,180,120,1,0
220 @SPHERE,500,100,40
230 REM ** PLACE RANDOM WINDOWS **
240 DO WHILE A$<>"Q"
250 X=FNR(400)+100:Y=FNR(75)+50
260 X$=STR$(X):Y$=STR$(Y)
270 @STASH,1,1,0,X,Y,100,50,1
280 @WINDOWOPEN,X,Y,100,50,1
290 @COLOR,FNR(15),0,0:@CLEAR,0
300 @CHAR,254,1,8,1,1,2,"WE ARE AT:"
310 @CHAR,254,1,24,1,1,2,"  X:"+X$
320 @CHAR,254,1,40,1,1,2,"  Y:"+Y$
330 GETKEY A$
340 @FETCH,1,X,Y,0
350 LOOP
360 @TEXT
```

What this 27 lines of Basic code does is demonstrate how Basic 8 can be used to create the same kind of windowing found in GEOS and Bobsterm or the "new-wave" machines. If you type this program in and run it, notice how fast the windows pop-up and recede, on just a 16K VDC RAM machine! (It can be done even faster with 64K of VDC RAM)

Here is how the program works: Lines 110-170 setup the program. First we declare that we are going to be using only 16K of VDC RAM (@WALRUS,0) and then tell the computer that we are going to work with a 640 x 176 pixel 8 x 8 color cell screen (@MODE,1 and @SCREEN,2). Then in line 140 we declare that we want to have a screen with a blue background, yellow foreground, and a black border without a background pattern (@COLOR,2,13,0:@CLEAR,0). In lines 150 and 160 we set up a buffer to store the screen data that will be over written when our window is placed on the screen. By doing this, our windows in effect become non-destructive, because when we wish the window to

Doing It In Basic 8 Continued

appear we simply retrieve the data that previously occupied that area of the screen before the window was placed on the screen. Line 150 actually declares this buffer to be a 15K area in BANK 1 of the C-128's random access memory from 1024 to 16384. In reality this buffer is much larger than we actually need. Because BANK 1 is usually used for Basic variable storage we must tell the C-128 not to overwrite our buffer with variable storage. This is done by resetting the start of Basic variable storage pointers at locations 47 and 48 to 16385 in low-byte/high-byte format in line 160. Then in line 170 I define a user defined function for generating random numbers. This will be used for placing our window at random locations on the screen.

Lines 190-220 draw some objects on the screen for demonstration purposes, and then the core of the program actually begins with the DO-LOOP that begins on line 240. In line 250 we use our random number function to pick a random horizontal and vertical location for our window. Then in line 260 I store these values as string values so that later they can be displayed on the screen using the @CHAR command. Line 270 contains the @STASH command that actually stores the 100 x 50 pixel section of the screen that will be overwritten by the window. Line 280 opens a window at the random location the computer chose and line 290 chooses a random background color for our window and clears it. Lines 300-320 place a message describing our location in the window using @CHAR commands. Then line 330 waits for the user to press a key. Once a key is pressed the previously stored section of our screen is retrieved using the @FETCH command and the window in effect disappears. Finally, if the key pressed is a Q then our loop in terminated and we return to the text screen, otherwise we repeat the process of displaying another random window on the screen. And that is all there is to it! Experiment with this program, see ya next time with some more Basic 8 wonders!

Making Superbase "Perform" *by Randy Margolis*

One of the mightiest commands in Superbase 128's rich programming language doesn't really do anything at all. I'm talking about the PERFORM statement. What it does do is allow you to reduce your programming effort by executing previously defined strings of characters with themselves for the Superbase 128 commands.

Consider: You want to create (or update) a sub-list of all customers who are 'prospects' and who have red hair. A line of code to do this task might look something like this:

```
10 FIND "list" WHERE [stat] IS "=prospect";[hair] IS "=red"
```

If you want to write a list update program, you will have many such lines, which, besides being a pain to create would not really be easily revised. This task is precisely the reason the PERFORM command was included. One effective programming method involves building string arrays of the variable data in program lines like the one above whose subscripts match menu choices made by the user. The fixed portions of the command, or traits common to all instances, can then be assigned to single string variables. In addition, you can use good old DATA statements to hold other parts of the required string.

The following example program utilizes a neat trick gleaned from that indispensible reference work "Superbase, The Book", put out by Precision Software (publishers of the program), and available at a well stocked Commodore software store. It involves using the MEMO and HELP functions to build menu screens. You just go into the MEMO writer, create your customized numbered menu, and call it with the HELP command. This approach is much easier than coding a bunch of DISPLAY @ lines and, if you include a strategically placed ASK statement (which appears right on the HELP menu!), you'll save yourself lots of tedious work.

The Signal subroutine (lines 200 - 220) first clears the screen, and then displays a running count of the process. When the FIND or SORT is complete, a tone sounds and you are prompted to press RETURN to continue. (A hint: always put your subroutines at the beginning of the program and use the first program line to skip over them. This trick, valid for all BASIC programs will increase your program speed, sometimes a significant amount.)

Line 270 assigns a part of the PERFORM string which is common to all choices to the string al$. Hereafter, you just have to include this common aspect of the tasks in the string building that is to follow. Lines 280 through 309 are an example of using DATA statements to fill arrays, in this instance, this is where you name your key lists and sorted lists. Line 305 takes care of other constant parts of the PERFORM string, including defining the double quote mark as sp$. This helps in line 307, in which all the filenames are enclosed in quotes (required for Superbase 128 command lines).

Making Superbase "Perform" Continued

Lines 310 through 320 actually do the work of defining the PERFORM strings. By carefully concatenating all the parts that have been defined previously, you can see that distinct and familiar Superbase 128 commands have been constructed. After making sure the proper file is selected (line 350), the technique mentioned above is invoked by line 360. The MEMO screen "menu" must have been constructed previously, and will contain a list of numbered tasks, one of which is ASKed for in line 370. Note that the ASK statement overlays a line at the bottom of the menu screen, a really nice touch for your user, and a wonderful time saving method for you (the programmer).

Finally, line 420 PERFORMs the Superbase 128 task, and sends you back to your menu, whose last choice should always be to quit the program (that is, go back to the regular Superbase 128 menu). With a little imagination and planning I'm sure you can expand upon this skeletal program to fit your needs. When it comes to constructing complex Superbase 128 commands with variable data, there is almost nothing the PERFORM statement cannot do.

```
170 GOTO 260
180 rem Signal Subroutine
200 DISPLAY @0,147
205 COUNT x:DISPLAY @15,4&3,0x:"Records processed in:" ;kl$(q)
210 VOL 5:SOUND 1,4360,25
220 DISPLAY @20,0"Press <RETURN> to Continue.":WAIT
230 rem
260 DIM kl$(14),pa$(14)
270 al$="@22[common field 1]@51[common field2]"
280 DATA sortlist1,list2,list3,etc...
300 FOR c = 1 to 13:READ kl$(c):NEXT
305 sp$=CHR$(34):p1$="FIND ":p2$=SORT ALL "
:w$="WHERE"
307 FOR i = 1 to 13:kl$(i)=sp$+kl$(i)+sp$:NEXT
310 pa$(1)=p2$+"ON [sort field 1] to kl$(1)
315 pa$(2)=p1$+kl$(2)+w$+"[find field 1] is "+sp$+
"parm 1"+sp$
320 rem more pa$ definitions go here
350 FILE "yourfile"
360 HELP "menu"
370 ASK @46;q
380 IF q<1 OR q>14 THEN 370
400 IF q=14 THEN MENU
420 PERFORM pa$(q)
430 GOSUB 200:GOTO 360
```

Hacking Fast Hack'em *by Frank Prindle, forward by Loren Lovhaug*

The 1571 upgrade Rom has indeed been a blessing for Commodore 128 owners as it has transformed the 1571 into the disk drive Commodore designers always meant it to be. But perhaps the most amazing thing about the 1571 upgrade Rom is that very few problems with program incompatibility have arisen. Considering the fact that Commodore disk drives are very complex "intelligent" devices that resemble more a computer than a mass storage device, the upgrade is a real tribute to the prowess of Fred Bowen and the others involved in its design. But this is not to say there are not a few incompatibilities. Perhaps the most well-known incompatibilities were with the original release of Big Blue Reader - CP/M and several of the latest versions of Fast Hack'em. In the case of Big Blue Reader, S.O.G.W.A.P. software has already corrected the problem with a new release which can be obtained by contacting them in writing at: S.O.G.W.A.P., 611 Boccaccio Avenue, Venice CA 90291. However, in the case of Fast Hack'em we have not been able to verify if Basement Boys Software intends on releasing a fixed version or if Basement Boys are even still in business.

A few weeks ago I discovered a set of messages on USENET authored by Mr. Frank Prindle which describes how to modify Fast Hack'em so that it functions correctly with the 1571 upgrade Rom. I must tell you that although I was very pleased to see this development since I like the speed and ease at which Hack'em can be used to make archival copies of data and programs I am very leery about the publication of this information. As most of you are probably aware, Fast Hack'em is perhaps one of the most well known "pirate tools" around. I really hate the idea that through the publishing of this information we could be indirectly and most certainly unintentionally assisting those who steal software. In fact, I have on many occasions seriously considered pulling this article for that reason. However, after discussing this article with many software developers and people who work in the

Hacking Fast Hack'Em Continued

Commodore computing industry, I have decided to run this article since there is some doubt whether there will actually ever be an upgrade of Fast Hack'em. It is my hope that this information will be valuable to our readers who have purchased Fast Hack'em and use it for legitimate purposes. Again I implore you not to take this information and use it improperly, for the end result will most certainly serve to stifle Commodore software development and leave yourself open to criminal prosecution. The following is an excerpt from Mr. Prindle's messages. These fixes have been tested and verified by Fred Bowen at Commodore. We suggest that before anyone attempts to make the modifications described below that you first read them through carefully and do them on a backup copy of Fast Hack'em. Twin Cities 128 assumes no responsibility for any damages that might result from attempting this modification or the use of Fast Hack'em in any way. Our intent is only to support the legitimate users and the legitimate use of this program.

The following files from versions 3.0A through 4.1 need to be patched: "64", "FASTBOOT V2", "SINGLE", "V2 NIB" (3.0A only), "S NIB", "128 SINGLE". The patches are necessitated by 3 changes in the 1571 Rom:

1. Rom location $c000 contains a new value. The 'fastboot' processing checks this value to determine if it is appropriate to install and utilize the fast booting code. Finding the new value there, which does not match any known drive type, it bypasses the fast booting capability. This may be restored by making the processing check default to 1541/1571 mode if $c000 is unknown.

2. Ram location $01b4 is now used by the DOS as a flag. When code is uploaded from the C64 into drive Ram $0174-$01b9, the last one or two instructions is overwritten by the DOS in this location (a $01 is overwritten); thus the uploaded program executes incorrectly. This may be corrected by installing a harmless instruction with it's operand at this address: ORA ($00,X). The remaining code is moved down or prior code is moved up, as necessary to fit under $01ba.

3. The -05 Rom clears bit 1 ($02) of the disk controller's Peripheral Control Register (PCR) at $1C0C in the drive whenever it is not reading or writing the disk. Apparently, this bit, when cleared, disables the generation of "byte ready" signal (as bytes are read and assembled by the disk controller) which appears as bit 7 of port $180F. This only seems to occur when the drive is in "fast-serial" (i.e. 128) mode. The "128 single" module assumed that this bit would be set and did not set it. Additional code must be added to ensure that this bit of the PCR is set during both read and write portions of the copy operation.

(Note: notice that the "2-Drive" modules are not afflicted by these problems, and do not require modification).

```
Module: "64":
Symptom: Main menu boots up slowly on a C64.
Cause:   Signature change in Rom causes drive not
         to be recognized as a 1541 or a 1571.
         Defaults to no fast boot.
Fix:     Alter code so that unrecognized signature
         defaults to 1571.

Module Loads At: $102 to ? (i.e. in the stack)
Patch:
0135: BNE +9 ;Default unknown $c000 to 1541/1571
             ;i.e. the second byte of BNE
              instruction is 9

Hint: Use "DISK DOCTOR" or any other sector editor
to change first sector of file, position 55=208,
position 56=9 (these numbers are decimal).

Module: "FASTBOOT V2"
Symptom: Even when FASTBOOT is enabled, modules
         don't load fast
Cause:   Signature change in ROM causes drive not
         to be recognized as a 1541 or a 1571.
         Defaults to no fast load.
         Alter code so that unrecognized signature
         defaults to 1571.

Module Loads At: $C800 to $CFA6-1

Patch:
C878: NOP       ;Default unknown $c000 to 1541/1571
C879: NOP
C87A: NOP
C87B: NOP
C87C: NOP

Hint: Must use "DISK DOCTOR" or other sector editor
to change first sector of file, positions 124 thru
128 to 234; This file is not loaded by name, but by
position on disk, so it must be modified in place!
If you move it, it won't be found.

Module:  "SINGLE"
Symptom: 1541 Single Drive Fast Copy hangs up
Cause:   Fast uploader code uploaded to $1b4 in the
         drive is overwritten by the new DOS, which
         has allocated $1b4 for it's own use.
Fix:     Alter uploaded code so that it doesn't
         matter what is in $1b4.
```

Hacking Fast Hack'Em Continued

```
Module Loads At: $07F8 to $13F0-1

Patch:
13ED: ORA ($00,X) ;skip over drive $1b4
13EF: NOP
13F0: JMP ($01A9)

0FAD: $B6     ;change refs to $1b3 -> $1b6
1008: $B6
106C: $B6
10A9: $B6
10EB: $B6
10FE: $B6
```

Hint: File must be loaded, altered, and resaved with a ML monitor. When re-saving file, save $07F8-$13F3 (i.e. 3 more bytes than loaded to compensate for added code)

Module: "V2 NIB" (Version 3.0A only)
Symptom: 1541 Single Drive Auto Nibbler hangs up.
Cause: Same as for "SINGLE"
Fix: Same as for "SINGLE"

```
Module Loads At: $07F8 to $17A9-1
Patch:
```

First, transfer $15A2-$15D3 to $15A0 (i.e. move that block of code backward two bytes). Then:

```
159B: LDA #$04     ;correct for move-back
15C8: BNE $159B    ;correct for move-back
15CF: ORA ($00,X)  ;skip over drive $1b4
15D1: JMP ($01A9)

119D: $AA          ;change refs to $1ac -> $1aa
1266: $AA
12B7: $AA
12C2: $AA
12CA: $AA
131A: $AA
131E: $AA
```

Hint: Easiest to do this with a ML monitor; Because code is moved back instead of ahead, just save same area as loaded: $07F8-$17A9.

Module: "S NIB":
Symptom: 1541 Single Drive Nibbler hangs up.
Cause: Same as for "SINGLE"
Fix: Same as for "SINGLE"

```
Module Loads At: $07F8 to $179B-1
Patch:
```

First, transfer $1769-$179A to $1767 (i.e. move that block of code backward 2 bytes). Then:

```
1762: LDA #$04      ;correct for move-back
1762: LDA #$04      ;correct for move-back
178F: BNE $1762     ;correct for move-back
1796: ORA ($00,X)   ;skip over drive $1b4
1798: JMP ($01A9)

136F: $AA           ;change refs to $1ac -> $1aa
1438: $AA
1489: $AA
1494: $AA
149C: $AA
14EC: $AA
14F0: $AA
```

Hint: Easiest to do this with a ML monitor; Because code is moved back instead of ahead, just save same same area as loaded: $07F8-$179B.

Module: "128 SINGLE"
Symptom: 1571 (128 mode) Single Drive Fast Copy hangs up.
Cause: New ROM -05 clears bit 1 of PCR when not reading from disk. This inhibits ability to detect "byte ready" by testing the sign bit of $180F. 128 SINGLE assumes that this bit is set when it gets control from the DOS and hangs forever waiting for "byte ready".
Fix: Modify the current subroutine which delays while waiting for drive motor to come up to speed so that it achieves a similar delay, but also ensures that bit 1 of the PCR ($1C0C) is set. This may seem like a silly place for a patch, but there is no other available patch area.

```
Module Loads At: $1C00 to $2E24-1 (in 128 mode)

Patch:
2D0B: LDA #$00     ; modified drive motor delay
2D0D: TAX
2D0E: TAY
2D0F: LDA $1C0C    ; loop begins
2D12: ORA #$02     ; set bit 1 of PCR
2D14: STA $1C0C    ; inside loop for more delay
2D17: NOP          ; still more delay
2D18: DEX
2D19: BNE $2D0F    ; end of inner loop
2D1B: DEY
2D1C: BNE $2D0F    ; end of outer loop
2D1E: RTS
```

Hint: Easiest to do this with a ML monitor; Because code is not moved and no new code is added, save same area as loaded: $1C00-$2E24.

Sparrow's Slick Tips *by Loren Lovhaug*

01000100: The other day I downloaded a public domain program for the C-128 which in effect turned the computer into a scientific calculator. When this program is executed the author uses character graphics to draw a conventional calculator, complete with keys, on the 80 column screen. The author did a fine job creating the calculator image with one exception: the author used the standard C-128 numeric characters for his calculator display instead of the more realistic squarish ones that are found on most LED and LCD displays. Since the squarish calculator digits are not available in the C-128's character set I can forgive this person, but because it is so easy to redefine characters on the C-128 (especially the ones on the 80 column screen) I set about adding the following BASIC code to the calculator program to make the effect complete (Note: the first four lines are added for demonstration purposes here and can be eliminated):

```
100 SYS 52748:REM                          * INIT 80 COLUMN CHARACTER SET
110 PRINT CHR$(142);"0 1 2 3 4 5 6 7 8 9":REM  * PRINT UPPER CASE SET DIGITS
120 PRINT CHR$(14);"0 1 2 3 4 5 6 7 8 9":REM   * PRINT LOWER CASE SET DIGITS
130 WR=52684:WM=52682:REM                  * WR:WRITE REG, WM:WRITE MEM
140 FOR I= 13056  TO 13200 STEP 16:REM     * FROM 1ST BYTE OF THE 1 TO
150 FOR J= 0 TO 7:REM                      * 1ST BYTE OF 0, 8 BYTES EACH
160 L=I+J:REM                              * 1ST BYTE + ROW = LOCATION
170 READ D:REM                             * READ BYTE FOR THIS ROW
180 HB= INT(L/256):REM                     * HB: HIGH BYTE OF LOCATION
190 LB= L-(HB*256):REM                     * LB: LOW BYTE OF LOCATION
200 SYS WR,HB,18:REM                       * WRITE HIGH BYTE TO REG 18
210 SYS WR,LB,19:REM                       * WRITE LOW BYTE TO REG 19
220 SYS WM,D:REM                           * WRITE DATA BYTE TO LOCATION
230 NEXT J:REM                             * DO ANOTHER ROW
240 NEXT I:REM                             * DO THE NEXT DIGIT
250 DATA 126,102,102,102,102,102,126,0:REM - CHARACTER DATA FOR 0
260 DATA 24,24,24,24,24,24,24,0:REM        - CHARACTER DATA FOR 1
270 DATA 126,6,6,126,96,96,126,0:REM       - CHARACTER DATA FOR 2
280 DATA 126,6,6,126,6,6,126,0:REM         - CHARACTER DATA FOR 3
290 DATA 102,102,102,126,6,6,6,0:REM       - CHARACTER DATA FOR 4
300 DATA 126,96,96,126,6,6,126,0:REM       - CHARACTER DATA FOR 5
310 DATA 96,96,96,126,102,102,126,0:REM    - CHARACTER DATA FOR 6
320 DATA 126,6,6,6,6,6,6,0:REM             - CHARACTER DATA FOR 7
330 DATA 126,102,102,126,102,102,126,0:REM - CHARACTER DATA FOR 8
340 DATA 126,102,102,126,6,6,126,0:REM     - CHARACTER DATA FOR 9
```

01000101: A little known fact about Basic 7.0's marvelous PRINT USING command is that the format string portion of the command does not necessarily have to be a string constant. Indeed the format string portion of the PRINT USING command can be a string variable. By using variables in place of string constants in your Basic programs you can add a great deal of power and flexibility to your output formatting routines and save yourself a lot of typing. Consider the following example:

```
100 FM$(0)="    ################       ######"
101 FM$(1)="########## ############## ###.##"
102 FM$(2)="  TOTAL HOURS WORKED:   ###.##"
103 DATA REGULAR STAFF,HOURS,0,BIG,BILL,47.23,1,WILLIAM, WONDERFUL,28.6,1
104 DATA MAX, HEADROOM,12.2,1,SPECIAL STAFF,HOURS,0,SUPER,MAN,1.6,1
105 DATA CAPTAIN,KIRK,2.5,1,PEEWEE,HERMAN,1.2,1,93.33
106 FOR I= 1 TO 2
107 READ A$,B$,FM
108 PRINT USING FM$(FM);A$,B$
109 FOR J= 1 TO 3
110 READ A$,B$,HR,FM
111 PRINT USING FM$(FM);A$,B$,HR
112 NEXT J:NEXT I
113 PRINT USING FM$(0);"","-----"
114 READ HR:PRINT USING FM$(2);HR
```

Loren Lovhaug is the founder and Managing Editor of Twin Cities 128. In addition to running Twin Cities 128, Loren also is the C-128 Sysop on Quantum Link's Commodore Information Network, and the Telecommunications Columnist for RUN magazine. Loren also hosts a weekly C-128 real time conference on the GEnie telecommunications network and writes C-128 software in his spare time.

Loren lives in beautiful Minneapolis, Minnesota with his wife Avonelle and his two sons, Graham and Lewis. Loren calls himself "a prototype 21st century worker" because he believes that in the next century decentralized labor in home offices will be the norm, rather than the exception. Loren prides himself in his ability to earn a living while staying home and caring for his children and sees his unique occupation as the ultimate solution to the dilemmas often faced by dual income families with child care needs.

If you are interested in contacting Loren, or any of the authors of articles in this book, or are interested in obtaining Twin Cities 128 subscription information, write:

Twin Cities 128, P.O. Box 4625, Saint Paul MN 55104

Twin Cities 128 Compendium Book #1

Published by Voyager Mindtools Inc.

Photography by Randy Margolis
Photo processing by Bruce Jaeger