open apple gazette

Sixth Edition **Volume 1, Number 6** **May 1983**

Business BASIC:

West Coast Computer Faire Report

by Richard Hart

Apple /// owners, you now have:

- 8" drives from 3 manufacturers (SOS and CP/M compatible)
- 3" drives from 2 manufacturers
- IBM 3270, 2780, 3780 emulation
- Proto cards
- Printer cards
- SOS-to--IBM--to SOS utilities
- New software galore

Here's a report from the West Coast Computer Faire in March.

Imagine backing up a whole ProFile on 4 floppies! The Faire featured a hot contest between two different manufacturers to sell 8" and 3" drive controllers for Apple ///:

Burtronix
1667 N. O'Donnell Way
Orange, CA 92667
(714)974-6171

Megabyter
11722 Sorrento Valley Rd.
San Diego, CA 92121
(619)452-0101

Both were selling the controllers, with drivers, for $299 until the end of March.

BURTRONIX designed some of the hardware for the Apple /// CP/M card. Their card was up and running under CP/M and SOS side by side at the show, using Tandon 1/2 height drives. Their controller will daisy chain 4 8" drives off of a single controller card! They claimed that MEGABYTER (a division of SVA, Sorrento Valley Associates) did not have its systems demonstrating CP/M because they couldn't--too many bugs. "Not so," said MEGABYTER, "Ours will run CP/M." SVA, on the other hand, had an Apple /// running off of dual 3 1/2" Sony drives! BURTRONIX claims it is writing a driver for 3" disk drives this week. Both systems claim support of Backup.

The best price I could find on drives was $950 for two Tandon 1.2 MB half-heights, power supply, enclosure, and cables. If you go for one drive only (which makes sense for just backup use), you could be into the controller card for $299, and a single 1.2 MB 8-incher for $600.

(But why would anyone want to run CP/M, anyway? That would mean you'd be forced to use WordStar.)

A MicroSci representative told me the company had decided against marketing an 8 inch drive for either the Apple II or ///.

Elcom Systems Peripherals has a new Netcom Communications card which will allow the Apple /// to communicate with:

- Any mainframe using IBM 2780, 3780, or 3270 emulation
- Wang, DEC, ICL, Prime, Amdahl, Facom, and Nat Semi systems
- SNA programs
- OMNINET

It costs $1195. The company reps came to the West Coast Computer Faire to sell the cards, but, at the last minute, Apple asked them not to. Apple is making a bid to buy the technology and market the boards itself!

Elcom Systems Peripherals
439 Harrison St., Suite A
Corona, CA 91720
(714)734-8220

Miscellaneous other stuff at the Faire included:

- A spectacular parallel card from Burtronix: 16 color printer/graphics dump and much more. (Comes with CP/M graphics utilities, too, but why would anyone want CP/M? For chrisakes, it just got graphics a month ago, anyway.) $125.00

- A spectacular protocard from Burtronix: 1 serial and 2 parallel ports with drivers to allow interface with any hardware device you chose, and room on the board for tons of chips. 6522 compatible, 6809 adaptor, too. Fully SOS rigged. $150.00

- SOS--IBM--SOS 8" transfer utility for all files. From SVA. $150.00

Chuck Colby, of Colby Computers, announced a box into which an Apple //e motherboard could fit to become a portable. He said he is considering doing the same for the Apple ///.

The Faire featured significant numbers of mice running under word processors on machines other than Lisa. Lisa was there, however. Even though you couldn't see it for the crowd. This year's Faire was far better than last year's. Especially for Apple /// users.

- /// -

Disk Formatting Bug Cured

by Don Norris

Have you ever had your /// give you the VOLUME NOT FOUND message when you have tried to write to a diskette you know you have previously formatted. Well you are not alone.

This error message was occuring with 3 out of every 10 diskettes which I had formatted for copying club programs onto. Using another /// produced the same problem. Next thing to check was the diskettes themselves. Since I was using bulk pack generic diskettes with no labels, I tried the fancy package brand name diskettes. 3 out of 10 of these, produced the VOLUME NOT FOUND error message.

Checking with a few other /// owners, I found they were also having the same problem. I informed some of the people I knew at Apple to see they were aware of any formatting problem. Yes, there was an occassional problem and they were working on a solution.

Well the solution is now available and it has been sent to all the dealers. On the SOS Revision Utility Version A01, is a new Format Driver Version 1.3 to put on your System Utilities Program diskette.

Does it cure the formatting problem? Sure does. After installing the new format driver onto my System Utilities Program diskette, I formatted 100 bulk pack generic diskettes with only one of them being a problem, and it turned out to be a bad diskette.

If your dealer does not have this utility, order it from the Original Apple ///rs for $10.00.

- /// -

Beginning Business BASIC Lesson #2

By Stan Guidero

One of the powers that a computer has over a calculator is the ability to repeat things over and over. A calculation that would take a person several hours to do may only take minutes or even seconds to accomplish with a computer and may be repeated as often as you wish. One of the commands that allows you to do this is the GOTO command. That's not a misprint. GOTO is spelled as one word. Let's get Business BASIC up and running on our ///s and type in this program:

```
NEW

10 PRINT "Over and "
20 GOTO 10

RUN
```

Hold down the CONTROL key and press 'C' to stop the program.

After line 10 prints the string "Over and ", line 20 tells the computer to return to line 10 and do it again. Pressing Control-C causes the program to exit the endless loop and halt.

The command GOTO is considered a non-argumentative statement. It doesn't fool around. It goes exactly where you tell it to, no questions asked. As a review of lesson one you might try experimenting with the string in line ten "Over and ". Try placing a comma or a semicolon at the end of line ten.

As a more practical example type in this program: (First we type NEW to clear the memory of our old program.)

```
NEW
 5 HOME
30 INPUT "Type in first number: ";A
40 INPUT "Type second number: ";B
50 LET TOTAL=A+B
60 PRINT "The sum of the two numbers
          is: ";TOTAL
70 GOTO 30

RUN
```

Line 5 clears the screen while lines 30 and 40 accept the input for the variables A and B. As you probably noticed this is a new twist to our INPUT statement. It's actually a combination INPUT and PRINT command. Line 50 does the math by adding variables A and B together and places the answer in variable TOTAL. Line 60 then prints the answer to the screen and finally line 70 tells the computer to go to line 30 for another input. Unfortunately, we are now in one of those endless loops and the only way out, short of turning the computer off, is to use CONTROL-C. If the program seems not to stop you may also have to press RETURN.

Whenever you write a program you should try to make it as easy to use as possible. Using CONTROL-C to stop a program is not the recommended method. There should be a more eloquent way, and there is. This is where our next new command comes in. The IF-THEN. Let's change some of our existing program by adding the following line:

```
10 PRINT "To stop the program, type a 0
          for the first number."

20 IF A=0 THEN END
```

LIST the program to make sure that everything is there. Now type RUN.

The IF-THEN statement in line 20 is what is called a conditional branch statement. If the variable 'A' is equal to zero, then the program ends. If it doesn't the computer goes to the next line and continues on. One of the quirks of BASIC is that there is more than one way to do the same thing. We could have typed any of the following in instead:

```
20 IF A<>0 THEN GOTO 30:ELSE END

20 IF A<>0 THEN 30:ELSE END

20 IF A<>0 GOTO 30:END
```

These all do the same thing. The <> means 'does not equal'. Using the first line 20 we are saying that if 'A' does not equal zero then go to line 30 and continue, otherwise end the program. The ELSE command is an optional command and as you can see from our last example ELSE is only needed for clarity. I should also mention that the LET command isn't needed in line 50 as it too is also for clarity. Getting back to our IF-THEN. Which should I use you ask? The one that uses the least amount of memory. How do I know which one uses the least amount of memory? Count how many characters are in each line. The computer will have to go through and read each character one by one. The more there is to read the longer it will take. So use the least to accomplish the most. Our original line fits the bill nicely. Incidentally, I will return in the next lesson with an explanation of logical expressions like the aforementioned 'does not equal'.

To show you another example of the repetitive capabilities of the computer and introduce you to two more commands, let's type in this program. (If you wish to save our little program type: SAVE ADDIT. This will save the program under the name ADDIT.)

```
NEW

 5  HOME
10  PRINT "TRIP COST"
20  PRINT
30  PRINT "Miles","MPG","Time","Fuel
          Price","Total"
40  PRINT "Traveled","","for
          trip","per.gal.","Expense"
50  PRINT
          "--------","---","--------","-
          -------","-------"
100 REM
          ------------------------------
          -------------------
110 REM        Read values from data
120 REM
          ------------------------------
          -------------------
130 READ MILES , MPG , TYME$ ,
          FUELPRICE
200 REM
          ------------------------------
          -------------------
210 REM        Do calculations
220 REM
          ------------------------------
          -------------------
230 TOTAL = MILES / MPG * FUELPRICE
300 REM
          ------------------------------
          -------------------
310 REM        Print it out
320 REM
          ------------------------------
          -------------------
330 PRINT MILES , MPG , TYME$ ,
          FUELPRICE , TOTAL
400 REM
          ------------------------------
          -------------------
410 REM        Do it again
420 REM
          ------------------------------
          -------------------
430 GOTO 130
500 REM
          ------------------------------
```

```
                   -------------------
     510 REM        Here's the data
     520 REM
                   ------------------------------
                   -------------------
     530 DATA 125,25,"1:30min",1.33
     531 DATA 245,25,"2:45min",1.33
     532 DATA 578,27,"6:50min",1.33
     533 DATA 35,23,"45min",1.34
     999 END

     LIST
     RUN
```

The REM statements make this program look much larger than it is. REM statements are used to help the programmer remember what he was thinking of when he wrote that section of the program. REM statements (short for REMARK) are not acted on by the computer.

Lines 10 thru 50 are used for the header. Line 130 reads the data from line 530 one data element at a time and assigns one to each variable in order. '125' is assigned to MILES, '25' to MPG and so on. Next, line 230 does the calculations and line 330 prints out the results. Line 430 sends the computer back to line 130 to retrieve more data until there is no more left. When the program runs out of data it stops with an OUT OF DATA ERROR message. Try adding a few data lines of your own, using the other examples for ideas.

The variables that you use must be of the same type as the DATA. You must use numeric variables for calculations or string variables for string characters. Notice that a string variable was used for the 'time'. All strings must be in quote marks. Each variable and each data element must be separated by commas. Data lines must not be followed by a line with a statement that can be acted on by the computer. If you use an integer variable like MILES% and it reads a real number (like 1.33) it will round it off. Also if the number is larger than 32767 the computer will come back with an ILLEGAL QUANTITY ERROR and stop.

You may have noticed that I spelled the time variable as TYME$. TIME$ is a reserved word like PRINT or GOTO; it controls the onboard clock in your Apple ///.

There is one more command that is used with the READ DATA statement and that's the RESTORE command. If you wish to use the same data over again in the program the RESTORE command is placed at the end of the data like this.

```
 700  DATA 124,24."1:34MIN",1.33
        710  RESTORE
```

To try to give you a better idea of how this works, imagine that there is a pointer that is first set at the first piece of data. As the READ command goes along, the pointer is moved one DATA element at a time until it reaches the end. It will then halt program execution and print 'out of data error' on the screen unless a RESTORE command tells the pointer to go back to the beginning.

In the next issue we will cover The IF-THEN-ELSE and IF-GOTO statements in more detail. Happy programming.

- /// -

GAMEPORT III

Numerous inquiries have been received from Apple /// owners about using paddles or joysticks with the /// in Lobotomy Mode (Emulation).

Most of you are well aware that paddles or joysticks will not work with most Apple][games in the Emulation Mode. T.G. Products with their special Emulation Diskette and joystick made some of the games usable on the ///. However, this was only a partial answer.

Alan Silver at MICRO-SCI has solved the problem!!!! The solution is called GAMEPORT III. GAMEPORT III is a card which fits into one of the four slots on your Apple ///. On the card is a connector for an Apple][type joystick. Alan also modified the emulation diskette so that some of the games which previously would not boot in the /// in Emulation mode now will. As far as we no there are NO games which will not work with this modified emulator and the GAMEPORT III.

The Gameport III's are available to members at a special reduced price. Just another reason to belong to the **Original Apple ///rs.**

- /// -

Word Juggler And Prowriter

By: Rod Whitten

For those Apple /// owners that have Word Juggler and a dot matrix printer, the following is a guide to some of the available features not supported by the commands on the templates. Specifically it has been tested using the Prowriter, but should be applicable to most other dot matrix printers.

The C Itoh 8510A (Prowriter) and the NEC 8023 printers utilize control/escape codes similiar to those for the Epson printers written up in previous editions of the Open Apple Gazette. The NEC 8023 is to be a parallel version of the slightly faster (120 CPS) Prowriter, which has serial and parallel interfaces built-in. As the pins on the ///'s RS-232 port match exactly with the Prowriter, so a standard pre-made cable from an electronics store works without modification.

Word Juggler has a number of escape commands already on the keyboard templates. Most of these are supported with the above printers; however, the change of pitch and enlarged characters are not. In order to accomplish these changes, one must use the PRINTER CONTROL which is "ESCAPE p". The next line then must contain a $ followed by a two digit hexadecimal code. For example, to turn on the enlarged characters the commands would be "ESCAPE p <shift> 4 0 <shift> e. This would appear on the screen as:

PRINTER CONTROL
$0E

Note that the p is lower case, the E and all other letters in the control codes are be uppercase and it is a zero as there aren't any o's in hex codes. A summary of the control codes to change characters are as follows:

Pica (10 cpi)	$1B$4E
Elite (12 cpi)	$1B$45
Proportional	$1B$50
Condensed (17 cpi)	$1B$51
Enlarged On	$0E
Enlarged Off	$0F
Bold On	$1B$21
Bold Off	$1B$22
Underline On	$1B$58
Underline Off	$1B$59

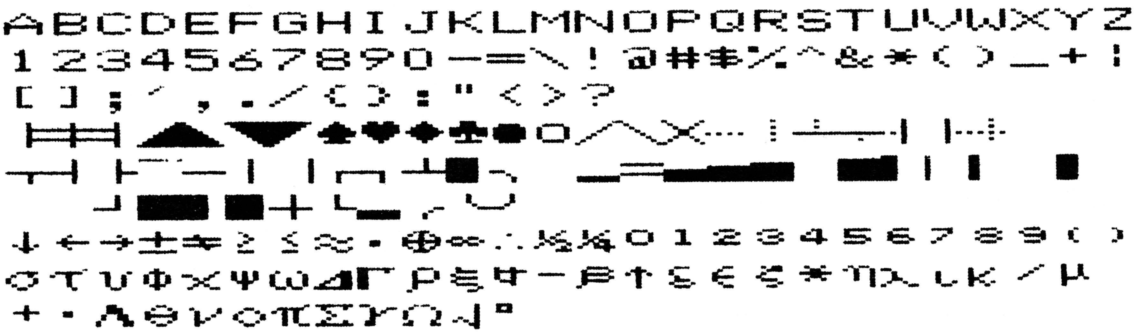
The last four control codes are better supported using the Open Apple Key, as they can be embedded in the middle of a line. The disadvantage of using the ESCAPE p is that the command must apply to the entire line. Also the enlarged, bold and underline commands must be turned off after being turned. With the Epson printers the enlarged mode turns off at the end of the line automatically. For example, the following sequence will yield the type of print shown in parenthesis:.lm+5

PRINTER CONTROL
$0E$1B$45
(Enlarged Elite-See Print Sample #6)
PRINTER CONTROL
$1B$51
(Enlarged Condensed-See Print Sample #8)
PRINTER CONTROL
$0F$1B$21
(Bold Condensed-See Print Sample #12)

1. ABCDEFGhijklmno
2. ABCDEFGhijklmno
3. ABCDEFGhijklmno
4. ABCDEFGhijklmno
5. ABCDEFGhijklmno
6. ABCDEFGhijklmno
7. ABCDEFGhijklmno
8. ABCDEFGhijklmno
9. ABCDEFGhijklmno
10. ABCDEFGhijklmno
11. ABCDEFGhijklmno
12. ABCDEFGhijklmno
13. ABCDEFGhijklmno
14. ABCDEFGhijklmno
15. ABCDEFGhijklmno
16. ABCDEFGhijklmno

The use of the four character sets in combination with the enlarge and/or bold command yields sixteen different combinations of type styles within the United States ASCII system. Also accessible via PRINTER CONTROL is a graphics symbols table and the Greek letters shown below in the enlarged mode. These character sets are accessed using $1B$23 and $1B$26 respectively, while $1B$24 takes one back to the ACSII characters. The Japanese Katakana is also available, but requires resetting three dip switches.

Another limitation is that the justify command does not work properly in the proportional mode. There are many other control codes available, some are in the C. Itoh and NEC 8023 manuals and most of the codes for the Epson are the same, if



one has access to any of the above. Lexicheck, Quark's spelling checker, works as efficiently as usual as it skips over the control codes. Lexicheck as it is the first and only spelling checker, I am aware of, for the ///; unless one performs a lobotomy and runs in emulation or CPM. While I would prefer to not have the limitations described above, I am quite happy with such a bug free program. Perhaps a later version of Word Juggler will have the control codes redefinable as part of the edit configuration. It is possible to do things like super- and subscript, but it involves many more commands than is practical to do on a continuing basis.

- /// -

Corrections to the Software Listing in Vol 1. Number 5

Kent Hockabout our Vice-President has found the following corrections to be made to the software listing in our Vol 1. Number 5.

Product Names:	Comments:
Context MBA	no plans for Apple ///
Multi-year planning (805) 324-6437 Financial Data Services and other Data-Systems Software	
RestAnal restaurant analysis management control concept	location unknown
Construction loan reporter computerized construction management systems	location unknown
Strategic simulations	no games for Apple /// in native mode

- /// -

File Cabinet Notes

Dear Don:

I recently ordered and received a copy of File Cabinet /// from your library. Upon using it I found some bugs which I thought I would bring to your attention if no one else has already.

I. The program refused to acknowledge a stored report format. Some detective work uncovered the reason. It was looking for a "RN" appendage to file while having placed a "RH" appendage there when storing it. A little investigative work with the help of "super sleuth" AppleWriter /// [F] function turned up the apparently offending line. Changing "RH" to"RN" in the line apparently cured the problem. It's too early to say whether it caused others.

II. Entry of more than thirteen characters as a file name caused line #876 to generate a never ending series of error messages.(Talk about being chastised for making a mistake!) Changing the GOTO from line #870 to 865 in line 876 took care of that one.

III. The accompanying manual is outdated, particularly in its reference in using the built-in clock. There are in fact no meaningful program lines beyond line 20130, and those which are there can be deleted.

It is, otherwise, a very nice little program which I have put to work cataloging and indexing my barely manageable repertoire of "floppies".
(Sure wish I had a Profile sometime, especially since I saw Quark's Catalyst program in action!)

Keep up the good work, and get those presses rolling, huh?

Ken Johnson
Amherst, MA.

Original Apple ///rs diskettes

OA3.BASIC.001

Business BASIC PROGRAMS ON THIS DISKETTE ARE:

HELLO The HELLO program is the program on the diskette which you should run first. After you put this program onto a Business BASIC boot diskette it will run automatically when you boot your ///.

DOC.READER This is a file reader that will allow you to read this file which was written with Apple Writer ///. You can use the up and down arrow keys to view the text. Use the ESCAPE key to exit and catalog this disk.

DISK.DOC Is a disk which is written with Apple Writer ///. It contains text file information.

INVENTORY This program utilizes several formulas to help you figure inventory costs.

MORTGAGE This and the next three programs were converted from Applesoft BASIC with the use of APPLECON conversion program available through the Original Apple ///rs. Mortgage will find out how much that new house really will cost you!!!!!

CAL.COUNTER This program helps you keep track of those awful calories. You may add more (!!) items to the list of foods if you want....

BIORHYTHM Will print out your biorhythm to a printer or screen. If you need to dup the information to a Silentype or an Epson change the line to whatever device name you wish.

GOLF Practice your golfing strategy using this program (See you at Pebble Beach). A text game converted from Applesoft. The program is originally from the San Francisco Apple Core Library.

SURVIVAL Can you survive in the wilderness??? Take this test and find out.

MAKE.MENU This is a feature program. It's a utility that allows you to make a menu to use in a program. The program that MAKE.MENU PRODUCES could be used as a HELLO program. The documentation for this program is called:

MAKE.MENU.DOC The documentation for MAKE.MENU can be read with Applewriter ///, Word Juggler, or any text reader.

PICKAFONT. This program donated by Jim Linhart, enables you to select a font or character styles or pitches. The program is designed to run in conjunction with an Epxon MX-100 printer. You can set the printer to whatever font or character style you want and it will remain in this setting until changed. The menu of this program is as follows:

1 -- Normal Width
2 -- Double-Width
3 -- Small font. (No underline available)
4 -- Compressed font

NOTE:

Most of the programs on OA3.BASIC.001 disk were converted from Applesoft BACIS using the conversion program APPLECON. The programs are from the San Francisco Apple Core's public domain library and modified by SAtan Guidero. The original authors names are included in REM statements in the listing. The APPLECON program is available through the Original Apple ///rs' for $10.00 or $8.50 to members.

More File Cabinet Notes

Dear Sirs:

As a member of Original Apple ///rs, we recently purchased a copy of the public domain software "File Cabinet ///" (version 2.0 dated 7/20/81). It is a nice program and will have application to several needs in our organization. After using the program and becoming familiar with it, we discovered a couple of items that did not work properly. They were the routines that save and delete files containing print format specifications. It may be that the others have had this problem or that we received an older copy of the program or corrections have been published and not seen by us. After considerable effort, we made changes in the program that appear to have corected the problem and not caused other problems. Please bear in mind that we are not "expert" programmers, and other corrections may need to be made.

Following are the corrections we made:

Change line 620

```
OLD-- F$=RN$(RF)+"RN":DELETE F$
NEW-- F$=RN$(RF)+"RF":DELETE F$
```

Change line 992

```
OLD-- F$="RN":GOSUB 350:FOR I= 1 TO
      NR:DELETE R$(I)+"RH":NEXT
NEW-- F$="RN":GOSUB 350:FOR I= 1 TO
      NR:DELETE R$(I)+"RF":NEXT
```

Add line 7155

```
OLD-- THIS LINE WAS NOT IN OUR PROGRAM
NEW-- RENAME FR$+""RF", TOO$+"RF":ON
      ERR GOTO 7180
```

Change line 9652

```
OLD-- PD$=DP$:DP$="":F$=RN$(NN)+
      "RH":GOSUB 350:...
NEW-- PD$=DP$:DP$="":F$=RN$(NN)+
      "RF":GOSUB 350:...
```

Change line 9880

```
OLD-- F$=RN$(NN)+"RN":ON ERR GOTO 9910
NEW-- F$=RN$(NN)+"RF":ON ERR GOTO 9910
```

AppleWriter /// Bug ??

We had a problem with another software package for the Apple /// that we wanted to describe to you to see if you had heard of others who had encountered the same problem. The software package is Apple Writer /// and it has been failing intermittently; that is, the keyboard will "lock-up" and even CONTROL-RESET will not function. The only way to reboot the software is to power off the machine. We use Business Basic, Access ///, Visicalc ///, Systems Utilities, Emulator, and other programs and have only had this problem with Apple Writer ///. Here is what we have done to try and solve the problem:

1. Cleaned both disk drives.
2. Upgraded SOS.KERNEL to version 1.3
3. Used the backup copy of Apple Writer // that has been unused since purchase.
4. Traded machines with our local dealer (he could find no problems with our machine). His machine would also "lock-up" intermittently.
5. Ran the confidence program immediately after failures. This procedure has produced mixed results. Sometimes a RAM error is indicated and other times no problems are detected. When a RAM error has occccurred, it has always been the same BNK and one of two ADRs.
6. We have a surge suppressor on the computer and all accessories.
7. A fan has been installed to help cool the machine although no accessories have been added inside the machine.

Any ideas or comments you have on the Apple Writer /// problem would be appreciated.

By the way, the Open Apple Gazette has been most informative to us. Keep up the good work!

Sincerely,

Joe Pase and Charles Bryant
Lufkin, TX

BUG REPORT

By: James A. Milligan, D.V.M.

Occassionally, in Apple Writer ///, when the [P] command is given, instead of a nice columnar listing of the printing parameters, the whole list flashes before my eyes on one line in the upper left corner of the screen, all in about 2 seconds. I have also had this happen one time with the listing options for [Q]. Needless to say, this makes checking the printing settings quite difficult, and once that quirk happens, it repeatedly does so until you go to the "Editor Menu" (pg.3 in manual) by hitting "Open Apple ?". I don't know what causes the bug to appear (and it does so quite infrequently), or why the key sequence of getting to the Editor Menu fixes it, but I did stumble on to that fact, and it has worked every time. I discussed this with a technical person at the Apple booth at Applefest, who said yes he had heard of it, and no, he didn't know the problem, but that it was a software problem.

An Introduction to VisiCalc Matrixing for Apple and IBM

by Harry Anbarlian

McGraw-Hill Book Company, New York, NY

252 pp., $22.95.

Reviewed by Stuart A. Forsyth.

VisiCalc -- the granddaddy of spreadsheet programs -- is the best-selling software program for microcomputers. It replaces paper, pencil, and calculator with a flexible, interactive electronic worksheet; and it does so in a subtly sophisticated manner which rises to the level of a microcomputer language. Part of its popularity is based on the user's ability to customize the program's power for their particular applications.

But how does one learn to use VisiCalc? Some persons are able to avoid that task by employing templates or models designed by others. But many want to build their own models, or at least modify those already designed for them. The traditional means of learning have been user manuals, tutorials, sample templates and personal training.

Harry Anbarlian has a different idea. He believes that one can learn to use VisiCalc by constructing a number of different matrices (or templates, or models, as they are called in other books) which embody the fundamental concepts of VisiCalc, have varying levels of difficulty, and are diverse and useful. It is an admirable goal and a tall order. He brings it off well.

A clear and concise introduction affords the reader an understanding of the book's purpose, organization, and use as a hands-on learning aid. It is intended to be used interactively with VisiCalc running on the reader's microcomputer. By page 11 the reader is ready to boot VisiCalc. The requisite foundation for using VisiCalc is built in the next 22 pages where the author introduces the matrix concept using a "boxes on a blackboard" analogy and explains the key VisiCalc commands in both Apple and IBM-PC dialects.

After only 40 pages the reader is ready to start using VisiCalc -- without wading through user manuals or tutorials. Now the fun begins.

The examples are broken into two sections, one for Apple users and another for IBM-PC users. Each section contains nine exercise matrices, arranged in groups of three according to complexity.

The simple matrices are: petty cash voucher, appointment calendar, credit card record, price earning ratio, inventory cost, and organization chart (yes, a titles-in-boxes-connected-by-lines organization chart done on VisiCalc without a single mathematical calculation!).

The moderately complex matrices are: treasury bill investment yield, payroll, construction trades equal employment opportunity, student's budget, education/selection impact ratio, and travel expense voucher.

The complex matrices are: bar graph, electric bill, zero base budget, departmental age analysis, cost/sales comparative bar graph, and stock portfolio.

Each is thoroughly explained, containing a clear statement of the objective, an explanation of how the task is accomplished by hand without VisiCalc and a microcomputer, and step-by-step instructions for creating the VisiCalc matrix. Each also includes an illustration of the completed blank matrix, an example of how to use it, steps for inserting data, and an illustration of the completed matrix with data and resultant calculations. To ensure that the matrices work and are free from defects, the author enlisted his wife - who had no knowledge of microcomputers or VisiCalc - to test each matrix.

While the examples in each section have been tailored to the keyboard of either the Apple II or the IBM-PC, the author has furnished enough information to enable even the novice Apple II user to explore, learn from, and use matrices from the IBM-PC section, and vice versa. Apple /// users likewise should have no problem. Using the examples on other microcomputers may require a little more familiarity with VisiCalc commands in order to adapt the matrices.

There is plenty of meat in this fine work. There is also an unusual and outstanding dessert, in the form of a final section on how to create polished matrices by inserting lines, spaces, titles, names, and dates and on consolidating and printing matrices. This seldom discussed topic takes the

VisiCalc user all the way to a professionally looking printed result.

The author's VisiCalc credentials include extensive personal use and user group experience. As a member of the Big Apple Users Group in New York City, he originated and was chairman of its VisiCalc Users Sub-Group. Along the way he learned to teach and write very well, and readers benefit immeasurably from his clear, well organized presentation.

For new users, An Introduction to VisiCalc Matrixing for Apple and IBM is an excellent alternative to the traditional ways of learning to use VisiCalc. It may even be better because it teaches painlessly, maintaining a high level of excitement and rewarding the reader with useful results, quickly obtained. While it is not designed to replace user manuals, it will make them more intelligible by serving as an experience building introduction to them.

Even experienced VisiCalc users may learn much from this work. The variety and imaginative selection of topics for the eighteen matrices means there is a good chance that even the most tenured user will find fresh ideas and techniques.

As just an introduction to VisiCalc, this book is excellent. But it achieves even more by also being a source of creativity and excitement in the use of VisiCalc. Following an introduction to VisiCalc in this book, users may well find themselves continuing to turn to it as a resource and reference (yes, there is an index). The author has not only achieved, but also transcended his stated purpose -- to the lasting benefit of his readers.

Stuart A. Forsyth is Director of the Office of the State Bar Court of the State Bar of California.

Reprinted with permission of SATN
(Software Arts Technical Notes)

- /// -

Exploring Business Basic - Part Five

By Taylor Pohlman
Reprinted from Softalk Magazine

Last month's column promised the answer to the question, How many bytes of memory are available in a 256K Apple III? As you know, the 256K Apple III has been announced and is beginning to be available, so the answer can now be revealed: 191,484 bytes! That's more than three times the workspace available in any other personal computer BASIC. (Aren't you glad you've got an Apple III? Don't you wish everybody did?) We were discussing some sorting techniques for our database that can make good use of that space. This time we'll explore a mixed bag of items, deferring our discussion of the "print using" capabilities of Business BASIC until next time.

Our Mixed Bag. The first bagged item this month is the mailbag. Several questions have come my way since this series started in September; the most interesting ones have to do with programming style and philosophy. The most intriguing question concerned why I always use lower case variable names in my programs, especially since the BASIC keywords (like "print") all seem to be in caps. While it would be easy to say that I lack the strength or will to operate the alpha lock key, the real reason has to do with the way BASIC itself works.

As you probably know, Business BASIC defers its syntax checking (looking for errors) until you actually run the program. BASIC does perform some tasks as each statement is entered, however; the process is generally referred to as "tokenizing". Simply stated, this means that BASIC scans each statement and converts each keyword, sometimes called a "reserved word", into a special internal one-byte code called a "token". This code not only saves space, but also simplifies error checking and program execution.

Almost all BASIC interpreters use the tokenizing technique. One of the consequences of this method is that program statements cannot be listed out without the BASIC "list" command converting these tokens back to their English-language equivalents. In converting the tokens, BASIC always prints out the upper-case version of the keywords. I type in all BASIC statements - both variables and keywords - in lower case so that when I list out a program, I can see what BASIC interpreted as keywords. If I misspell "print", BASIC will not recognize it as a keyword, and the fact that it remains lower case makes such an error easy to spot in a listing. In addition, Business BASIC requires spaces between keywords and variable names, to allow variables to contain keywords themselves.

Ever try to use a variable like "Orange" in Applesoft, only to discover that "or" is a reserved word (and therefore your variable must be renamed to something like "rnge")? Typing variable names in lower case will allow you to spot those times you forgot to space and ended up with "fori=1 TO 10" instead of "FOR i=1 TO 10". The first instance will produce an error, since BASIC will assume you are trying to assign the value of 1 to the variable "fori" and for some reason put the phrase "TO 10" onto the end of the statement. Some examples will clarify:

Typing: 10 prunt x*53

will result in: 10 pruntx*53

whereas: 10 print x*53

will result in: 10 PRINT x*53

Typing: 10 on xgoto 20,40,50

will result in: 10 ON xgoto20,40,50

whereas: 10 on x goto 20,40,50

will result in: 10 ON x GOTO 20,40,50

See how much easier it is to catch the error when it's displayed visually?
As with every rule, there are exceptions. Any variable that starts with the letters "FN" will be assumed to be a function name. Again, typing all lower case will help you spot the problem:

Typing: 10 xval=aval* fnumber

will result in: 10 xval=aval* FNumber

and you'll immediately know something's wrong (assuming that you really wanted to use "fnumber" as a variable name).

There's another little quirk in BASIC that this technique helped me spot. As you may know, we've used the on "eof#" statement quite a bit to take action if a program tries to read past the end of file. According to the manual, the part following "eof#n" can be any executable statement. So far, we've generally used "goto" or "gosub" statements to take action.

Consider the following:

Typing: 10 on eof#1 goto 20

will result in: 10 ON EOF#1 GOTO 20

as you'd expect. But:

Typing: 10 on eof#1 xval=20

will result in: 10 ON EOF#xval=20

For some reason BASIC treats the whole thing as one variable. The solution involves dredging up a bit of BASIC folklore. Remember in your first class in BASIC when they told you that all assignment statements started with the keyword "let"? Most BASIC dialects have long since made the "let" keyword optional, and most people have quit using it altogether. An example of the use of "let" is:

> 10 LET x=45 which is usually written
> simply: 10 x=45

If there's any ambiguity to the way a statement can be interpreted, "let" can be used to clear it up. With our new version of the "eof" statement:

> Typing: 10 on eof#1 let xval=20
>
> will result in: 10 ON EOF#1 LET
> xval=20

and everything works fine. The fact that BASIC failed to upshift the reserved word "eof" in the example above is very important to an understanding of the problem. The technique of entering everything in lower case has saved me countless hours of debugging my errors. I recommend it.
Bag Item Number Two. Last month's list of new goodies in Business BASIC 1.1 completely overlooked one item which, while it may seem minor, has important consequences. The change is an extension to the standard "get" statement. Normally, as is the case in Applesoft and some other BASICs, "get" allows reading the keyboard one character at a time, including all special control characters and delimiters. This means you can bypass control-C and return, read commas, and so on.

Business BASIC 1.1 extends "get" to allow "get#n". This means you can read any SOS file one character at a time, without respect to what kind of file it is. This can be very handy for reading all characters from the communications port (via the .RS232 driver) or for reading other character streams from special devices. One of its most interesting traits, however, is the fact that it can be used on disk files as well. Remember that one file is just like another in the SOS environment, so if we open a text file on disk, "get#" will allow us to read one character at a time from it.

This means that there's now an easy way to read text files that contain more than 255 characters without a return character. Normally a string overflow error results if you attempt to read such text files with the BASIC "input" statement. Even more interesting is the fact that we can also open and read from the BASIC data file. Remember that I described the data file as having special tags, called "type bytes", that enable BASIC to determine what data type is stored next in the file. Remember also that numeric data is stored in a data file in its binary form. Get# allows reading this binary information, one byte at a time. One example is worth a thousand explanations:

```
5     INPUT"File to dump: ";a$
10      IF a$="" THEN 100
15      OPEN#1,a$
20      ON EOF#1 GOTO 100
25      cr$=CHR$(13)
30      GET#1;a$
40      IF a$=cr$ THEN PRINT
50      PRINT a$;
70      GOTO 30
100       CLOSE
110       END
```

This simple example will dump any text file to the screen, no matter how long the intervals between carriage returns. A good example of a text file with arbitrarily long strings is the file I'm creating now, using Applewriter III. Return characters are inserted only at the end of paragraphs which, as you'll notice, tend to run on indefinitely.

Note that this program looks for return characters by loading the variable "cr$" with a return (decimal 13) and then testing for it before printing. If you wanted to reconstruct strings from the file, you could do so by using a string variable to accumulate characters, stopping when a return was encountered. You'd need to test to be sure you hadn't overflowed the 255 character limit.

This program has one serious deficiency, however. Printing arbitrary characters from a file (especially a data file) can have weird consequences when the output device is the console, as it is in the example program. The console uses lots of different control sequences to perform functions, including setting windows and changing from black and white to color text modes. Also, a byte can contain 256 different characters, and the ASCII character set defines only 128. Clearly, we need a safe and consistent way to display any byte readable from a file. So, like most programs that start out short and simple, this last one's about to get complex:

```
5     INPUT"File to dump: ";a$
10      IF a$="" THEN 95
15      OPEN#1,a$
20      INPUT"File for output: ";a$
25      OPEN#2,a$
30      ON EOF#1 LET eof.occurred=1:
    GOTO 80
35      bytes=0:eof.occurred=0
40      line$=""
45      PRINT#2;HEX$(bytes);"-";
    HEX$(bytes+31);"";
50      FOR i=1 TO 32
55        GET#1;a$
57        val=ASC(a$):IF val>127 THEN
      val=val-128
60        IF  val<32 THEN  line$=
      line$+".":ELSE:line$=line$+
      ""+CHR$(val)
65        outhex$=HEX$(ASC(a$))
70        PRINT#2;MID$(outhex$,3,2);
75        NEXT i
80      PRINT#2:PRINT#2;"     ";line$
85      bytes=bytes+32
90      IF eof.occurred=0 THEN 40
95      CLOSE
120       END
```

As you scan through the program, note that in addition to opening the file to be dumped, we open a second file to which the output is written. This gives us more flexibility, and still allows us to use .console to see the output on the screen. Line 30 sets up our end-of-file condition, using the "let" statement to get around the problem we described earlier, and demonstrates one other handy thing. We can embed periods in variable names to improve readability. It's obvious that "eof.occurred" is easier to interpret than

```
0000-001F  2E636A0D5420482045202054204820492042204420204220412053204920430D

            . c j . T   H   E       T   H   I   R   D       B   A   S   I   C

0020-003F  0D6279205461796C6F7220506F686C6D616E0D0D0D2E6C6A0D4578706C6F7269

            . b y   T a y l o r   P o h l m a n . . . . l j . E x p l o r i

0040-005F  6E6720427573696E65737320426173696320202D2050617274206669766650D0D4C

            n g   B u s i n e s s   B a s i c   -   P a r t   f i v e . . L

0060-007F  6173742074696D6520492064726F7070656420736576657261C2062726F6164

            a s t   t i m e   I   d r o p p e d   s e v e r a l   b r o a d

0080-009F  2068696E74732061626F7574206E657720736F667477617265206616E64206861

              h i n t s   a b o u t   n e w   s o f t w a r e   a n d   h a

00A0-00Bf  72647761726520686170706656E696E6773206F6E2074686520417070C65202F

            r d w a r e   h a p p e n i n g s   o n   t h e   A p p l e   /

00C0-00DF  2F2F2E2020486F706566756C6C79206279206E6F7720796F7520686176652078

            / / .     H o p e f u l l y   b y   n o w   y o u   h a v e   h

00E0-00FF  61642061206368616E636520746F20676F20646F776E20746F20796F75722064

            a d   a   c h a n c e   t o   g o   d o w n   t o   y o u r   d
```

Figure 1.

```
------------------------------------------------------------------------
```

```
0000-001F  00000000000000000000000000000000000000000000000000000000000120B

            . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

0020-003F  2C21094345454245343535343214A16617221800034EA713C9C4000000121BC321

            , ! . C E E B E 4 5 4 2 . ! f . " . . . N ' . I D . . . . . C !

0040-005F  0943444341443130333114A14CCE1D18000373AD91E0A40000001223F6210944

            . C D C A D 1 0 3 1 . ! K N . . . . s - . ' $ . . . . # v : . D

0060-007F  4441444538323339149E25AA7B180002FBF1C2308C000000120BDE2109414442

            D A D E 8 2 3 9 . . % *   . . .   q B O . . . . . .   ! . A D B

0080-009F  41433434353014A135A30818000169EF7321C6000000120EE621094141424544

            A C 4 4 5 0 . ! 5 # . . . . i o s ! F . . . . . f ! . A A B E D

00A0-00BF  3930353714A007D1D0180002BAAF52D728000000

            9 0 5 7 .   . Q P . . . : / R W ( . . .
```

Figure 2.

"eofoccurred", and this is especially true for more complex variable names (remember that Business BASIC permits 64-character names).

Lines 35 and 40 initialize variables. We will be using the "line$" string to accumulate the characters read from the file for later printing. After each line of "print" we will reinitialize the string. Since we'll be printing thirty-two characters at a time from the file, line 45 uses the "hex$" function to set up the labels for each line.

A note about hex is appropriate here. Hex stands for hexadecimal, or base-16, arithmetic. Since any hex digit can be represented by four binary bits and a byte can be exactly represented by two hex digits, it is convenient to use hexadecimal numbering in many aspects of computing. It is preferred over decimal and octal notation and is, of course, much more compact than binary. What usually throws people is that to represent all values between 0 and 15 with a single digit, hex uses the numerals 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, respectively. F thus is equivalent to decimal 15, and 1F to decimal 31 (the 1 is in the sixteens place).

We won't try to offer an in-depth explanation of hexadecimal notation here. If you aren't familiar with it, any beginning text on computers usually covers the subject thoroughly, and readers of Roger Wagner's column in this magazine have been inundated with help on hex. Suffice it to say that the "hex$" function will convert any reasonable numeric quantity into a four-byte string of hex digits.

Getting back to our program, the loop from line 50 to line 75 is the main one where we dump thirty-two bytes at a time in hex format, while providing character representations for those within the displayable range (hex 20 to 7F, decimal 32 to 127). The back of your BASIC manual contains an ASCII code chart that will help you follow along with the decoding. Line 57 in the program sets the variable "val" to the ASCII value of the byte just read, and then an "if" statement checks to see if the value is in the 128 to 255 range. If so, 128 is subtracted from the original value to bring it within the normal ASCII range.

Line 60 checks to see if the resulting character is a control character, and if so, represents it as a period in "line$" to signify that it is unprintable. Otherwise, the character representation is stored. The characters are right justified in each two-byte cell, because they'll be printed below the hex values. Next, the hex value of the original character is assigned to "outhex$" in line 65, and printed to the output file in line 70. Since we want only the rightmost two hex digits, the "mid$" function is used. After the loop prints out the thirty-two values, lines 80-90 print the ASCII equivalents stored in "line$", bump the byte count, check for "eof" condition, and repeat the sequence.

Figure 1 shows how the output from this little jewel looks when run against the file for the first draft of this article.

Messy, huh? Let's look more closely at the output to see if it makes sense. The first line tells us we are looking at bytes 00 through 1F (0 to 31 decimal), and the top line is the hex representation of the characters, two digits per character. The first character in the file is 2E in hex, which happens to be a period. Notice that 2E is the character printed below on the next line. The next two characters in the file are 63 and 6A which correspond to the ASCII characters c and j. This is understandable, since Applewriter III uses the print format command .cj for center-justify, which is what I wanted done with the title. The next character is 0D which translates to decimal 13, or a return character. Note that a period is substituted for this character on the print line, since return is in the control character range. And so on, and so on. Practice on a few text files of your own and get a feel for reading the notation.

It really gets interesting when we begin reading files whose exact format is normally pretty obscure. Data files are an excellent example since, although the "read#" statement can get data out, things like the type bytes and string-length bytes are normally inaccessible. To see how our dump program would work on a data file, we need a way to generate an interesting file at which to look. The following simple program will do the trick. When we get serious later on about sorting techniques, we'll need such a program, so I'll introduce it now:

```
5   OPEN#1,"junkfile",30
6   INPUT"Number of records to
 create: ";n
10    FOR i=i TO n
12      i%=RND(1)*10000
```

```
     13     WRITE#1,i;i%:PRINT i%,
     15     a$=""
     20     FOR j=1 TO 5
     30       a$=a$+CHR$
           (65+INT(6*RND(1)))
     35       NEXT j
     41     FOR K=1 TO 4
     42       a$=a$+CHR$
           (48+INT(10*RND(1)))
     43       NEXT k
     45     WRITE#1;a$:PRINT a$,
     48     val=RND(1)*1E10:WRITE#1;val:
        PRINT val,
     49     i&=CONV&(RND(1)*1E15)
     50     WRITE#1;i&:PRINT i&
     55     NEXT i
     60    CLOSE
     70    END
```

This program will create a random access data file of arbitrary length containing an integer, a string, a real, and a long integer in each record. What's noteworthy here are the two small loops that build the string value. They're set up in such a way as to insure that the first five characters are upper-case alpha and the next four are decimal digits. Type the program in now and run it to create a small file, say five records. Although each run will differ, the output should look something like this:

```
2092 CEEBE4542 7.72055E+09 930904428626944
7107 CDCAD1031 6.87212E+09 971614244086784
9206 DDADE8239 6.94853E+08 839965717072896
3038 ADBAC4450 6.09472E+09 397952126404096
3814 AABED9057 2.27867E+09 768212125296640
```

Now for the fun. When you run your dump program against the file this program creates, the output should look something like figure 2.

Well, nobody said computer science was for the faint of heart! By the way, the term generally used to refer to this type of listing of file contents is "formatted dump". Formatted, because we have organized the information in the printout, and dump, because it is a nonselective output of the exact contents of the file.

Now the fun begins. The first thing to notice is that almost the entire first line is composed of zeros. Remember that although our dump program starts at the beginning of the file, the program we used to create this file began at record 1. Since the record size was thirty bytes, we would expect to find an empty record of thirty bytes at the beginning, and that's exactly what the dump shows. This means that the hex 12 in byte 31 of the file must be at the beginning of record 1. Now something that was mentioned earlier about type bytes in data files becomes important. Remember that the general format of a data file is:

```
   Type byte    Data bytes (2,4 or 8)
```

for numeric values (integer, real and long integer), and:

```
Type byte length byte Data bytes (0 to 255)
```

for strings. This information should enable us to decode the information in this dump.

Since the first value in the record was an integer, the hex code 12 must be the type byte for integer data. Following our format, this means the next two bytes (hex codes 08 and 2C) must be the binary integer value. Evaluating the hex value 082C yields decimal value 2092, exactly the value our printout led us to expect.

The next value in the file is a string, which contained CEEBE4542. Referring again to our format for strings in data files, we'd expect the next file byte to be the type byte. That's the hex code 21. Next is the length byte, which, since the string is nine characters long, should be equal to 9. That's hex code 09, one of those lucky hex numbers that is the same as its decimal equivalent. After that, our format line shows that indeed, the string value is CEEBE4542.

The next value in the record was real. Since the next byte after the string should be the type byte for reals, we can conclude that the hex 14 found in position 2C (44 in decimal) is the floating-point type byte. Floating-point numbers are stored in a thirty-two-bit internal format in Business BASIC, so we would expect the next four bytes to contain the binary value. Proving that this value (hex A1661722) is equal to 7.72055E+09 is a considerably more complex task and will be left to the numerically inclined reader. That phrase "left to the numerically inclined reader" is this author's equivalent to the famous line found in all math texts - "it can easily be shown that..." - and is just as big a cop-out.

The last value in the record is a long integer, and the type byte in position 31 (decimal 49) has the value of hex 18. Long integers are stored as eight byte quantities, therefore the next sixteen hex digits should represent the number. Since that hex value is 00034EA713C9C400, it follows that converting this value should yield the decimal value originally printed out: 930904428626944.

As a little added bonus in this article, let me offer a program that demonstrates the truth of the preceding statement. This program will convert any reasonable hex value into decimal and print it out rather quickly, using the long-integer data type and Business BASIC's conversion functions. Forthwith, it is:

```
5     sixteen&=16
10       INPUT"hex value: ";a$
15       IF a$="" THEN 100
20       cum&=0
25       mult&=1
30       FOR i=LEN(a$) TO STEP-1
35           val&=CONV&
          (TEN(MID$(a$;1,)))
40           digit&=mult&*val&
45           cum&=cum&+digit&
50           mult&=mult&*sixteen&
55           NEXT i
60       PRINT cum&
65       GOTO 10
100       END
```

The program simply brute forces the problem, one digit at a time, but since the long integer arithmetic is very fast, program performance is quite reasonable. This program knows nothing about sign bits, though, so it will fail to convert negative integers expressed as hex constants. A fix for this limitation would be to check for the high-order bit and negate the final result, but the program would then lose its general nature. Anyway,. it's free.

Well, that got us completely off track. Going back for a second to the formatted dump, we are now at position 3A hex (58 decimal), which is really position 28 decimal in this record. The remaining two bytes of the record (remember that we declared the record to be thirty bytes long) should be empty; sure enough, they show up here as zeros. This gets us to position 3C, the beginning of the next record, and there we find the integer type byte 12, signaling that we can start the whole process again. I leave that to you if you want to try your hand at decoding. Some of what we have learned can be summarized in the following table:

Data type			
	TYP() function	Internal	file code
name	value	hex	decimal
Integer	2	12	18
Real	1	14	20
Long Integer	3	18	24
String	4	21	33

Don't forget that the "get#" statement can be used in lots of other interesting ways and that its primary function is to process console input effectively without those characters being first processed by BASIC. I just thought the examples above would give us a chance to explore several interesting topics at once.

Final Thoughts (Bottom of the Bag). I'd fully intended to explore one more topic that had previously generated questions, but this tome grows overlong. The topic I had in mind was the use of the "request" invokable module. Those of you who are writing programs that do lots of reading and writing of numeric arrays to disk should tune in next time when we show how to get at least twenty times the performance improvement over using "for-next" loops to accomplish the same task. That, combined with the huge memory space available for arrays, provides some significant capability to the person interested in data analysis and sophisticated file indexing.

I also promise to get to my thesis on "print using", especially since Business BASIC allows some tricks not available in most other BASICs. One of these days we'll get to graphics as well, and discuss how to use "bgraf" and "download" to create some really interesting stuff.

Until then, just one last note. I looked back over this article and decided that the word "hex" was mentioned so many times that we have left the era of "voodoo economics" and entered a new era of "voodoo BASIC". Oh well. Maybe if I wore garlic while typing . . .

- /// -

GAMEPORT III

User Comments
by Richard Lawler

I have found that the one I have installed works well with all the games I have tried (these include some games from Broderbund that they say will run on the Apple][e). The Gameport III seems to cause some strange effects on the Apple /// in native mode. It might be a good idea to to tun off the power to the machine whenever rebooting to native mode from][mode. This will deactivate the Gameport. Also be careful not to push the button on the joystick (which activates the Gameport) in native mode.

Another problem is the inability of the Gameport III to work with the Apple /// joysticks. I talked to the Micro Sci people at the West Coast Computer Faire, and they did not know why the engineers did not allow for Apple /// joysticks. Aside from these minor problems, the Gameport III does just what it is supposed to do.

(Editors Note) Micro-Sci did not make provision for Apple /// joysticks for two reasons:

1. Very few Apple /// owners have Apple /// joysticks.
2. The internal construction of an Apple /// joystick is different from an Apple][joystick. This prohibits it from being used with the GAMEPORT III under any circumstances.

- /// -

SOS 1.3 released by Apple

Apple /// Software Revision Utility, Version A01 has been sent to all Apple /// dealers. This utility diskette is used to replace the SOS.KERNEL on all your boot diskettes with SOS 1.3, dated 1-Nov-82. This new SOS.KERNEL supercedes all previous releases.

The changes include:

1. Support for Backup ///. Each time a file is modified, the operating system turns on a status flag located in the file's directory on the disk. Backup /// then uses this flag when performing the Modified Files backup. This enhancement allows users of the backup program to save time and reduce the number of backup diskettes required.

2. Eliminate the possibilities of data loss when using the ProFile mass storage system during high interrupt operation.

3. Correct calculation of dates during leap year.

Also included on the diskette in a subdirectory named NEW.DRIVERS are several updated drivers. These drivers are all Version 1.3.

PROFILE.DRIVER	Profile Driver
FMTDX.DRIVER	Format Driver
CONSOLE.DRIVER	Console Driver
SERPRINT.DRIVER	Serial Printer Driver
PARPRINT.DRIVER	Parallel Printer Driver
RS232.DRIVER	RS232 Serial Driver
TCLOCK.DRIVER	Thunder Clock Driver
GRAFIX.DRIVER	Graphics Driver

NOTE If you are using Quark's Catalyst, DO NOT change the .CONSOLE driver or the .PARALLEL driver on your Catalyst boot diskettes. You have to have Quark's drivers and these MUST NOT be changed.

The new format driver for Disk /// (FMDX.DRIVER) now has code which will check the speed of the disk drive. The driver will generate new device-specific errors if the drive is too slow (#33) or if too fast (#34). If you ever receive one of these errors during formatting the particular drive should be taken to your dealer for speed adjustment. The requirement for a drive to be within speed calibration before a disk is formatted is anticipated to prevent creation of bad diskettes and resultant I/O errors when the disks are read on other drives.

You can use the System Utilities Program to check the version numbers of your drivers on your boot diskettes. If you do not have these latest versions use your System Utilities Program to add the new drivers you need to your boot diskettes.

- /// -

Cable Pin Assignments

Universal Parallel Card to
Microbuffer to
IDS Microprism Model 480

by Barney Simonsen

As a follow up to my evaluation of the IDS Microprism Model 480 Printer, I would like to describe the recent change in it's configuration. I elected to install the Apple Universal Parallel Interface Card to free the RS-232 plug for use with a modem, and at the same time add the Microbuffer In-Line Parallel Printer Buffer from Practical Peripherals, Inc.

The buffer I elected to purchase was configured with 64K of memory, which I have found to be ample for the programs and files I am handling. The unit is able to be expanded later to 256K.

Since none of the three components (UPIC, buffer, or printer) had any cables to connect this configuration, I was forced to use the manuals provided. The IDS manual had the most detail, including a mapping of the pin positions in a standard Centronics compatible cable. The UPIC Operating Manual also provided ample mapping of the expected pin functions, although the terms utilized were different from those used by IDS. That left the buffer as the unknown, and after reviewing the manual, there were few clues. At the time I was having the cables prepared, I also wanted to have a cable to connect the UPIC directly to the printer in case of failure of the buffer. This turned out to be the easiest of the cables to create because of the mappings indicated above. I am providing a listing of the pin positions which resulted from this exercise in case anyone is interested in duplicating the installation.

I have been very pleased with this configuration. The buffer allows complete printing times of less than one minute for lengthy runs, allows duplicate copies to be printed without use of the computer, and frees the RS 232 port. Once the cables were properly prepared, the system has operated for the last month with no problems. Control characters for the IDS printer are passed with no difficulty.

Cable Pin Assignments

UPIC Pin #	Function	Buffer Pin #
1	Signal Ground	19-30, 14,16,33
2	ACKNOWLEDGE INPUT	10
3	Data Input Bit 0	-
4	Data Input Bit 1	-
5	Data Input Bit 2	-
6	Printer in Check Input	-
7	Printer Ribbon Out Input	32
8	STROBE OUTPUT	1
9	Printer Out of Paper Input	12
10	Data Output Bit 0 (LSB)	2
11	Data Output Bit 1	3
12	Data Output Bit 2	4
13	Data Output Bit 3	5
14	Data Output Bit 4	6
15	Data Output Bit 5	7
16	Data Output Bit 6	8
17	Data Output Bit 7	9
18	Printer Online Input	13
19	Printer Power On Input	-
20	Signal Ground	17

Buffer 36 pin Centronics Female Pin #	Function (DB-255)	IDS Printer Pin #
1	Strobe Output	3
2	Data Output Bit 0	14
3	Data Output Bit 1	13
4	Data Output Bit 2	12
5	Data Output Bit 3	11
6	Data Output Bit 4	10
7	Data Output Bit 5	9
8	Data Output Bit 6	15
9	Data Output Bit 7	16
10	ACKNOWLEDGE INPUT	22
11	Busy	19
12	Printer Out of Paper Input	24
13	Printer Online Input	4
14,16, 19-30, 33	Signal Ground	7
17	Chassis Ground	1
32	Printer Ribbon Out Input	18

UPIC Pin#	Function	IDS Printer Pin#
1	Signal Ground	1
2	ACKNOWLEDGE INPUT	22
3	Data Input Bit 0	-
4	Data Input Bit 1	-
5	Data Input Bit 2	-
6	Printer In Check Input	-
7	Printer Ribbon Out Input	18
8	STROBE OUTPUT	3
9	Printer Out of Paper Input	24

10	Data Output Bit 0	14
11	Data Output Bit 1	13
12	Data Output Bit 2	12
13	Data Output Bit 3	11
14	Data Output Bit 4	10
15	Data Output Bit 5	9
16	Data Output Bit 6	15
17	Data Output Bit 7	16
18	Printer Online Input	4
19	Printer Power On Input	-
20	Signal Ground	7

I hope the above can save someone the effort to search out the names and positions of these functions.

- /// -

Problems with Updating SYSTEM.WRK.TEXT
or
Wonder why I can't save it?

When you first used the Apple /// PASCAL Editor and decided to use the default file name SYSTEM.WRK.TEXT for saving your work, it very likely did what you wanted. After you made the usual mistakes, made corrections to the file, and decided to save the changes back into SYSTEM.WRK.TEXT, you probably got the message:

```
ERROR:Opening the file.
Please press <space> to continue.
```

When this happened to me, I immediately concluded that I had a bad disk or the PASCAL system had a bug. After talking to my good friend Jon Stevens and being accused of being a dummy, I realized that there was not enough room on the PASCAL1: disk to save the updated file, presumably because the old file is not deleted until the new one is successfully saved. The solution to the problem is to open the manual "Pascal Introduction, Filer, and Editor" and, following the instructions on Page 9, use the PASCAL1: disk to create NEWPASCAL1: and NEWPASCAL2:. As explained there, the SOS.KERNEL, SOS.DRIVER, and SOS.INTERP files needed to boot the Apple /// essentially fill up the available space on the disk, so a two-stage boot has to be used. Maybe someone can explain why the disks don't come set up this way.

Reprinted from HAAUG Apple Barrel.

- /// -

Bench Marking the Apple ///

by Arthur Anderson III

"Eratoshenes Revisited", Byte Magazine, Jan. 1983, page 283, benchmarked a large number of computer systems using various languages. Spurred by this recent article, I made my own measurments on the Apple ///. I am sure the Original Apple ///rs would be interested in my recent measurements on the Apple /// combined with its software.

PASCAL///

TEST			ACTUAL
CONDITIONS	LOOPS		TIME
	ONE	TEN	TEN LOOPS
STANDARD DEFAULTS	37.9	379	6:19
NO RAINCHECK	33.5	335	5:35
NO RAINCHECK/DISPLAY OFF.	26.5	265	4:25

COBOL///

			ONE LOOP
STANDARD DEFULTS	338	3380	5:38
'COMP.' VARIABLES	317	3170	5:17
'COMP.' /NO DISPLAY	266	2660	4:26

BASIC///

SIMILAR TO BYTE ARTICLE	363	3630	5:17
ABOVE & NO DISPLAY	287	2870	4:47
SLIGHTLY REWRITTEN FOR SPEED	313	3130	5:13
ABOVE & NO DISPLAY	247	2470	4:47
ENHANCED BUT UNTESTED	186	1860	SEE BYTE MAG. 1/83 P. 294

These measurements bring to mind several Apple /// observations:

1. Pascal is 10 times faster than BASIC or COBOL. About 30 seconds compared to 5 minutes!

2. One can gain a 20% to 30% improvement in execution by turning the display off (Control 5 on the numeric keypad).

3. One can gain a 7% to 17% improvement in execution by compiler directives. (i.e. turn-off rangechecking, enable compact variables, etc.)

4. One can gain 15% and more, by cleverly rewriting a program.

These observations lead me to these (rather sweeping) generalizations for Apple /// code:

1. Use Pascal if possible. (Of course assembler is better still...)

2. Turn the display off, when the Apple /// is very busy. (Hence, design programs to turn the display "on" when they are ready for human interface.)

3. For about the same speed improvement, it is easier to use a compiler directive than to cleverly rewrite the code. However, this is not a license to write code poorly. Proper usage of algorithms and data structure are mandatory. (But who really cares which BASIC variables are used most frequently and who really wants to use variables instead of constants?)

In relation to measurements appearing in the Byte article, the authors appear to want the fastest possible timings. The barebones minimum is illustrated by the "Range check off" statement in the ADA program, the use of native code generators (rather than p-code), the "COMP" variables in the COBOL program, and the myriad of (of almost inexplicable) timing variations. Thus the Apple /// with Pascal /// should be compared with range checking off and, probably, with the display off. COBOL should use the "COMP" directive and, probably, the display off. Clearly Mr. R.W. Shore's contribution for Apple /// BASIC timing is better than my hasty efforts and nas been included in the table above.

The Apple /// in Pascal (265 s) performs similar to a Mill-enhanced Apple // (273 s), but not quite as fast as an Apple // programmed in Forth (190 to 208 s). The Apple /// in Pascal lags the 68000 family in Pascal (4.28 to 11.2 s)
No performance measurements of Pascal or COBOL were available for the IBM PC.
An IBM PC programmed in BASIC (1950 to 2400 s) runs similar to the Apple /// Business Basic (1860). The IBM PC programmed in "C" (22.1 s) or in Forth (70 s) are quick, but lag behind the 68000 family. The lack of Apple /// timings in these languages (and the lack of IBM PC timings in Pascal or COBOL) presently make these machines hard to compare, except in BASIC. The Apple /// COBOL (2660s) is faster than the Z80, CP/M, Microsoft V2.2 COBOL (5115 s). Most other COBOL comparisons are based on the minicomputer to maintain classes: HP3000 (58s), IBM3033 (0.0824),IBM Series 1 4995 (38.7s), Prime 300 (50.4), etc.

- /// -

Catalyst Notes

Dear Don:

In the "Public Domain Software" section of Open Apple Gazette, you've mentioned a "DOS to SOS Text File Convertor," always with the annotation that the Apple Writer Utility diskette does the same thing. Gee, I hope not. Some of the additional notes that I've added to my Quark Catalyst file include the following:

DO NOT INSTALL APPLE WRITER UTILITIES ONTO PROFILE

The Apple Writer Utilities diskette transfers DOS/SOS files back and forth, and it converts Mail List Manager files into source files for Apple Writer form letter names and addresses. Given a felt need for any of those functions, one is understandably tempted to add the program to Catalyst.
DO NOT, UNDER ANY CIRCUMSTANCES, SUCCOMB TO THAT TEMPTATION. Why? Let me tell you a story.

My wife has a //e, and my /// has the family letter-quality printer. For her convenience, I installed Apple Writer Utilities as a Catalyst main menu selection. No problem at all. Then I tested it, to see if one of my Apple Writer /// files could be successfully converted to Apple Writer][e format. Wrong test. It clobbered my Profile.

The Apple Writer Utilities program requires that the Apple][disk, no matter the direction of transfer, be in ".D1," the internal drive. That's what the documentation says. Actually, the program requires its presence in Pacal Unit #4.

The Apple /// disk, on the other hand, may be in any other drive, and the program references it by its SOS device name (.D2,.D3, etc). This lack of consistency was fatal. The program correctly found the Apple Writer /// file in .D2 and then blythely went about writing the converted DOS file to Pascal Unit #4 - the Profile. Good-bye. One otherwise perfectly good five megabyte disk written to in Apple][format. As I said, I'm a terrific systems tester.

The disaster could easily have been averted by good programming practice. If the program had been consistent in its disk drive references, the worst that would have happened would have been a "file not found" error message. Had SOS device names been used consistently, there would have been no problem. Had Pascal Unit Numbers been used consistently, the program would have looked for the Apple /// file in the internal drive and not found it. In any event, the program should have had the courtesy to see whether the output file was really a DOS disk. I will continue to wish several uncomfortable and improbable occurences to befall the author of Apple Writer Utilities.

The safest thing that you can do with Apple Writer Utilities-- at least if you have a version that doesn't know about Profile--is to leave it alone as a separately booted disk. Even if you only use it to convert Mail List Manager files, just having the other, deadly option available is a disaster waiting to happen.

If the program y'all are offering doesn't engage in deadly embrace with Profile, it is (1) an improvement and (2) well worth the money.

Yours truly,

Allan M. Bloom

Editors Note: We returned his check.

MENU GENERATION PROGRAM

by R. D. Biggs

Since the summer of 1980, when my wife and I purchased our first personal computer, an Apple][, I have been intrigued with the variety of problems that can be solved. A few months ago, we moved up to an Apple ///. I have written and rewritten many programs, and I have come to appreciate the advantages to the user of a well-conceived, menu-driven program.

It occurred to me recently that a utility program designed to create or write a skeleton program containing a menu would be handy and time-saving. The concept was to have the utility program write a text file with the format of the desired BASIC program. After the text file is saved on a disk, the text file could be converted to a BASIC program using the EXEC command. Hence, the utility program MAKE.MENU was born.

When the program is run, it prompts you for two title or header lines for the menu to be created, the creation date, from 1 to 10 menu labels and the path name for writing the text file to a disk. After you complete the data input, you are presented with four choices:

1. Print to console
2. Print to disk
3. Edit/Review entries
4. Exit program

Choice 1 prints the text version of the desired menu program to the console, which allows you to review the text file before writing to the disk.

Choice 2 writes the text file to the disk using the specified path name. This choice must be exercised before exiting the program if you wish to save the menu program .

Choice 3 allows you to review and edit any or all of the original input data before saving the text file.

Choice 4 closes the file and console driver, deletes the main body of the generator program, executes the EXEC command to convert the text file to a BASIC program in memory and runs the menu program. The program, as created, contains dummy lines to branch to when menu options are selected. As you develop the body of your program, you replace the dummy lines with appropriate lines of code.

Before saving the BASIC program, you will want to delete the extraneous lines 1 through 9 and lines 50000 and 50010. MAKE.MENU is documented and should be easy to read and modify. I am sure that there are many improvements to be made, and I am looking forward to feedback from interested Original Apple ///rs.

- /// -

Tidbits

by Paul Wilson

Tidbits is a column which is designed to allow you to send in things of general and specific interest which do not warrant a whole article. Send in your ideas related to Apple ///s, and the use thereof, and this will be a very helpful and interesting column indeed. Please limit your submissions to one or two paragraphs, but send in as many as you like.

Tidbit 1

The first tidbit relates to the use of control characters in Apple Writer. Control characters can be very useful in modifying the printed output for certain printers (see your manual). The Apple Writer manual says that the only way to enter control characters into your text is by the use of control-V. This is not so! Pressing the open-apple key as well as the control key when typing characters has the same effect as being in control character mode (control-V). Escape characters can also be generated easily by pressing the open-apple key and the <escape> key simultaneously. Try it.

For those of you who don't like paying lots of money for backups of those important and uncopyable programs like Visicalc and Apple Writer, you can use Locksmith to make backups. That's right, now you can pay lots of money for Locksmith 4.1 (about $200) and make backups for the cost of a diskette. This Apple II program runs fine in emulation mode and quite easily handles SOS diskettes.

In no way am I suggesting that you may make copies for your friends and relatives. Those of you who write programs to sell will be quite aware of the frustration and financial loss that this type of thing causes. Not only is it against the law but it is unfair. 'Nuff said.

As to what is copyable, I sat up late one night and copied Apple Writer (no problem), Business Graphics (medium difficult), and Advanced Version Visicalc (not too hard, easier than Business Graphics). It took me about three hours. It is also not too consistent. As an experiment I have copied Visicalc three times, each time I ran into different problems. You must use your ingenuity.

Editors Note: $200 buys a lot of backups and you don't have to stay up late at night trying to crack the copy protection. What is your time worth?

The manual assumes a bit more knowledge than that of the Average Apple /// user, but should be useful nevertheless. Along with the program you get a listing of the techniques which must be used when copying certain programs.

Fortunately all of the above mentioned programs are listed and it was much easier than starting from scratch. Locksmith also contains a few very helpful utilities such as a disk rotation speed check and disk surface certification.
All in all very helpful.

This also works in Business Basic. The DOWNLOAD program on the Basic disk (1981 version) allows you to load different character sets. Try downloading the "apple" character set. Then using the open-apple key type all the control characters. You will notice that there are some useful characters for those who like to write blackjack programs.

Tidbit 2

This tidbit is for all those Pascal system users (Business Graphics, Utilities) who are a bit miffed at the fact that the Alpha lock key is inactive when running Pascal programs. This is an attribute of the particular .console in the SOS.DRIVER in the boot diskette. If you want the Alpha lock key to be active, use SYSTEM CONFIGURATION to delete the .console of the present SOS.DRIVER for the particular program diskette, and replace it with Apple Writer's .console. This did not seem to work with BASIC's .console, although I cannot figure out why. Any ideas?

Editors Note: We are looking forward to receiving contributions for this column. "Stems and Seeds" has also been suggested as a name for this column. Indicate on your contributions whether they are for "Stems and Seeds" or "Tidbits". We will use the name which is most popular.

- /// -

open apple gazette

1850 Union Street #494
San Francisco CA 94123

Original Apple ///rs

CLUB INFORMATION

MEETINGS

Meetings are held at 7:30 PM on the third Wednesday of each month. The location is the Board Room of the California Bar Association offices at 555 Franklin Street in San Francisco.

MEMBERSHIP

Annual membership dues are $30 from the date application received. Your check payable to the Original Apple ///rs may be mailed to the address below.

OPEN APPLE GAZETTE POLICTY

All manuscripts, photographs, and other materials are submitted free and released for publication. They become the proporty of the Original Apple ///rs and the Open Apple Gazette. Authors should clearly mark all material submitted for publication so that credit may be given.

The publishers/editors do not necessarily agree with, nor stand responsible for, opinions expressed or implied by other than themselves in this publication.

The Original Apple ///rs is a non-profit organization comprised of, and supported by, Apple /// owners and users. The Original Apple ///rs is run by volunteer officers and committees, and the club endeavors to aid other Apple users through this educational publication - "OPEN APPLE GAZETTE." Address all inquiries to: Original Apple ///rs, 1850 Union Street #494, San Francisco, CA 94123.

REPRINT POLICY

All articles appearing in the Open Apple Gazette not copyrighted by the author may be reprinted by another non-profit Apple user group so long as proper credit is given to both the Open Apple Gazette and the author. Proper credit is defined as article title, author, and the words "Printed from VOL X, NO Y of the Open Apple Gazette." Permission to reprint a copyrighted article may be obtained by writing to the author c/o the Original Apple ///rs.

ARTICLE SUBMISSION POLICY

The Open Apple Gazette welcomes any and all articles dealing with the Apple /// Computer and its associated hardware and software. Articles should be submitted on diskette as an ACSII text file, such as those produced by either Word Juggler, Apple WRiter /// or the Pascal Editor. Typewritten double spaced articles are also acceptable.

OFFICERS

PRESIDENT	Don Norris	(415) 921-3774
VICE PRESIDENT	Kent Hockabout	(415) 521-5414
TREASURER	Julia Amaral	(415) 383-3088
SECRETARY	Charles Coles	(415) 386-8623
CONSULTANTS	Ransom Fields	
	Ken Silverman	

Back Issues

Volume 1, Number 1	$3.00 each
Volume 1, Numbers 2-5	$4.00 each

Mail Requests for Back Issues to:

Open Apple Gazette
1850 Union Street, #494
San Francisco, CA 94123

-///-