Oct. 1998 - Feb. 1999 Vol. 6 Number 2 $5.00US (North America)

the world of 68' micros

Support for Motorola based computer systems and microcontrollers and the OS/9 operating system

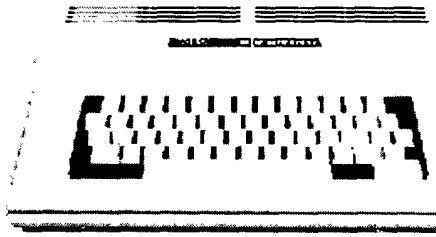
Greetings!

Thanks for sticking with "the world of 68' micros".

Be sure to check the "Editor's Page" for the latest on changes and ideas. Also, please take note of my contact information listed below.
All renewals from this point on should be sent to the address below, and any checks should be made payable to Stephen Disney.

***NOTE* - This contact information is NEW. Please keep it somewhere handy so that you'll know where to send those renewals and letters!**

Stephen Disney
273 Peach Orchard Rd.
Statesboro, GA 30458
(912) 865-9077
email: disneys@bulloch.com (all attachments)
or disneys@hotmail.com



CONTENTS

POSTMASTER:
If undeliverable return to:
268'm
273 Peach Orchard Rd.
Statesboro, GA 30458-6682

If your address is incorrect, send me a postcard!

The Editor's Page

NOTE from Frank:

There was no OCT/NOV/DEC issue, just in case anyone was wondering. I thought I'd be able to get one out, but just couldn't. This will be made up for by extending existing subscriptions by one issue. I'm still around and contributing to the magazine, as well as assisting Stephen (I'll be listed as "associate editor" for the next few issues. Our agreement is that Stephen is required to provide only one year of magazines. It will be up to the readers to make it worth while for him to continue printing beyond January of 2000. I hope he is well supported and continues! I will be paying all printing/shipping costs for his first issue. If you need to reach me for any reason, I can be contacted at:
FARNA@worldnet.att.net.
May you all have a happy New Year!!!

Hello,

In case you haven't heard, my name is Stephen Disney. I am the new editor and I hope that each of you will continue to support "the world of 68' micros" for at least the next year.

A lot happened in the CoCo community in 1998. We had two great fests. We lost a couple of beloved friends (look for a memorial hopefully in the next issue). There was a little animosity, and a lot of generosity. However, if 1998 will be remembered for anything, it ought to be these two things: great ideas and new users. Over the course of the next few issues, I hope to have articles that will stress these two items. Remember when OS/9 levels one and two were both successfully burned into an EPROM? How about the CoCo that was installed into a Toyota truck? Or maybe a CoCo with a built in 720k disk drive installed into a camper? Would you like to build a CoCo 3 from scratch? It's been done! How about that IDE interface? Sure, we haven't received it yet, but maybe soon... We may very well have a 1 gig partitioned hard drive, LS-120, or CD-ROM drive attached to CoCos by this time next year. There were also many 2 meg ram boards built last year. OS/9 level 2, v.3 was finally released. And that's just the tip of the iceberg. The new CoCoists are hunting all over for answers to some pretty common problems and some more complex ones as well. I hope to cover all of the above in the next few issues, as well as bring you the latest news and up-to-date information, boundry-pushing programs, hardware and software reviews, and some new features.

Yes, there will be changes. For instance, starting with this issue the names of new subscribers or renewers (with any information that they'd like to make public; e.g. email, interests, etc.) will be published in the magazine. Those receiving there last paid issue will also find their name. Each issue will have an interview with one of the "Great CoCoists". I am also starting a small "Coming Attractions!" spot to hype upcoming articles or events. You will find a great "One Line Wonder!" on page 6. I hope to include one or two in each issue, so SEND THEM IN!. In the next issue, I hope to start a small spot called "Jokes and Quotes". This section will feature a great quote (computer related when possible) and a computer related joke. Come on everyone! This is your big chance to share that great Microshaft joke or rag on Windoze '98. The most important section to me, however, will be an oldie but goodie. That's the Letters section. I want feedback. I would like you to consider it your obligation to send me at least one letter or email a year! Tell me what you like or dislike. How can I improve? I need your votes of confidence.

Speaking of feedback, here's a question. Would you like to see the microDisk feature return? How about an on-line accessible version? Or maybe both ways? Let me know. Don't be afraid. Keep in touch!

the world of 68' micros

Publisher:
THUNDER SOFT
273 Peach Orchard Rd.
Statesboro, GA 30458-6682
email: disneys@bulloch.com
http://www.home.pon.net/kf6ntg/68micros

Editor-in-Chief:
Stephen T. Disney
Associate Editor:
Francis (Frank) G. Swygert

Subscriptions:
US/Mexico: $22 per year
Canada: $28 per year
Overseas: $48 per year (airmail)
Back issues within 2 years are $5 each.
Older issues are $3 each.
An order of 6 issues will not surpass $22.
Shipping varies depending on destination.

Advertising Rates:
Contact the publisher. We have scales to suit every type of business. Special rates for entrepreneurs and "cottage" businesses. Rates include a subscription and may be negotiable on an individual basis.

Contributions:
All contributions are welcome. Submission constitutes a warranty on the part of the author that the work is original unless otherwise specified. Publisher reserves the right to edit or reject material without explanation. Editing will be limited to corrections and fitting available space. Authors retain copyright. Submission gives the publisher first publication rights and right to reprint in any form with credit given to the author. Each contributor will receive a complimentary issue.

General Information:
Current publication frequency is bimonthly. Frequency and prices are subject to change without notice. All opinions expressed herein are those of the individual authors, not necessarily of the publisher. No warranty as to the suitability or operation of any software or hardware modification is given nor implied under any circumstances. No further support should be expected as the result of an ad or article published herein. Use of any information in this publication is entirely at the discretion and responsibility of the reader.

All trademarks/names are the property of their respective owners.

VENDOR AD POLICY - effective 1/1/99

DEFINITION OF A VENDOR
In the eyes of "the world of 68' micros" a vendor -
a) sells more than one item (exceptions made only for exceptional items), and
b) sells said items for at least six months.

AD RATES
For a year of ads (six issues) paid in advance (includes a copy of each issue) -
1/4 page ad (3-1/2"x4-3/4", 1-3/4"x10", or 7-1/4"x2-1/4") = $50
1/2 page ad (7-1/4"x4-3/4" or 3-1/2"x10") = $80
Entire page ad (7-1/4"x10") = $140

For a single issue paid in advance (includes a copy of the issue) -
1/4 page ad (see above) = $15
1/2 page ad (see above) = $25
Entire page ad (see above) = $35

Non-vendor spot ads (maximum of 1/8th page - formatted by editor) will be FREE to subscribers or $5 to non-subscribers.

Rates may be negotiable on an individual basis.
The value of a hardware/software trade will be taken off the ad price.

Letters

(ed. - This following letter was for Frank.)

I for one hate to see you go but, I understand your reasons and I am confident that Stephen Disney will do just as good a job as you did. As long as there is a TWO68K, I'll keep renewing.
Thanks for all you've done Frank and good luck to Stephen.
Jim Cox

- ed. - Thanks for the vote of confidence Jim. It's very important for me to know that everyone is dedicated to the continuity of this magazine. Also, you got it backwards. As long as 268'm has subscribers, I'll keep printing it. ;-)

Happy New Year, Stephen -

I used to subscribe and have the first few years. I let the subscription lapse when it went to every other month. I would like to subscribe and also get the missing back issues. The home page doesn't work for me. It still gives Frank's E-Mail address. Does this mean He is handling the back issues? Let me know where to send the money, and how much.
Garry - the Florida CoCo Nut

-ed.- Thanks again for the vote of confidence. All money should now be sent to my address. It is on the front cover. I will be handling all aspects of the magazine. Contact me to order back issues. I will be putting a list of the contents of the previous issues in an upcoming issue.

Hello, Stephen,

A charter subscriber to The World of 68' Micros, I now am about to renew my subscription. The last issue I received, Jul/Aug/Sep '98, mentioned that the price will drop by $2. So I plan to send at least $22 for a year's sub to your Statesboro address. I say "at least," because I have probably missed one or more issues since you became the new editor. Heretofore, back issues were sold for the cover price of $4. But has the cover price also been lowered? How many issues have I missed?
Ad<thanks>vance.
David Baker

-ed.- Guess what! You haven't missed a thing. Please send your renewal to the address on the cover. In case you noticed, the cover price actually went up. This is to discourage people from getting back issues in place of a subscription. The price of older issues went down to $3 though, so purchasing back issues is still relatively inexpensive.

New Subscribers and Renewals!

I would also like to thank the following people for sending in there renewal:

David Baker
David Hazelton
Franklin Sevier

If your name is listed below, this is the last issue of your subscription. I sincerely hope that you will consider a renewal.

Carl Boll
Art Boos
William F. Brown
Eleanor Buck
John Chasteen
Kenneth Drexler
E. J. Haas
William Hamblen
Rodney V. Hamilton
Henry O. Harwell
J. L. Henderson
William Hochstetter
Michael Hollick
Andre Jacobs
James Jones
Kevin Kounovsky
Terry Laraway
Howard Luckey
Roger Merchberger
I. Michaelides
Larry E. Olson
Tony Podraza
Monk Repair
Mark Rosenfield
Kurt Ryman
Ron Shively
James Toth
C. L. Tucker
Rick Ulland
Ray Watts
John-Michael Wong
Bill Yakowenko

UltiMusE-III and UltiMusE-K

Still Available and Better than Ever!

We're Celebrating the Tenth Anniversary of this Classic MIDI Sheet Music Composing OS-9 Application by declaring it FREEWARE!

512K Coco-III and MM/1 Versions and Manual Documents are free to download via FTP from RTSI.COM. If you can't download, then ask a friend. Spread UltiMusE around — It's legal!

It's all mine — and it can be yours.

Find out why dozens of Coco users paid big bucks for UltiMusE back in 1989. Low-cost MIDI keyboards are back — check out The Shack! There will never be a better time to get in on the fun and satisfaction of programming your own music. And I'm still using and upgrading this powerful package.

— Mike Knudsen —

Converting GW-BASIC to DECB Part 2

Frank Swygert

Creating a simple but fast database format

Introduction...

In the last installment I discussed how program parameter setup is accomplished within each GW-BASIC program setup. For the CoCo, a little more work was required for setup so a separate "BOOT" program was used which also included some "speed up" patches. As one can see, the differences in the programs are many and a straight conversion was impossible. It was simpler to rewrite the GWBASIC programs for the CoCo using the GW code as a guide. Notes on the code follows each segment in the listings.

Creating a fast database in BASIC...

Anyone who has programmed in BASIC knows it is far from the fastest language for a database, mainly due to speed. Yet I have had people assume CoCo Family Recorder was written in assembly language because of the unlistable program segment names and because of the relatively fast speed. Part of the speed comes from using the high speed poke and patching SDECB (Super Disk Extended Color BASIC) for reliable double speed disk access. But database access speed comes from the way the database was programmed.

In order to speed record access in the database, the database is given a permanent structure. Blank records are written on the disk in fixed positions. So all the disk access routine has to do is find the file. Each record has a fixed number so is very easy to find. And since the blank records were all written at one time, there is one single file instead of many small individual files. This is why it is important to start building your data from a blank disk -- doing so will make the program even faster because the file is stored in one contiguous area on the disk instead of being in fragments across the disk.

Database Limitations...

There are some limitations inherent with creating blank records ahead of time. One is that you need to make 2-3 backups of your data files. This is always a good idea, but more important in this case. Any given record number is always written to the same track and sector on the disk once the file is created. So if that particular bit of data goes bad, reentering will not help since the same sector will be reused. If that sector or a portion of it is bad it will always be bad. In that case you have to go to one of the backups. This means that high quality disks should be used and regular backups made. This is pointed out in the manual, and hasn't been a problem for any of the 60 or so users. While some have switched to more extensive PC based programs, there are still a few users religiously using there CoCo3 and CCFR.

Another limitation was the number of records that could be kept. The GW program has the ability to change the number of records, but SDECB doesn't. Rather than make the number of records flexible, I decided to simply create the maximum number of records SDECB would handle -- 256. The size of the resulting files dictated that two data disks be used, one labeled PERSFILE and the other MARRFILE. The first contains records of individual persons and the second records of marriages interlinking persons. The data files themselves would only take up the 68 granules that a single 35 track disk has for storage, but an index of each record is kept as an individual file. The indices require only one granule each, putting the data over the 68 granule limit. One user keeps all data on a single 40 track disk, which has 78 granules for storage. If the SDECB ROM has been modified for 40 track access it is safe to simply insert the same disk when creating the data files. If you don't have a modified ROM but have 40 track capable disk drives, a 40 track patch can be added to the BOOT program. I didn't include this initially as there was a possibility that a user may not have 40 track capable drives.

The following pokes are used to set SDECB for 40 track operation. The first set is for version 2.0, the second for version 2.1. Remember, the CoCo3 patches DECB and changes the version number from 1.0 and 1.1 to 2.0 and 2.1. The patches work on a CoCo2 with DECB 1.0 and 1.1 also.

Version X.0
POKE 50952,78 (kill command)
POKE 51083,78 (file allocation table)
POKE 51104,78 (granule allocation table)
POKE 52697,78 (free command)
POKE 53680,40 (backup command)
POKE 54111,78 (copy command)
POKE 54342,39 (DSKI$/DSKO$)
POKE 54642,40:POKE 54677,40
(dskini command)

Version X.1
POKE 50997,78 (kill command)
POKE 51034,78 (file allocation table)
POKE 51183,78 (granule allocation table)
POKE 52917,78 (free command)
POKE 53917,40 (backup command)
POKE 54349,78 (copy command)
POKE 54580,39 (DSKI$/DSKO$)
POKE 54879,40:POKE 54914,40
(dskini command)

Just for your information, the first set of five numbers is the memory location. Note that the locations are different for version X.0 and X.1, the reason for two different sets of POKEs. The second two digit number is a decimal number. "78" refers to 78 granules, 40 to 40 tracks. DSKI$ and DSKO$ starts counting at 0 rather than 1, hence the use of 39 instead of 40.

It would be best to add these commands in the BOOT file before running CCFR. Or write a short BASIC program to make all the POKEs that can be used with other programs as well. If you run CREATMAR and insert the PERSFILE data disk and error will occur when the disk fills. The MARRFILE data file won't be damaged. The disk will fill only if you attempt to create both files on a 35 track disk.

Conclusion...

As in the last issue he listings are side by side beginning on the next page. Compare them line by line when possible. There are some obvious differences, such as the CoCo only allowing only two characters for variable names whereas GW allows up to 40. There are also the GW and MS-DOS setup parameters. Others will be noted within or after the code. Enjoy the coding exercise and address any questions to the editor or directly to me.

Frank Swygert
Box 321
W.R., GA 31099
e-mail: farna@worldnet.att.net
Phone 912-328-7859 (not after 9pm EST!)

Listings on next page

GW-BASIC

IMPORTANT: *Because of overlapping subjects, it is necessary to read the notes in BOTH listings.*

```
100 REM CREATPER Program
110 REM Creates (Formats) a Persons File
120 REM By:  Melvin O. Duke.  Last Updated 2 February 1986.
200 REM Screen Definitions
210 WIDTH "scrn:", 80
220 SCREEN S1,S2,S3,S4
600 REM Titles
610 TITLE$ = "Create a Persons File"
620 TITLE$ = TITLE$ + " ON DISPLAY"
700 REM Terminate if not called from the Menu
710 IF DD.MENU$ <> "" THEN 770
720 COLOR 7,0 : KEY ON : CLS : LOCATE 15,1
730 PRINT "Cannot run the"
740 PRINT TITLE$
750 PRINT "Program, unless selected from the MENU"
760 END
770 REM OK
```

* The above segment reestablishes some of the setup parameters, making sure the screen hasn't been changed. It also prevents running unless started from the menu. I didn't know how to incorporate this into the SDECB program, since SDECB can't pass parameters from one program to another. So the encoded file names serve two purposes -- protection of the code from inexperienced BASIC programmers and preventing the ancillary programs from being run without running the boot and menu programs first.

```
1000 REM Produce the first screen
1010 KEY ON : CLS : KEY OFF
1020 REM Draw the outer double box
1030 R1 = 1 : C1 = 1 : R2 = 24 : C2 = 79 : GOSUB 1300
1040 REM Find the title location
1050 TITLE.POS = 40 - INT(LEN(TITLE$)/2)
1060 REM Draw the title box
1070 R1=3:C1=TITLE.POS-
2:R2=6:C2=TITLE.POS+LEN(TITLE$)+1:GOSUB 1500
1080 REM Print the title
1090 LOCATE 4,TITLE.POS : PRINT TITLE$
1100 LOCATE 5,40-INT(LEN(VERSION$)/2) :  PRINT VERSION$;
1230 REM Draw the Copyright box
1240 R1 = 19 : C1 = 21 : R2 = 22 : C2 = 59 : GOSUB 1300
1250 REM Print the Copyright
1260 LOCATE 20,40-INT(LEN(COPY1$)/2) :  PRINT COPY1$;
1270 LOCATE 21,40-INT(LEN(COPY2$)/2) :  PRINT COPY2$;
1280 GOTO 1700
1300 REM subroutine to print a double box
1310 COLOR P
1320 FOR I = R1 + 1 TO R2 - 1
1330  LOCATE I, C1 : PRINT CHR$(186);
1340  LOCATE I, C2 : PRINT CHR$(186);
1350 NEXT I
1360 FOR J = C1 + 1 TO C2 - 1
1370  LOCATE R1, J : PRINT CHR$(205);
1380  LOCATE R2, J : PRINT CHR$(205);
1390 NEXT J
1400  LOCATE R1, C1 : PRINT CHR$(201);
1410  LOCATE R1, C2 : PRINT CHR$(187);
1420  LOCATE R2, C1 : PRINT CHR$(200);
1430  LOCATE R2, C2 : PRINT CHR$(188);
1440 COLOR W
1450 RETURN
1500 REM subroutine to print a single box
1510 COLOR B
1520 FOR I = R1 + 1 TO R2 - 1
1530  LOCATE I, C1 : PRINT CHR$(179);
1540  LOCATE I, C2 : PRINT CHR$(179);
1550 NEXT I
1560 FOR J = C1 + 1 TO C2 - 1
1570  LOCATE R1, J : PRINT CHR$(196);
1580  LOCATE R2, J : PRINT CHR$(196);
1590 NEXT J
1600  LOCATE R1, C1 : PRINT CHR$(218);
```

SDECB

```
10 REM **** CREATPER Program
20 REM Copyright (c) by F.G. Swygert, September 1991
30 POKE113,0:WIDTH80:CLS6
40 IF PEEK(269)*256+PEEK(270)<>32401 THEN 780
```

* The above PEEK statement checks for the patches made in the MENU program. If the patches haven't been made, then MENU hasn't been run. While not essential for the operation of the CREAT programs, the patches do speed the process. I don't recall exactly which patch the statement checks for, but I believe it may be the double speed POKE.

```
50 ON ERR GOTO 560 : ON BRK GOTO 700
60 LOCATE 21,8 : PRINT "The CoCo Family Recorder Version 1.0"
70 LOCATE 28,12: PRINT "Create a Persons File"
80 LOCATE 13,18: PRINT "Place Blank Disk in Drive 1. Label this disk
PERSFILE."
90 SOUND150,4:LOCATE 27,22 : PRINT "Press any key to continue."
100 EXEC 44539
110 CLS 6
120 CLS 2:SOUND150,4:SOUND150,4
130 LOCATE 7,10 : PRINT "This program FORMATS a Persons-file by
writing new, empty records."
140 LOCATE 6,11 : PRINT "It will destroy any data which exists with the
same record-numbers."
150 LOCATE 5,12 : PRINT "If this is REALLY what you want to do, type
'Y' to continue then press"
160 LOCATE 12,13 : PRINT "the 'ENTER' key. Hit any other key to return to
the MENU"
170 LOCATE 6,15 : INPUT "DO YOU WISH TO CONTINUE";R$
180 IF LEFT$(R$,1)="y" THEN 240
190 IF LEFT$(R$,1)="Y" THEN 240
200 LOCATE 30,17 : PRINT "File was NOT Created"
210 SOUND150,4:LOCATE 27,20 : PRINT "Press any key to continue"
220 EXEC 44539
230 GOTO 530
```

* Always give ample warning before doing any potentially destructive processes! Things like this make the program very "user friendly". If there is a possibility for error, warn! Always assume the user is inexperienced when writing. These warnings may be a slight inconvenience for experienced users, but better to be safe than turn a novice away because of accidentally lost data -- especially when the programmer can prevent such accidents.

```
240 OPEN "D", #1, "PERSFILE:1"
250  FIELD 1,5ASFA$,20ASFB$,30ASFC$,2ASFD$,5ASFE$,5ASFF$,
5ASFG$,11ASFH$,18ASFI$,16ASFJ$,16ASFK$,11ASFL$,18ASFM$,
16ASFZ$,16ASFY$,11ASFP$,18ASFQ$,16ASFR$,16ASFS$
```

* The above line sets the number of spaces and string names. "5ASFA$" reads "5 spaces used as FA$" in plain English. Line 240 opens the data file and names it "PERSFILE" on drive 1.

```
260 FOR I=1TO550
270 TP=I
280 TP$=MKN$(TP)
290 LSET FA$ = TP$
300 TP$ = " "
310 TP = 0
320 LSET FB$ = TP$
330 LSET FC$ = TP$
340 LSET FD$ = TP$
350 LSET FE$ = MKN$(TP)
360 LSET FF$ = MKN$(TP)
370 LSET FG$ = MKN$(TP)
380 LSET FH$ = TP$
390 LSET FI$ = TP$
400 LSET FJ$ = TP$
410 LSET FK$ = TP$
420 LSET FL$ = TP$
430 LSET FM$ = TP$
440 LSET FZ$ = TP$
```

GW-BASIC

```
1610 LOCATE R1, C2 : PRINT CHR$(191);
1620 LOCATE R2, C1 : PRINT CHR$(192);
1630 LOCATE R2, C2 : PRINT CHR$(217);
1640 COLOR W
1650 RETURN
1700 REM ask user to press a key to continue
```

* All the above creates a semi-graphics screen with lots of boxes using the extended IBM character set.

```
1710 LOCATE 25,1
1720 PRINT "Have Data Diskette(s) in Place, then Press any key to
continue.";
1730 K$ = INKEY$ : IF K$ = "" THEN 1730
1740 KEY ON : CLS : KEY OFF
1800 REM Give the User one more chance to protect himself.
1810 LOCATE 10,1
1820 PRINT "This program FORMATS a Persons-file by writing new,
empty records."
1830 PRINT "It will destroy any data which exists with the same
record-numbers."
1840 PRINT
1850 PRINT "If this is REALLY what you want to do,"
1860 PRINT "type y to continue, and press the 'enter' key."
1870 PRINT "Otherwise, type anything else, and press the 'enter' key."
1880 PRINT
1890 INPUT "Enter your desired action.",REPLY$
1900 IF LEFT$(REPLY$,1) = "y" THEN 2000
1910 IF LEFT$(REPLY$,1) = "Y" THEN 2000
1920 PRINT
1930 PRINT "File was NOT Created."
1940 PRINT
1950 PRINT "Press any key to continue"
1960 A$ = INKEY$ : IF A$ = "" THEN 1960
1970 GOTO 2330 'to end the program
2000 REM CREATPER Program Starts Here
2010 OPEN DD.PERS$+"persfile" AS #1 LEN = 256
2020 FIELD 1,5ASF1$,20ASF2$,30ASF3$,2ASF4$,5ASF5$,5ASF6$,
5ASF7$,11ASF8$,18ASF9$,16ASF10$,16ASF11$,11ASF12$,18ASF13$,
16ASF14$,16ASF15$,11ASF16$,18ASF17$,16ASF18$,16ASF19$
2030 REM Write the Persons Records
2040 FOR I = OLD.MAX.PER + 1 TO MAX.PER
2050 TEMP = -I
2060 TEMP$ = MKS$(TEMP)
2070 LSET F1$ = TEMP$
2080 TEMP$ = " "
2090 TEMP = 0
2100 LSET F2$ = TEMP$
2110 LSET F3$ = TEMP$
2120 LSET F4$ = TEMP$
2130 LSET F5$ = MKS$(TEMP)
2140 LSET F6$ = MKS$(TEMP)
2150 LSET F7$ = MKS$(TEMP)
2160 REM all the rest are string
2170 LSET F8$ = TEMP$
2180 LSET F9$ = TEMP$
2190 LSET F10$ = TEMP$
2200 LSET F11$ = TEMP$
2210 LSET F12$ = TEMP$
2220 LSET F13$ = TEMP$
2230 LSET F14$ = TEMP$
2240 LSET F15$ = TEMP$
2250 LSET F16$ = TEMP$
2260 LSET F17$ = TEMP$
2270 LSET F18$ = TEMP$
2280 LSET F19$ = TEMP$
2290 LOCATE 23,1 : PRINT "Writing Record Number:";I
2300 PUT #1,I
2310 NEXT I
2320 CLOSE #1
2330 KEY ON : CLS : KEY OFF : LOCATE 21,1
2340 PRINT "End of Program"
2350 RUN DD.MENU$+"menu"
```

SECB

```
450 LSET FY$ = TP$
460 LSET FP$ = TP$
470 LSET FQ$ = TP$
480 LSET FR$ = TP$
490 LSET FS$ = TP$
500 LOCATE 27,22 : PRINT "Writing Record Number:";I
510 PUT #1,I
520 NEXT I
530 CLOSE #1
```

* Lines 260-530 writes the datafile to the disk.

```
540 CLS 3 : LOCATE 33,12 : PRINT "End of Program"
550     LOADCHR$(130)+CHR$(32)+CHR$(130)+CHR$(03)+"/
"+CHR$(03)+CHR$(12)+CHR$(13),R
```

* Since the program has ended, I'll return users to the MENU program automatically.

```
560 CLS 2
570 LOCATE 26,8 : PRINT "Error Number ";ERNO;" Has Occured"
580 LOCATE 26,8 : PRINT "Error Number ";ERNO;" Has Occured"
590 LOCATE 18,10 : PRINT "ERROR NUMBERS:"
600 LOCATE 18,11 : PRINT "3 = Out of Data        6 = Out of Memory"
610 LOCATE 18,12 : PRINT "27 = Bad Record Number   17 = Bad File
Data"
620 LOCATE 18,13 : PRINT "20 = I/O Error         21 = Bad File Mode"
630 LOCATE 18,14 : PRINT "25 = Disk Full         26 = File Not Found"
640 LOCATE 16,16 : PRINT "<<** Check for correct disk(s) in drive(s)!
**>>"
650 LOCATE 25,18 : PRINT "Place Program Disk in Drive 0"
660 SOUND150,4:LOCATE 15,20 : PRINT "PRESS ANY KEY TO RETURN
TO MENU, BREAK TO TRY AGAIN"
670 EXEC 44539
680 CLOSE : CLS 3 : LOCATE 33,12 : PRINT "End of Program"
690 GOTO550
```

* Lines 550-690 are repeated at the end of every program segment. I am giving all readers permission to use this error trapping routine in their programs, all I ask is a credit byline somewhere in the program listing. Of course you'll want to change line 550 if a different program needs to be run, or delete line 690 to end the program and clear the screen with lines 700-750. You may want to change line 760 to "CLS:END". No error handling routine is needed in GW, it has built in error trapping.

```
700 CLS 2
710 LOCATE 20,12 : INPUT "Do you wish to halt the program, Y or N"; R$
720 IF R$="Y" OR R$="y" THEN 750
730 IF R$ = "N" OR R$ = "n" THEN 770
740 GOTO 710
750 CLOSE:CLS3:LOCATE33,12:PRINT"End of Program"
760 GOTO550
770 CLOSE:CLS6:CLEAR 500:GOTO 40
780 CLS2:SOUND1,8:LOCATE 25,12:PRINT"PROGRAM MUST BE RUN
FROM MENU!"
790 GOTO550
```

One Line Wonders!

SUNSET OVER THE CITY - from Robert Rice - FPO Miami, FL
(-ed.- The following should be typed in one line.)

```
0PMODE4,1:PCLS:SCREEN1,1:CIRCLE(126,65),40,3,.7:POKE178,2:
PAINT(126,52),,3:POKE178,1:LINE(0,60)-(255,191),PSET,BF
:FORA=0TO356:CIRCLE(126,60),A,0,.4,0,.5:NEXT:FORA=ITO12:
X=RND(75):Y=RND(50):Z=RND(50):PSET(X+160,Z):PSET(X,Y):
NEXT:LINEINPUTA$
```

If you have a one or two liner that you would like to share, SEND IT IN! Have some FUN inventing one - THEN SHARE IT WITH US!

FARNA Systems

Your most complete source for Color Computer and OS/9 information!

Post Office Box 321
Warner Robins, GA 31099
Phone: 912-328-7859
E-mail: dsrtfox@delphi.com

ADD $3 S&H, $4 CANADA, $10 OVERSEAS

BOOKS:

Mastering OS-9 - $20.00
Completely steps one through learning all aspects of OS-9 on the Color Computer. Easy to follow instructions and tutorials. With a disk full of added utilities and software!

Tandy's Little Wonder - $25.00
History, tech info, hacks, schematics, repairs,... almost EVERYTHING available for the Color Computer! A MUST HAVE for ALL CoCo aficionados, both new and old!!! This is an invaluable resource for those trying to keep the CoCo alive or get back into using it.

Quick Reference Guides
Handy little books contain the most referenced info in easy to find format. Size makes them unobtrusive on your desk. Command syntax, error codes, system calls, etc.
CoCo OS-9 Level II : $5.00
OS-9/68000 : $7.00

Complete Disto Schematic set: $15
Complete set of all Disto product schematics. Great to have... needed for repairs!

SOFTWARE:

CoCo Family Recorder: Best genealogy record keeper EVER for the CoCo! Requires CoCo3, two drives (40 track for OS-9) and 80 cols.
DECB: $15.00 OS-9: $20.00

ADOS: Best ever enhancement for DECB! Double sided drives, 40/80 tracks, fast formats, extra and enhanced commands!
Original (CoCo 1/2/3) : **$10.00**
ADOS 3 (CoCo 3 only) : **$20.00**
Extended ADOS 3 (CoCo 3 only, requires ADOS 3, support for 512K-2MB, RAM drives, 40/80 track drives mixed) : **$30.00**
ADOS 3/EADOS 3 Combo: $40.00

Patch OS-9 - $7.00
Latest versions of all popular utils and new commands with complete documentation. Auto-installer requires 2 40T DS drives (one may be larger).

TuneUp : $10.00
Don't have a 6309? You can still take advantage of Nitro software technology! Many OS-9 Level II modules rewritten for improved speed with the stock 6809!

NitrOS-9:

Nitro speeds up OS-9 from 20-50% depending on the system calls used. This is accomplished by completely rewriting OS-9 to use all the added features of the Hitachi 6309 processor. Many routines were streamlined on top of the added functions! The fastest thing for the CoCo3! Easy install script! 6309 required.
Level 3 adds even more versatility to Nitro! RBF and SCF file managers are given separate blocks of memory then switched in and out as needed. Adds 16K to system RAM... great for adding many devices!
NitrOS-9 V.2.0: $10.00
NitrOS-9 Level 3: $10.00

ONLY 3 LEFT!

That's right, only 3 "Mastering OS-9" book and disk sets are left! I'll print no more!

New from...

THUNDERSOFT

!FLASH! - Available for the first time in 12 years....Green Mountain Micro's famed "Learning the 6809" set. This set contains an in-depth text book autographed by the author (Dennis Báthory-Kitsz), a set of Motorola data sheets, and 12 cassettes featuring two thirty minute lessons each (or one MPEG3 CD with all 24 lessons). Each lesson is packed with a mixture of spoken text, questions, and loadable programs. You will not have to type any demo programs. All examples are already in the lessons! A Hot CoCo reviewer once referred to the set as "a first-rate, honest product...9.8 out of a possible 10". Make a New Year's resolution to finally learn the 6809 processor, then buy this set to do it right. Once you've learned the 6809, adding programming knowledge of the disk system, GIME, and 6309 should be a snap. This set is for use on a CoCo 1/2 with a tape setup and EDTASM. We do not guarantee use on a CoCo 3. We are hoping to provide updates in the future, but at this time neither ThunderSoft nor the author are offering support. This is a time tested set. We feel there should be no problems.

Learning the 6809 set (as described above - please specify tapes or CD) - $50 (plus $5 S&H in the U.S.)
Learning the 6809 text only (for replacement purposes) - $10 (plus $3 S&H in the U.S.)

Make all checks payable to
Stephen Disney and mail to:
ThunderSoft
273 Peach Orchard Rd.
Statesboro, GA 30458

And now for the stupid and amazingly obvious disclaimer:
"The Thundersoft (distribution and publishing) run by Stephen T. Disney (editor of "the world of 68' micros") is not in any way affiliated with any other company by the same name." - S.D. (ThunderSoft)

microNotes

Notes and news from all over related to the CoCo, OS/9 and of interest to readers. Got something interesting to let the CoCo/OS-9 world know about? Send it to us!

New Pac-man distributor!

Jim Davis has taken over from Rick Cooper as the official distributor for Pac-man along with all remaining CoCo-Pro and Sundog products. As if that wasn't enough, he has also taken over the CFDM disk magazine! Obviously someone with lots of spare time on his hands.....not! Contact Jim Davis for more information regarding CFDM, I highly recommend it!! Thank you to Rick Cooper for all his hard work acting as the U.S. Pac-man distributor and for his support to the CoCo community! The full registered version of Pac-man is available for the reasonable price of $20 from:

Jim Davis
(e-mail: gearboxed@geocities.com)
P.O. BOX 1704
NIXA, MO. 65714

New Micros, Inc., has recently introduced a system for real time programming called IsoMax which is further explained at their web page - http://www.newmicros.com/isomax.html. They are looking for your comments, all and sundry, on this page and this concept. "Our idea is to help real time programming develop solutions using what we call IsoStructure. Generally, IsoStructure makes problems involving concurrency much easier to solve. It improves fault tolerance by not having all state information stored on the CPU's program counter register. IsoStructure permits multiprocessing across non-similar processors, and compiling into hardware, as well as other nifty features." If you have an interest in IsoStructure, please contact them for more information and documentation.

Chet Simpson has Doom and Doom II running on both the x86 and PPC versions of OS-9. They are currently supporting a wide variety of configurations (minimum 16mb recommended for ROM based systems) with both 256 color displays as well as 16bit. It has been tested with Doom I and Doom II data files as well as total conversion WADS such as "Alien". It has been run on both the MBX boards and the Enhanced OS-9 for x86 release. Stay tuned for more great OS-9/x86 news!

I have made a new product called the "Pro-Tector". What this product does is buffer the address bus in the CoCo3. I have used this many times when troubleshooting bad boards that would normally blow out the processor. The end result is a protected circuit. Since the 63x09ep is getting harder and harder to find, we now have to get them from S. Africa. I thought this would be a good way to save that investment in case of a local disaster on the address bus of your CoCo3. The cost of this unit is $18, plus $4 shipping within the US. I will update my web page when I get my camera back on line. If you are interested in this product please Email me privately and not to this list. I'm trying to get an approximate board count so I can send it off to the board manufacturer.

- Mark Marlette (Cloud Nine)

(-ed.- See this issue's Cloud Nine ad.)

October 12, 1998

DES MOINES, Iowa - Microware Systems Corporation (NASDAQ: MWAR), today announced their superior real-time operating system (RTOS) OS-9 / Hawk supports Motorola's new MPC8260 PowerQUICC II processor. Designed for communications applications such as remote access servers, LAN-to-WAN bridges, cellular base stations, and telecom switch controllers, the PowerQUICC processor is the latest of Motorola's next generation products.

Motorola's PowerQUICC II combines a high-speed PowerPC core, a powerful communications engine processing up to 710 megabits of data per second and a circuit board's worth of system interface and control functions on a single chip. Its unparalleled degree of system integration can reduce a system's component count by up to six chips, shorten customer's time-to-market by six months, and slash a system's component cost by as much as 70 percent.

The MPC8260 PowerQUICC II microprocessor is the first member of Motorola's MPC8000 series, a high-performance extension to Motorola's popular MPC800 series of integrated PowerPC processors. The new PowerQUICC II family offers high performance for the latest high-bandwidth internetworking and telecommunications systems, continuing Motorola's legacy of proving highly integrated and powerful communications processors.

Major communications equipment makers such as Alcatel, Bay Networks, Fujitsu, Italtel, Lucent Technologies, Motorola, Newbridge Networks, Nokia and Siemens are examining the PowerQUICC II processor for their next generation systems.

For more information on Microware's products, visit www.microware.com; send e-mail to info@microware.com; or call 1-800-475-9000 or 515-223-8000.

Adventures in Assembly, Part 3

Art Flexser

Using a Bubble Sort for strings

This tutorial continues the discussion of machine-language sorting techniques that was begun in the last issue. Be sure and read that tutorial before this one!

In this tutorial, a string sort is demonstrated, using the same bubble-sort algorithm that was used in Part 2 to sort single bytes in screen memory. The string sort is more complicated, since we are dealing with multiple-byte quantities to be sorted. The string sort will allow us to sort a string array, or a portion of such an array, into alphabetical order (or more precisely, into order according to ASCII codes).

The string sort also differs from the single-byte sort of Part 2 in that the strings will be sorted indirectly, by swapping pointers that point to the string data rather than by sorting the string data itself. Swapping pointers is much more efficient than swapping the strings themselves, since, by this method, long strings take no more time to sort than short strings. Also, the fact that the strings may vary in length creates no difficulties for this method.

Here is how BASIC treats strings. Each time you create a string from a BASIC program, say by inputting the value "ABCDEF" in response to an INPUT A$ prompt, things happen in two different areas of memory. If A$ has been newly defined, a 5-byte entry is created for this variable in the variable storage space immediately above where your program resides in memory. This 5-byte entry is called a string descriptor (the address of this descriptor is the value returned by BASIC's VARPTR function).

Only three bytes of the 5-byte string descriptor for A$ are actually used; the other two are there simply for conformity with the format in which floating-point variables are stored, since a floating point number takes up five bytes. The first of the 5 bytes for a string variable descriptor contains the length of the string. The other two bytes of the descriptor that are used, the third and fourth bytes, contain a pointer to the actual string data. This string data (the ASCII string "ABCDEF") is generally stored in the area reserved for string storage at the top of available memory.

Thus, to swap values between two string variables, say to swap A$ and B$, all we need to do is swap both the first bytes of the two descriptors (the string lengths for A$ and B$), and to swap bytes 3-4 of the two descriptors (the pointers to the string data for A$ and B$). When this has been done, A$ will have the value formerly assigned to B$ and vice versa, even though the string data itself has not been touched.

So far, we have been discussing scalar string variables, but our sort program works with string arrays. The only difference that this fact makes is that we can depend on the string descriptors for the array elements to be consecutively stored in memory: the descriptor for A$(3) right after that for A$(2), etc., five bytes per array element. For a multidimensional array, the order of storage is with the first subscript varying fastest: A$(0,0), A$(1,0),...A$(n,0),A$(0,1), A$(1,1),...A$(n,1), A$(0,2),...., etc.

Our sort program will be usable with either single-dimension or multidimensional arrays, so long as the elements being sorted have their descriptors stored consecutively in memory. In other words, elements in a multidimensional array can be sorted by first subscript only. That is, we could sort A$(0,0) through A$(n,0) with one call to our sort routine, A$(0,1) through A$(n,1) with another call, etc., but we could not sort A$(0,0), A$(0,1), A$(0,2), etc., because these are not consecutively stored.

Using the Sort Routine

The sort routine MLSORT is designed to be called from a BASIC program using BASIC's USR function with a string argument, as in Y$ = USR0(A$(0)), where the argument is a string array element. This string array element tells the sort routine which string array we wish to sort, and what element of the array we wish the sort to begin at.

That is, if we wanted to sort only array elements A$(21) through A$(30), the argument would be A$(21). We also need to tell the sort routine how many array elements (10, in the example of A$(21) through A$(30)) we wish to include in the sort. Unfortunately, BASIC permits user-defined functions to have only one argument, so passing the second argument to our sort routine is more awkward than would otherwise be the case.

The simplest method is to have BASIC POKE the argument into some location (or rather, pair of locations, since we do not wish to limit our sort to 256 elements or less) that the ML routine knows about. This means that the user's BASIC program must convert N, the number of array elements to be sorted, into a most-significant byte and a least-significant byte to be POKEd into two consecutive memory locations.

The BASIC code for performing this conversion generally goes something like this:

MS=INT(N/256):LS=N-256*MS

(An alternate expression for LS is LS=N AND 255. See Part 2 for an explanation of the logical AND operation if this expression is not clear to you.) Here, MS and LS stand for the most significant byte (MSB) and least significant byte (LSB) of N. For example, if the hex equivalent of N is &H1287, MS=&H12 and LS=&H87.

Our sort routine will reserve the top two locations of user RAM ($3FFE-F in a 16K CoCo, or $7FFE-F in a 64K machine) to accommodate the MSB and LSB of the number of array elements to be sorted, which will be poked in from BASIC.

Comments on TUTA3.SRC

On entry to the routine, the X register will contain a pointer to the string descriptor of the array element specified in the USR argument, the lowest array element to be sorted. The bit of magic that accomplishes this is written into the ROM code for the USR function, so we need not go to any special trouble with VARPTR or anything to get X pointed to the address of the start of the bunch of string descriptors that we want to fiddle with. However, finding the RAM location of the END of the array (or rather, the end of the array portion that is to be sorted) does involve a bit of trouble, since we need to take the number of elements to be sorted, multiply by 5 (since there are 5 bytes per array descriptor), and add this product to the address that was in X when the routine was entered, the address of the descriptor for the lowest element to be sorted. The routine needs to know the location of the end of the array in order to efficiently perform the comparisons that determine when the inner and outer loops reach their respective upper limits (this check could also be done using decrementing 16-bit counters to keep track of the number of iterations, but figuring out the position of the end of the array in advance gives a routine that runs faster and is also easier to write).

Anyway, as I was saying....finding the location of the end of the array (er, "end of the string descriptors for the portion of the array that is to be sorted", but I'll just refer to it as the end of the array from now on, if you don't mind) requires a multiplication by 5. The 6809 conveniently has a built-in multiply instruction (MUL), but unfortunately, it works on 8-bit quantities only, and we need to multiply a 16-bit quantity by 5. This is accomplished by multiplying the LSB by 5, then the MSB by 5, adding in the carry from the first MUL (note the use of self-modifying code to store the resulting product as the operand of CMPU and CMPX in the sort routine, replacing dummy $FFFF's).

There is a check to make sure that there is no carry from the second multiplication, since that would imply that our result was a 24-bit quantity and that the array to be sorted took up more than 64K. An occasional check for screwy results is an excellent idea in utility routines that are subject to user errors, since ML routines that have gone awry have an unfortunate tendency to crash the machine if they are allowed to get too far astray. In this spirit, there are also checks to ensure that the user-specified array portion lies between the addresses specified in $1D-E and $1F-20, BASIC's pointers to the start and end of array variable space. If any of these checks fail, the program exits to ROM address $B44A, which aborts the user's BASIC program with an FC error.

At the heart of the sort is the SWAP subroutine, which interchanges two strings if the string pointed to by X is alphabetically greater than that pointed to by U. As a first step, this subroutine ensures that X, rather than U, initially points to the string with the shorter length, calling the REVER1 subroutine to interchange the two strings (by swapping their length bytes and data pointers) to accomplish this if necessary. This initial step will simplify things later, since if in comparing the strings byte by byte

we come to the end of the X-string with no bytes having mismatched, we can simply exit and know that the shorter string will appear first in the sort ("ABC" alphabetically precedes "ABCD".)

After this initial step, X and U are made to point to the actual string data itself, rather than the corresponding descriptors, and a byte-by-byte comparison takes place of corresponding character positions in the two strings. If the two characters in a particular position match, the comparison must go on to check the following character position to determine which string is alphabetically first.

If a character position is reached where a mismatch is found, the subroutine performs an interchange of the two string descriptors if necessary, to make X point to the string with the lower-valued character, and then returns (this operation requires setting X and U to point to the descriptors again rather than to the string data). Finally, as noted above, if the end of the X-string (which we ensured was the shorter in length) is reached, the subroutine simply returns.

If the last explanation of the swap routine was less than clear, the following little BASIC program does pretty much the same thing as the SWAP subroutine and may be easier to follow the logic in. The main difference is that the REVER1 subroutine used by SWAP interchanges the descriptors, whereas the corresponding BASIC GOSUB in line 100 interchanges the strings themselves, which is much less efficient.

```
10 INPUT X$,U$
20 IF LEN(X$)>LEN(U$) GOSUB 100
30 FOR I=1 TO LEN(X$)
40 IF MID$(X$,I,1)>MID$(U$,I,1) GOSUB
100:GOTO70
50 IF MID$(X$,I,1)<MID$(U$,I,1) THEN 70
60 NEXT I
70 PRINTX$;" PRECEDES ";U$:GOTO10
100 'SUBROUTINE TO INTERCHANGE X$ AND
U$
110 T$=X$:X$=U$:U$=T$:RETURN
```

A demo in BASIC for you to use...

The following is a demonstration BASIC program that uses the ML string sort to order 500 randomly-generated two-character strings. This program displays BASIC's array area in PMODE4 while the sort takes place, so you can watch the string descriptors get swapped around as the sort progresses.

```
10 'ML STRING SORT DEMO (BUBBLE SORT)
20 'ART FLEXSER, MIAMI, JULY 1985
30 'SEE MLSORT.DOC FOR HOW TO USE THE
MACHINE-LANGUAGE SORT ROUTINE IN
YOUR OWN PROGRAMS
40 PMODE0:PCLEAR1:GOSUB320:DEFUSR=
&H3F7E+OF:CLEAR4500,&H3F7E+OF
50 GOSUB320 'THIS 2ND GOSUB NEEDED
BECAUSE "CLEAR" RESETS ALL VARIABLES
60 FORI=&H3F7E+OFTO&H3FFD+OF:READP$:
POKEI,VAL("&H"+P$):NEXT
70 DATA 9C,1D,25,45,EC,8D,0,78,10,83,0,1,
23,3A,86, 5,3D,34,6
80 DATA E6,8D,0,69,86,5,3D,EB,E4,E7,E4,4D,
26,28,1F,10,E3,E1,10
90 DATA 93,1F,22,1F,ED,8D,0,F,83,0,5,ED,8D,
0,F,33,5,8D,13
100 DATA 33,45,11,83,FF,FF,25,F6,30,5,8C,
FF,FF,25, ED,39,7E,B4,4A
110 DATA 34,50,E6,84,E1,C4,23,2,8D,17,E6,
84,27,F,AE,2,EE,42,A6
120 DATA 80,A1,C0,22,7,25,3,5A,26,F5,35,
D0,35,50,A6,84,E6,C4,A7
130 DATA C4,E7,84,EC,2,10,AE,42,ED,42,10,
AF,2,39
140 N=500
150 DIM A$(N)
160 PRINT"GENERATING RANDOM ARRAY":
PRINT"OF";STR$(N);" TWO-CHARACTER
STRINGS:"
170 FORI=1 TO N
180 'GENERATE RANDOM TWO-LETTER
STRINGS
190 GOSUB330:A$(I)=CHR$(C)+CHR$(D):
PRINTA$(I)" ";:NEXT
200 PRINT:PRINT"PRESS ANY KEY TO START
SORT"
210 IF INKEY$="" THEN 210
220 PRINT"SORT IN PROGRESS...."
230 POKE&HB6,4:POKE&HBA,PEEK(&H1D):
SCREEN1,1 'DISPLAY ARRAY VARIABLE
AREA IN PMODE 4
240 MS=INT(N/256):LS=N AND 255
250 POKE&H3FFE+OF,MS:POKE&H3FFF+OF,
LS
260 X$=USR(A$(1)) 'SORT DEM STRINGS!
270 PRINT:PRINT"SORT COMPLETE":PRINT
280 PRINT"PRESS ANY KEY TO SEE":PRINT
"SORTED ARRAY"
290 IF INKEY$=""THEN 290
300 FORI=1TON:PRINTA$(I)" ";:NEXT
310 END
320 IF PEEK(&H74)>&H3F THEN OF=&H4000:
RETURN ELSE RETURN 'SET OFFSET TO
&H4000 FOR 32/64K MACHINES (&H74-5
CONTAINS "END OF RAM" ADDRESS)
330 C=&H40+RND(&H1A):D=&H40+RND
(&H1A):RETURN
```

The statements in lines 40-130 of MLSORT.BAS and the subroutine in line 320 are suitable for general use. They will cause the ML routine to be poked in to the highest available user RAM for the memory size of your machine, will protect the routine from BASIC, and will set up things so the routine can be called with the USR function.

Depending on the memory demands of your particular application, you may wish to adjust the PCLEAR argument and the first argument of the CLEAR statement (which determines how much space is reserved for string storage). If your program seems to "hang up" for long periods and then resume by itself, you are experiencing delays due to garbage collection (reorganizing of the string storage area to reclaim wasted space). Such delays may be alleviated or eliminated by increasing the storage allotted to strings in the CLEAR statement, if available memory permits.

To use the sort routine, you must tell it two things: the array element that the sort is to begin with (i.e., the element with the lowest-numbered subscript), and how many consecutively-numbered array elements are to be included in the sort. The sort routine gets told the starting element of the sort by making that element the argument of the USR call. Note that the USR argument must be an array ELEMENT, not simply the NAME of the array. The number of elements to be included in the sort is told to the sort routine by converting this number to a two-byte quantity (most significant byte, least significant byte) and POKing these two numbers into the two bytes directly above the end of the ML routine ($3FFE-F in a 16K machine, $7FFE-F in a 32/64K machine).

Line 240 of MLSORT.BAS does this conversion of the number of elements (N) into the required two-byte format, and line 250 pokes the resulting most-significant and least-significant bytes into the required addresses prior to the USR call in line 260. The quantity X$ which occurs to the left of the USR call in line 260 is a dummy variable whose value is not used.

TUT2A.SRC

```
00100 *STRING SORT USING BUBBLE SORT
ALGORITHM
00110 *ART FLEXSER, JULY 1985
00120
00130 *ON ENTRY, X REGISTER CONTAINS
ADDRESS
00140 *OF DESCRIPTOR OF 1ST STRING
ELEMENT.
00150 *NUMBER OF ARRAY ELEMENTS TO
BE SORTED
00160 *IS POKED BY BASIC INTO LEN AND
LEN+1.
00170      ORG    $3F7E  USE $7F7E FOR
64K
00180 *ORG SELECTED TO MAKE ADDR OF
LEN
00190 *COME OUT TO $3FFE-F OR $7FFE-F
(TOP OF MEMORY)
00200 START  CMPX   $1D    IS ARRAY ST.
LEGIT?
00210      BLO    FCERR  NO, ?FC ERROR
00220      LDD    LEN,PCR GET ARRAY
SIZE
00230      CMPD   #1     RETURN IN TRIVIAL
CASE
00240      BLS    RTN
00250 *MULTIPLY 16-BIT ARRAY LENGTH BY
5
00260 *AND ADD TO ARRAY START TO FIND
ARRAY END.
00270      LDA    #5     B=LSB OF LEN
00280      MUL           MULT BY 5
00290      PSHS   A,B    STORE ON STACK
00300 *A=CARRY, B=LSB OF 5*LEN
00310      LDB    LEN,PCR GET MSB OF LEN
00320      LDA    #5     MULT BY 5
00330      MUL
00340      ADDB   ,S     ADD PREV. CARRY
00350      STB    ,S     B=MSB OF 5*LEN
00360      TSTA          ENSURE PRODUCT <
64K
00370      BNE    FCERR  ELSE GIVE ?FC
ERROR
00380      TFR    X,D    GET PTR TO
ARRAY ST.
00390      ADDD   ,S++
END=START+5*LEN
00400      CMPD   $1F    IS ARRAY END
LEGIT?
00410      BHI    FCERR  NO, ?FC ERROR
00420      STD    ULIM,PCR UPR LIM FOR U
00430      SUBD   #5     BACK 1 ELEMT
00440      STD    XLIM,PCR UPR LIM FOR X
```

```
00450 *NOW BEGINS THE ACTUAL SORT
00460 NEXTX  LEAU   5,X    INITIALIZE U =
X+1
00470 NEXTU  BSR    SWAP   INTRCHG
X&U IF NEC.
00480        LEAU   5,U    INCREMENT U
00490        CMPU   #$FFFF  DONE W/INNER
LOOP?
00500 ULIM   EQU    *-2    ULIM REPLACES
$FFFF
00510        BLO    NEXTU  NO, NEXT U
00520        LEAX   5,X    INCREMENT X
00530        CMPX   #$FFFF  DONE W/OUTER
LOOP?
00540 XLIM   EQU    *-2    XLIM REPLACES
$FFFF
00550        BLO    NEXTX  NO, NEXT X
00560 RTN    RTS           DONE
00570 FCERR  JMP    $B44A  ?FC ERROR
00580 *
00590 *THIS SUBROUTINE IS ENTERED WITH
ADDR.
00600 *OF 1ST STRING DESCRIPTOR IN X,
2ND IN U.
00610 *IF THE X-STRING IS ALPHABETI-
CALLY GREATER
00620 *THAN THE U-STRING, THE TWO
DESCRIPTORS ARE
00630 *SWAPPED, WHICH EFFECTIVELY
INTERCHANGES
00640 *THE TWO STRINGS.
00650 *
00660 SWAP   PSHS   X,U    STORE ENTRY
PARMS
00670 *ENSURE THAT X POINTS TO
SHORTER STRING
00680 *BY SWAPPING IF NECESSARY
00690        LDB    ,X     GET 1ST LENGTH
00700        CMPB   ,U     COMPARE TO 2ND
00710        BLS    ON
00720        BSR    REVER1 SWAP IF
GREATER
00730 ON     LDB    ,X     EXIT IF SHORT
STRING
00740        BEQ    OUT    HAS ZERO
LENGTH
00750 *NOW B CONTAINS LENGTH OF
SHORTER STRING
00760        LDX    2,X    POINT X TO
ACTUAL STR
00770        LDU    2,U    SAME FOR U
00780 LOOP   LDA    ,X+    GET A STR
BYTE
00790        CMPA   ,U+    COMPARE TO
OTHER STR.
00800        BHI    REVERS IF >, INTER-
CHANGE
00810        BLO    OUT    DONE IF <
00820        DECB          =, GET NEXT BYTE
00830        BNE    LOOP   UNLESS STR.
ENDS
00840 OUT    PULS   X,U,PC RETURN
00850 *THIS ROUTINE INTERCHANGES THE 2
STRING
00860 *DESCRIPTORS, THEN RETURNS FROM
SWAP ROUTINE
00870 REVERS PULS   X,U    POINT X,U AT
DESCR'S
00880 REVER1 LDA    ,X     1ST LENGTH
BYTE
00890        LDB    ,U     2ND LENGTH BYTE
00900        STA    ,U     SWITCH LENGTHS
00910        STB    ,X
00920        LDD    2,X    1ST STR. ADDR.
00930        LDY    2,U    2ND STR. ADDR.
00940        STD    2,U    SWITCH STR.
ADDRS.
00950        STY    2,X
00960        RTS
00970 LEN    RMB    2
00980        END    START
```

(continued from page 13)

Listing 1

```
Type        set     Systm+Objct
Revs        set     ReEnt+1
            mod     OS9End, OS9Name,Type,Revs,Cold,256
OS9Name     fcs     "OS9p3"
            fcb     1               edition number

            ifp1
            use     /dd/defs/os9defs.l2
            endc

level       equ     2
            opt     -c
            opt     f

* routine cold
Cold        leay    SvcTbl,pcr
            os9     F$SSvc
            rts

F$SAYHI     equ     $25

SvcTbl      equ     *
            fcb     F$SAYHI
            fdb     SayHi-*-2
            fcb     $80

SayHi       ldx     R$X,u
            bne     SayHi6
            ldy     D.Proc
            ldu     P$SP,y
            leau    -40,u
            lda     D.SysTsk
            ldb     P$TASK,y
            ldy     #40
            leax    Hello,pcr
            os9     F$Move
            leax    0,u
SayHi6      ldy     #40
            ldu     D.Proc
            lda     P$PATH+2,u
            os9     I$WritLn
            rts

Hello       fcc     "Hello there user."
            fcb     $0D

            emod

OS9End      equ     *
            end
```

Listing 2

```
Type        set     Prgrm+Objct
Revs        set     ReEnt+1
            mod     OS9End, OS9Name,Type,Revs,Cold,256
OS9Name     fcs     "SayHi"

            fcb     1               edition number

            ifp1
            use     /dd/defs/os9defs.l1
            endc

level       equ     1
            opt     -c
            opt     f

* routine cold
Cold        equ     *
* The following three instructions are important. They cause the link
* count of this module to increase by 1. This insures that the module stays
* in memory, even if forked from disk.
            leax    OS9Name,pcr
            clra
            os9     F$Link

            leay    SvcTbl,pcr
```

(continued on page 13)

OS-9 401: System Extension Modules

Boisy G. Pitre

Welcome to the first in a possible series of articles dealing with OS-9/6809 internals. If you are like I was when I first discovered OS-9, then you spent quite a bit of time reading the documentation that came with the operating system. The technical information, especially in the OS-9 Level Two manuals, is brimming with details and information that can unlock a wealth of understanding about how OS-9 works. Unfortunately, some of this information can be hard to digest without proper background and some help along the way. This series of articles is intended to take a close look at the internals of OS-9/6809, both Level One and Level Two. So along with this article, grab your OS-9 Technical Manual, sit down in a comfortable chair or recliner, grab a beverage, relax and let's delve into the deep waters!

Assemble Your Gear

For successful comprehension of the topics presented in this and future articles, I recommend that you have the following items handy:

- OS-9 Level Two Technical Reference Manual OR
- OS-9 Level One Technical Information Manual (light blue book) and the OS-9 Addendum Upgrade to Version 02.00.00 Manual
- A printout of the os9defs file for your respective operating system. This file can be found in the DEFS directory of the OS-9 Level One Version 02.00.00 System Master (OS-9 Level One) or the DEFS directory of the OS-9 Development System (OS-9 Level Two).

In this article, we will look at a rarely explored, yet intriguing OS-9 topic: system extensions, a.k.a. P2 modules. When performing an mdir command, you have no doubt seen modules with names like OS9p1 and OS9p2 in OS-9 Level Two (or OS9 and OS9p2 in OS-9 Level One). These modules are essentially the OS-9 operating system itself; they contain the code for the system calls that are documented in the OS-9 Technical Reference documentation. In the case of OS-9 Level One, the modules OS9 and OS9p2 are located in the boot track of your boot disk (track 34). In OS-9 Level Two, OS9p1 (equivalent to the OS9 module in Level One) is found in the boot track while OS9p2 is located in the bootfile. Both of the modules are of module type *Systm* and define the basic behavior and structure of OS-9. Even the module IOMan is a system extension, containing code for the I/O calls in the operating system.

While drivers and file managers have been the most common area to expand the capabilities of OS-9, they are pretty much limited to expanding the functionality of I/O. What system extensions allow you to do is even more powerful: they can add new system calls or even replace existing ones. Such functionality allows you to change the behavior of OS-9 in a very fundamental way. Of course, with such power, caution must be exercised. It is not wise to radically modify the behavior of an existing system call; such an action could break compatibility with existing applications.

What we aim to do in this article is not to replace an existing system call, but rather to add a new system call by looking at the example provided in Tandy's OS-9 Level Two documentation. Although the example is written for OS-9 Level Two, we will look at how it can be changed to run under OS-9 Level One as well. But first, let's get a little background on system calls and how they are constructed in OS-9.

Table 1 – OS-9 Level One System Call Ranges

System Call Range	Function
$00-$27	User mode system call codes
$29-$34	Privileged system mode call codes
$80-$8F	I/O system call codes

Table 2 – OS-9 Level Two System Call Ranges

System Call Range	Function
$00-$7F	User mode system call codes
$80-$8F	I/O system call codes
$90-$FF	Privileged system call codes

The System Call

As an operating system, OS-9 provides system level functions, or system calls to applications. These system calls give applications a base by which they can operate consistently and without fear of incompatibility from one OS-9 system to the next. The system call in OS-9/6809 evaluates to an SWI2 instruction on the 6809, which is a software interrupt. Suffice it to say that when this instruction is encountered by the CPU, control is routed to OS-9, which interprets and performs the system call on behalf of the calling process.

While system calls are generally hidden by wrapper functions or procedures in high-level languages such as Basic09 and C, we can see the system call in its native form by looking at 6809 assembly language. Consider the following assembly source fragment:

```
            lda     #1
            leax    mess,pcr
            ldy     #5
            os9     I$Write
            rts
mess        fcc     "Hello"
```

In the middle of what appears to be normal 6809 assembly language source code is a mnemonic called os9. This is a pseudo mnemonic, since Motorola did not place an os9 instruction in the 6809 instruction set. The OS-9 assembler actually recognizes this pseudo mnemonic as a special case, along with the I$Write string, and translates the above piece of code into:

```
            lda     #1
            leax    mess,pcr
            ldy     #5
            swi2
            fcb     $8A
            rts
mess        fcc     "Hello"
```

The $8A which follows the swi2 instruction is the constant representation of the I/O system call I$Write. Since the swi2 instruction calls into the OS-9 kernel, the code in the kernel looks for the byte following the swi2 instruction in the module (the $8A) and interprets that as the system call code. Using that code, OS-9 jumps to the appropriate routine in order to execute the I$Write.

Since the system call code following the swi2 instruction is a byte, in theory this would allow OS-9 to have up to 256 different system calls that can be executed on behalf of an application. Under OS-9 Level Two, this is the case; however under OS-9 Level One there are restrictions placed on exactly which codes are available. The following tables show the range of system call codes.

The idea behind *User mode* vs. *System mode* is to allow two different points of execution for the same system call, depending on whether the calling process is running in user state or system state. OS-9 controls this by maintaining two system call tables: one for user state and one for system state. When installing a system call, as we'll soon see, we can specify whether our system call should only be called from system state (hence only updating the system table) or from both user and system state (updating both the user and system tables).

An example of a system call that can be executed in both user and privileged modes is the F$Load function code (pp. 8-25 in the OS-9 Level Two Technical Reference manual; pp. 106 in the OS-9 Level One Technical Information manual). Since F$Load can be called from a user state process as well as from a driver or other module running in system state, OS-9 installs this system call in both the user and system tables. On the other hand, a privileged mode system call such as F$APProc (Level Two: pp. 8-74; Level One: pp. 141) can only be called from system state and therefore a user state process attempting to call it will receive an error.

Notice that in both OS-9 Level One and OS-9 Level Two, codes $80-$8F are reserved for I/O system call codes. When the OS-9 kernel receives one of these codes, it passes the code along to IOMan for processing. I/O system calls cannot be added since they are under the control of IOMan.

Installing a new system call involves selecting a free system call code, determining whether the call will be accessible from both user/system state or from system state only, and building a table in assembly language that will be used to install the system call. Interestingly enough, the method of installing a system call is by calling a system call! It's called F$SSvc and is documented in your respective OS-9 Technical manual.

Installing a System Call in OS-9 Level Two

The source code in Listing 1 is the system extension module, os9p3.a, which contains the code to install the system call, as well as the system call code itself. Incidentally, this is virtually the same code that is found in the OS-9 Level Two Technical Reference Manual on pp. 2-2 to 2-4. I've eliminated the comments for brevity since they are already in your manual, as well as changed the use directive. Instead of including /dd/defs/os9defs, I include /dd/defs/os9defs.l2. The rea-

son for this is that I do compiling of both OS-9 Level One and OS-9 Level Two modules on my CoCo 3 development system. Since the OS-9 definitions are different for each operating system, I have renamed their respective os9defs files with an extension indicating which operating system they belong to. Even if you just develop for one operating system or the other, I strongly suggest following the same naming convention; it will save you headaches in the long run.

This module, called OS9p3, installs the F$SAYHI system call. A process making this call can either pass a pointer to a string of up to 40 bytes (carriage return terminated) in register X, or set X to 0, in which case the system call will print a default message. In either case, the message goes to the calling process' standard error path. While not very useful, this system call is a good example of how to write a system extension.

The asm program is used to assemble this source code file. Notice that the entry point for the module is the label Cold, where Y is set to the address of the service table, SvcTbl. Each entry in this table contains three bytes. The first is the system call code that we have selected from a range that Microware says is safe to use for new system calls, and the remaining two are the address of the first instruction of the system call. The table, which can contain any number of entries, is terminated by byte $80. After setting Y to the address of the service table, a system call to F$SSvc is made, which takes the table pointed to by Y and installs the system calls.

The code for the F$SAYHI system call in listing 1 is for OS-9 Level Two only. It determines whether or not a valid string pointer has been passed in register X. If indeed the caller has passed a valid pointer, then control is routed to the label SayHi6 where Y is loaded with the maximum byte count and the process descriptor of the calling process is used to obtain the system path number of the process' standard error in register A. The separation of user and system state paths is an important concept to understand; however, we will discuss it in detail in another article. For now, let's continue analyzing the code.

The I$WritLn system call then prints the string at register X to the caller's standard error path. If on the other hand, register X contains a zero, then room is made on the caller's stack for the default message, which is then copied into the caller's address space using the F$Move system call. The moving of the default message from the system address space to the caller's address space is necessary due to the separation of a process' address space in OS-9 Level Two.

Once the module has been compiled, it should be included in your OS-9 Level Two bootfile. Reboot with the new bootfile, and the OS9p2 module will find OS9p3 then jump into the execution offset (the Cold label in this case). This will install the F$SAYHI system call and make it available for programs immediately.

Installing a System Call in OS-9 Level One

Listing 2 is similar to the code in Listing 1, except that the code to move the default message from system space to the caller's address space has been removed. Also, the code to install the system call has changed, and the module type is not of type Systm, but instead of type Prgrm. This is due to the lack of separation of address space in Level One, which makes writing system extension modules much easier than in Level Two.

The common address space between the system and all processes in OS-9 Level One also makes the F$SSvc system call available from user state as well as from system state. Unlike OS-9 Level Two, where the system extension module must be placed in the bootfile, installing a system extension in OS-9 Level One takes a different approach. Placing a module called OS9p3 in an OS-9 Level One bootfile will NOT cause the system extension to be called because there are no provisions for that in the kernel. Instead, system extensions are installed by creating a module of type Prog that contains both code to install the system call and the system call itself. Installing the system call entails executing the module from the command line.

Besides the sayhi.a source in listing 2, another example of the this is the Printerr command that comes with OS-9 Level One. This is a program that actually installs a newer version of the F$PErr system call. To install the new system call, you simply run Printerr from the command line. It then installs the call and exits. There is an advantage to OS-9 Level One's approach to installing system calls: it can be done at run-time without making a new bootfile and rebooting the system. However, additional care must be taken not to unlink the Printerr module from memory. Why? Because the code for the replacement F$PErr call is in that module, and if the module is unlinked, the memory it occupied is made available subsequent reallocation and at some point, a system crash will ensue.

Exercising Our New System Call

Listing 3 is a small assembly language program, tsayhi, which calls the F$SAYHI routine. It will work fine under both OS-9 Level One and Level Two. If you fork the tsayhi program without any parameters, then the F$SAYHI system call is called with register X set to $0000, which will cause the system call to print the default message. Otherwise, you can pass a message on the command line as a parameter and up to 40 of the message's characters will be printed to the standard error path.

Summary

Extension modules give us an effective way of altering the behavior of OS-9 by allowing us to add a new system call or modify the behavior of an existing one. Writing extension modules requires an extremely good understanding of the internals of OS-9. The particulars of writing a system extension vary under OS-9 Level One and Level Two primarily due to the differences between memory addressing.

Next time, we will explore the internals of OS-9 Level One and OS-9 Level Two by looking at their respective os9defs files. There is a world of information about both operating systems in these text files. To get a head start, begin to study them. They may seem a bit archaic for now, but we will soon break new ground in this fascinating study of OS-9 internals. If you have any questions about this article, or ideas for future topics, please contact me.

Boisy G. Pitre is a Principal Software Engineer for Microware Systems Corporation and owner of Planet 9 Systems (http://www.pitretech.com/planet9). He has been an avid OS-9 user and advocate for over 12 years and still uses his CoCo 3 with OS-9 Level Two. He can be reached via e-mail at: boisy@acadian.net.

Listing 1 starts on page 11.

(continued from page 11)

```
            os9   F$SSvc
            bcs   Exit
            clrb
Exit        os9   F$Exit

F$SAYHI     equ   $25

SvcTbl      equ   *
            fcb   F$SAYHI
            fdb   SayHi-*-2
            fcb   $80

* Entry point to F$SAYHI system call
SayHi       ldx   R$X,u
            bne   SayHi6
            leax  Hello,pcr
SayHi6      ldy   #40
            ldu   D.Proc
            lda   P$PATH+2,u
            os9   I$WritLn
            rts

Hello       fcc   "Hello there user."
            fcb   $0D

            emod

OS9End      equ   *
            end
```

Listing 3

```
Type        set   Prgrm+Objct
Revs        set   ReEnt+1
      mod
OS9End,OS9Name,Type,Revs,start,256
OS9Name     fcs   "TSayHi"

            fcb   1                 edition number

            ifp1
            use   /dd/defs/os9defs
            endc

level       equ   2
            opt   -c
            opt   f

F$SAYHI     equ   $25

* routine cold
start       equ   *
            lda   ,x
            cmpa  #$0D
            bne   SayHi
            ldx   #$0000
SayHi       os9   F$SAYHI
            bcs   error
            clrb
error       os9   F$Exit

            emod

OS9End      equ   *
            end
```

Recursive Procedures in Languages That Do Not Support Recursion

by Aaron Banerjee

A recursive procedure is a procedure which invokes itself. While this is not a particularly complicated idea, recursion is one of the more awkward concepts in computer programming. Why would anyone write a recursive procedure? How is recursion used to solve problems? After all, anything a recursive procedure can do can also be done with non-recursive techniques.

Although recursive procedures are never the only way to solve a problem, in certain cases, they may be the most intuitive or convenient. Consider the Fibonacci series, which is a series of numbers which appear in certain biological phenomena[1].

1,1,2,3,5,8,13,...

The first two terms are 1. Every term after that is the sum of the preceding two terms (1+1=2, 1+2=3, 3+5=8, etc.). This can be described with a recursive procedure:

f(n+2) = f(n) + f(n+1)
where f(1)=1, f(2)=1

This can easily be implemented in C code:

```
int fibonacci(int n)
{ int q;

   if ((n==1)||(n==2)) q=1
   else q=fibonacci(n-1)+fibonacci(n-2);
   return (q);
}
```

The corresponding code in TRS-80 Color Computer BASIC (shown below) would not run properly because (among other things), changing the value of N in one "iteration" will change it in every other. The C language (in versions which support recursion) "remember" the state of the previous iteration.

```
10 REM  Calculate Fibonacci Numbers (doesn't work)
15 REM  Input N, output Q
20 IF N=1 OR N=2 THEN Q=1 ELSE N=N-1:GOSUB
10:Z=Q:N=N-1:GOSUB 10:Q=Z+Q
30 RETURN
```

In order to make this work recursively, we would need a more complex BASIC program. In the simple case of Fibonacci numbers:

f(n+2)=f(n+1)+f(n) : f(0)=1, f(1)=1

it is better to solve the difference equation instead of worrying about recursion. Besides, on small computers that operate at 0.9 MHZ, you don't want to do any iterations you don't have to. The following program prints out the first 10 Fibonacci numbers in closed form:

```
5 CLS
10 REM  FIBONACCI NUMBERS = FNF(N)
15 F1=(1+SQR(5))/2:F2=(1-SQR(5))/2
20 D = F2 - F1
25 A=(F2 - 1)/D:B=(1-F1)/D
30 DEF FNF(N)=INT(.5+A*F1^N + B*F2^N)
35 FOR N=0 TO 10:PRINT N,FNF(N):NEXT N
```

The above program works rapidly on a TRS-80 Color Computer and for large numbers, even faster than a 486 25 MHz system running the recursive technique. This isn't a fair comparison because we didn't do a recursive approach. Instead, we simply found a totally different approach to solve the same problem. In general, there is a way around recursion, but sometimes recursion can be the simplest way to solve a problem. Consider a more challenging example: One of the most famous recursive problems is the "eight queens" problem. The object is to find each and every way one can place eight queens on a chess board without having any one queen being able to attack any other. For those unfamiliar with chess, a chess board is an 8x8 matrix. A queen can move any number of spaces vertically, horizontally, or diagonally. A queen may attack any other piece which is on a square that the queen can move to. One obvious solution to this is to do an exhaustive search. Set up an array and keep placing queens on it until none of them can attack each other. Placing queens at random gives you 64*63*62*61*60*59*58*57 = 1.78 x 10^{14} possibilities, which would probably take years on the Coco. There is a recursive algorithm which is known to solve the eight queens problem[2].

"The problem can be solved with a recursive procedure having two parameters—some representation of the chess board and an integer in the range zero to eight. When the procedure is called with the integer parameter having some value n, it is assumed that an acceptable placement has been found for queens in the first n columns of the chessboard. Thus if n is eight, the procedure should just print out the arrangement of queens on the chessboard and return. If n is less than eight, the program should try to place a queen in each of the eight squares of column n+1 in turn, and check whether a queen on that square is in the same row, left-to-right ascending diagonal, or left-to-right descending diagonal as a queen in some previous column. If this check reveals no conflicts with any queens in previous columns, the procedure should call itself recursively with the chessboard representation having a queen on that square and the integer parameter equal to n+1, to try to fill in the remaining columns."

From our exercise above with Fibonacci numbers, we see that the Coco does not lend itself particularly well to recursive procedures and furthermore, this time, however, there isn't a simple way out. In order to solve this problem recursively, first we have to solve the problem of the Coco not being able to recurse. One of the main reasons why the attempt at recursion in BASIC for the Fibonacci series (above) did not work is that all of the variables and states are global. The C example, on the other hand, stores the state before invoking the next iteration. An obvious solution would be to emulate a stack so that the program could "remember where it left off" before recursing, and restoring the state upon return (except for the quantity which was to be changed). Actually, some Coco functions do "remember where they left off". Try this short program and run it.

```
5 PRINT MEM
10 GOTO 5
```

Not very impressive. It simply prints your available memory (20805 in my case) and repeats it. Now try this one:

```
5 PRINT MEM
10 GOSUB 5
```

Run it and watch as your computer unceremoniously runs out of memory and gives you an ?OM ERROR. Every time you GOSUB, the computer has to use some memory to remember where you GOSUB'ed from so that when it encounters a RETURN, it will return to the correct place. If you had pressed <BREAK> before running out of memory, you would be able to type RETURN several times (as many as the program had cycled) without getting an ?RG ERROR. The trick to recursion on the Coco is to save everything which is needed instead of just return addresses.

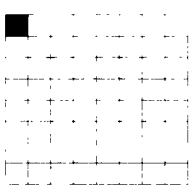
The following is a solution to the eight queens problem written in Coco BASIC.

```
10 CLS
20 GOSUB 600
30 CLEAR
40 DIM YS(10),NS(10),B(8,8)
50 SP=0:BC=0
60 REM
70 REM
80 REM          MAIN PROGRAM
90 REM
100 N=0:GOSUB 380
110 PRINT"YEE HA!"
120 END
130 REM  CLEAR THE BOARD
140 FOR IC=1 TO 8:FOR JC=1 TO 8
150 B(IC,JC)=0
160 NEXT JC,IC
170 RETURN
180 REM      CHECK FUNCTION
190 REM  INPUT N,Y
200 REM  OUTPUT C=0 (FALSE) 1=TRUE
210 REM
220 C=1
230 FOR IC=1 TO N-1:FOR JC=1 TO 8
240 IF B(IC,JC)=0 THEN 270
250 IF IC=N OR JC=Y THEN C=0
260 IF ABS(IC-N)=ABS(JC-Y) THEN C=0
270 NEXT JC,IC
280 RETURN
290 REM     PUSH Y AND N
300 REM
310 SP=SP+1:NS(SP)=N:YS(SP)=Y
320 RETURN
330 REM     PULL Y AN N
340 REM
350 N=NS(SP):Y=YS(SP):SP=SP-1
360 RETURN
370 REM
380 REM      PLACE FUNCTION (RECURSIVE)
390 REM
400 IF N>=8 THEN GOSUB 490:GOTO 460
410 N=N+1
420 FOR Y=1 TO 8
430 GOSUB 180
440 IF C THEN B(N,Y)=1:GOSUB 290:GOSUB
380
450 NEXT Y
```

```
460 GOSUB 330
470 B(N,Y)=0
480 RETURN
490 REM    DRAW BOARD
500 CLS:BC=BC+1:PRINT"BOARD "BC:PRINT:
PRINT
510 PRINTTAB(10)"EIGHT QUEENS":PRINT
520 PRINT"  " STRING$(26,128)
530 FOR JC=1 TO 8
540 PRINT "  "CHR$(128);
550 FOR IC=1 TO 8
560 IF B(IC,JC)=1 THEN PRINT" Q "; ELSE PRINT
"  ";
570 NEXT IC: PRINT CHR$(128): NEXT JC
580 PRINT "  "STRING$(26,128)
590 RETURN
600 REM  INTRO
610 REM
620 CLS
630 PRINT
640 PRINTTAB(9)"EIGHT QUEENS"
650 PRINTTAB(7)"BY AARON BANERJEE"
660 PRINT:PRINT
670 PRINT"THIS PROGRAM FINDS ALL POS-
SIBLE"
680 PRINT"WAYS TO PLACE 8 QUEENS ON A "
690 PRINT"CHESS BOARD WITHOUT ALLOW-
ING "
700 PRINT"ANY QUEEN TO ATTACK ANY
OTHER."
710 PRINT:PRINT"PLEASE BE PATIENT. THE
PROGRAM"
720 PRINT"IS EXTREMELY SLOW (SEVERAL "
730 PRINT"MINUTES FOR THE FIRST BOARD)"
740 PRINT
750 INPUT"PRESS ENTER TO BEGIN";A$
760 RETURN
```

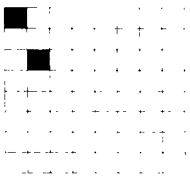
This program executes the recursive algorithm for solving the 8 queens problem. The representation of the board is the B(8,8) array. Each element has a value of 1 if there is a queen present, 0 if not. The value N is the integer whose value is from 0 to 8. To start the program, N is given a value of 0 and the subroutine is called.

Let us now consider operation of the program. As mentioned before, N is set to zero and the Place procedure (line 380) is invoked. The algorithm calls for each square in the N+1 column (column 1 in this case) to be checked. In this design, to check the N+1 column, N is incremented by 1 and a FOR Y loop is set up in line 420 to start the count. The first time the check function is called from line 430, it will be checking to see if any queen can attack square 1,1. Since there aren't any queens on the board yet, the check function will set C=1. Line 440 places a queen at the location (1,1). The algorithm states that at this point the procedure should be recursively called for the next (N+1 = 2) column. If we do that, we will be altering the value of N. In addition, we haven't completed the FOR Y loop yet, which will cause problems. At this point, the board should look like this:

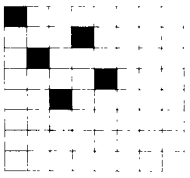


In this example, the upper left corner is taken as square 1,1. The column number (N) progresses to the right, the row number (Y) progresses downward. In line 440, we push Y and N on a simulated stack and GOSUB 380 (calls the Place function from itself).

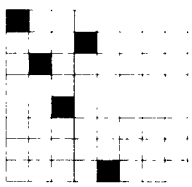
In the second iteration, N is incremented to 2 and a new FOR Y loop is started. The computer has forgotten about the old FOR Y, but this will be dealt with shortly. The new FOR Y loop looks at square 2,1 (to the right of the previous queen). This is not acceptable so it checks 2,2 (also unacceptable). Square 2,3 will check out (C=1) so a queen is placed there. The board now appears like this:



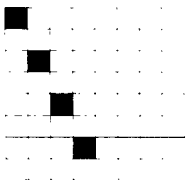
Again the procedure will push the values of Y and N (2,3) onto the stack (on top of the 1,1 pushed previously). This process will keep repeating and pushing Y and N on the stack until the following pattern is encountered:



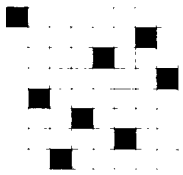
At this point the stack will have (1,1), (2,3), (3,5), and (4,2). It will push (5,4) onto the stack and GOSUB 380 again. This time, there will be no spaces in the N=6 column where a queen can be placed without being able to attack any other. The NEXT Y will be reached. The program then pulls N=5,Y=4 from the stack. Since there were no good places to place a queen in the sixth row with the position of the first 5, the queen at 5,4 is removed in line 470. Since we called Place (line 380) from line 440, the RETURN in line 480 will not return us from the subroutine, but rather back to line 450. Although we have already exited the FOR Y loop, we have reset the counter Y back to 5 and jumped back in. NEXT Y is not that smart and will not realize we've ever left. This is an unorthodox technique that is generally frowned upon because different BASIC interpreters may react differently. Computer programmers who disdain the use of GOTO would probably consider jumping in and out of FOR loops and altering the counter anything from "spaghetti code" to blasphemy. In order to get recursion in non-recursive environments, some unconventional techniques are necessary. In any event, the location of the last queen (5,4) was pulled from the stack and removed from the board. The NEXT Y in line 450 increments Y to 6 and the next square is checked. Notice that we've fooled the computer into going back to where it was in the N=5 row when it left. The computer will find another place to place a queen at (5,8) but again there will be no safe place to put a queen on the N=6 row. This is shown below:



The computer will pull the (5,8) from the stack and remove the queen, and then pull the (4,2) from the stack (N=4, Y=2) and remove that queen. When the NEXT Y in line 450 is encountered, it goes back to Y=3 in the N=4 row, effectively "going back to where it left off". It eventually finds a safe spot at Y=7:



Although the above pattern will not result in eight queens, since we have saved the values of N and Y on the stack, we will eventually be able to "get back" to earlier columns. Ultimately, a valid solution will be found. The first valid solution is shown below:



When this solution is eventually encountered by the program, N will be equal to 8. Note that in line 400, if N>=8, the program simply displays the board and returns (first pulling the last values of N and Y off of the stack). Note that the procedure does not stop there or clear the board. It will recurse back and try to find other similar solutions. Ultimately, it will find all 92 solutions to putting 8 queens on a chessboard such that none can attack another. There are a few important quirks to the 8 queens program:

1. It does not run on all BASIC interpreters. For example, GWBASIC will usually run out of memory. I'm using Extended Color Basic 1.0 with Disk Extended 1.1.
2. It is extremely slow. On a 0.9 MHz Color Computer, it takes several minutes just to find the first solution.
3. I haven't written a routine to record the solutions to the board. Once it finds the second solution, it forgets the first. This is easy to overcome by altering the "draw board" subroutine at line 490 (even something as simple as PRINT #-2 instead of PRINT would do the trick). Make sure you print the board to a tape/disk file or to the printer if you're planning on leaving the program running overnight to get all of the solutions. Don't print a CHR$(128) to a printer. It can have some interesting side effects on some printers.
4. The graphics aren't very good.

There are other famous recursive routines that could be solved on the Color Computer. One is the method of printing out the contents of a binary tree in order. The logic for programming would be the same as in the chess example. Before the GOSUB to the recursive subroutine, push needed quantities onto a stack. Immediately before RETURNing from the subroutine, pull the same quantities off of it.

Aaron Banerjee <aaron@mirror.his.com>
7620 Willow Point, Falls Church, VA 22042

[1] The ratio of one Fibonacci number over a preceding one approximates a "Golden Ratio" which occurs frequently in nature. For example the length of a leaf over it's breadth is a Golden Ratio. Even the dimensions of my credit card very closely approximate a Golden Ratio.

[2] Cohen, N.H. "ADA as a Second Language". McGraw-Hill. 1986. Exercise 9.1

Great CoCoist Interview Series

John Kowalski

By Nick Marentes

John Kowalski, currently 27, is the wizard behind many of the CoCo3's most spectacular "Demos". John has literally pushed his machine to the limits, doing what many previously believed to be impossible on a stock CoCo3. For John, it's a case of, "possible unless proven otherwise"! John has a very informative web site featuring many of his CoCo achievements. Most of his programs are shareware and are available for download via his web site. Support John's efforts and show your appreciation by sending him a shareware registration. With enough support from the CoCo community, John will continue to amaze us with his programming expertise and continue to remind us how powerful our favorite "8-bitter" really is!

Programming Achievements
Mouse Maze - 1986 - CoCo3
Twilight BBS - 1987-97 - CoCo3
Demo 1 - 1989 - CoCo3
Demo 2 - 1992 - CoCo3
Boink - 1993 - CoCo3
Digi-512 - 1993-96 - CoCo3
Twilight Terminal - 1995 - CoCo3
Williams Arcade - 1996 - SNES
CoCo Tracker - 1996 - CoCo3
Gloom - 1996 - CoCo3
Atari Collection - 1997 - SNES
Gloom2 - 1998 - CoCo3
(Note: Jeff Vavasour was the main programmer of Williams Arcade)
Web Page - http://www.axess.com/twilight/sock/
E-Mail - sock@axess.com

INTERVIEW

Q: Why are you referred to as "Sock Master"?
A: I've written programs under several pseudonyms in the past, Dave Osborne, and most notably Sock Master. The Sock Master name is still more famous than my real name, so I tend to use that one a lot. They originate from the old days of BBSing, when most users had aliases rather than real names on the BBS.

Q: Where do you live?
A: Montreal, Quebec, Canada. Montreal isn't technically accurate anymore since I moved into the suburbs a couple of years ago. I'm really in St-Hubert, but nobody's heard of St-Hubert.

Q: How and when did you become interested in computers?
A: It all started when I was still a kid. School was close enough to home to be walking distance, and there was a Radio Shack along the way. Back then, Radio Shack was a friendlier place and I used to hang out there after school or during lunch (when I wasn't at the arcade, instead). Anyway, I got along well with the people there, and they didn't mind if I stayed in the store for hours just tinkering with the TRS-80, especially since I ran errands for them now and then or made demos on the computer or something. When the TRS-80 Model III came out, I convinced my parents to get me one. Interestingly enough, when I first saw a CoCo, I didn't think much of it because of it's blurry (TV) no-lowercase 32 column display. I didn't get a CoCo 2 until years later. Lucky for me that I did get one. Machine language was less intimidating on the 6809 and I learned how to program it myself. My first machine language programs were hand coded and poked into memory from BASIC.

Q: What computers have you owned and currently own?
A: I've got a fair pile of them, though I've lost a few over the years. Most of them are not hooked up, and just sitting in the attic. I don't have my TRS-80 Model I or III anymore, but I still have a Model 4D. I also owned a VIC-20 for 3 days when they came out before returning it to the store, and got another one recently and had it for a week before giving it away to someone who actually LIKED it! I have an Atari 130XE, an Atari 520 ST, and a 1040 ST (I used to use the ST regularly). Also have a couple of Apple IIs and a Mac SE. My wife has a Mac LC475. There's a Tandy 1000, and enough spare parts to put together a few 8088, 386 and 486 PCs. I'm typing this on my K6 (Pentium clone) PC. Most importantly, I have a 64K CoCo 1, 64K CoCo 2, and about six CoCo 3s in various states of functionality and configurations. I also have a few odds and ends like a Timex/Sinclair ZX81(??) and a Seiko computer watch (with printer and keyboard!) which can run games/programs off a rompack, or typed in BASIC.

Q: What was your favorite computer and why?
A: My favorite computer is the CoCo 3. Yes, it's obsolete now, but it's one of the few computers out there that is FUN to use, easy to program, and has tons of hidden potential. Yes, I have a PC and use that a lot too, but when it comes to making computers do something specific that you want done, I use the CoCo 3. For things like that, the only real disadvantage the CoCo 3 has is that it's slower than a modern PC. The fact that it's more fun makes up for it, especially if you're making it do something that it was never envisioned as being capable of doing.

Q: What products have you developed?
A: I worked on Arcade's Greatest Hits - The Atari Collection 1 and Williams Arcade's Greatest Hits for the Super Nintendo. I've made a ton of CoCo programs, but many of those are free or shareware. (I haven't received any registrations in months! hint!hint!) The most notable ones being Twilight Terminal because of it's 20 color 640x225 graphics and proper Extended ASCII font set, and CoCo-Tracker because it finally lets the CoCo play 4 voice MOD files. I also wrote a number of BBS programs over the years. The only one still up is mine, Twilight BBS which you can call 300 to 14400 baud at (450) 926-8444. I also made a number of demos that show off some things a CoCo3 can do. I was hoping that by showing that some neat things CAN be done, maybe people would start writing programs that actually used them. So far, not many people have tried using those ideas. One of these days, I'd like to write a game that incorporates as many neat effects as possible and just wow everybody. Wouldn't it be great if there was a CoCo game that had hundreds of colors on the screen, 4 voice digitized sound, raster video effects (graphics scaling, on-the-fly palette changes, multiple layer smooth scrolling...), and maybe really fast 3D graphics?

Q: Your 'Boink' bouncing ball demo looks fantastic! How does it work?
A: The Boink bouncing ball demo is one of my more interesting programs. It doesn't specifically push the CoCo's hardware to the limits or anything, but it combines a number of hardware tricks all together in one elegant 'package'. Along with probably most CoCo users out there, I found Vaughn Cato's bouncing ball demo very interesting. The only problem I had with it, was that the background was empty. The original Amiga demo had a checkerboard background, so why couldn't the CoCo? The problem was that when you scroll the ball around the screen, you're scrolling the whole screen - including the background. Can't have a checkerboard background if it's just going to bounce along with the ball... So, I wanted to make a demo that somehow kept a steady background. Unfortunately, the only legal way to do that would be to redraw the ball over the background every time it moved. There are a couple of demos out there that do it that way, but they end up with low frame rates (choppy movement), because it takes the CPU a long time to draw the ball every frame (unless you want to have a very small ball). I came up with the idea of just putting vertical stripes in the background. This would allow the ball to move up and down without affecting the background (though, that's not a checkerboard yet), but the stripes still follow the ball when you move it left & right. To fix that, I rigged the program to generate four copies of the graphics screen in memory, each with the stripes in the background offset by a few pixels. Now I could rig the demo to 'select' which of the 4 backgrounds to use depending on the horizontal position of the ball itself. Technically, it would have required 8 copies of the screen to cover all ball positions for the width of the stripes, but I could eliminate half of them by reversing the palette settings of the background to 'fake' the other 4 screens. Well, that gives us solid stripes in the background. Where's the checkerboard? There actually isn't any

checkerboard in the background of my demo. It just looks like there is because I alternate the palette settings every 12 scan lines - color 1 is black and color 2 is white, and then 12 lines later, I switch color 1 to white and color 2 to black...and on. I decided that since I was switching palettes every few scan lines on the screen anyway, I might as well use that to my advantage. Instead of just black & white checkers in the background, I made it do a rainbow effect. If you find that it looks familiar, it's because it's actually modeled after the colors on the cover of the CoCo3 BASIC manual! At that point, I decided that it would still be too easy to guess what the trick behind the demo was, so I wanted to add something to throw people off. I wanted to put some non-moving graphics at the bottom of the screen that were more than just checkers to convince people that the background really WASN'T moving. The problem is, if you're scrolling the whole screen up, down, left & right, the bottom of the screen will scroll right along with the ball. A long time before this, I looked into the possibility of fooling the GIME chip into retriggering a graphics screen in mid-frame. Some way of forcing a new memory address to the video after it's already begun being displayed. That's how other systems did things like split screen two player games and other stuff. I gave up, I concluded that there was no way the GIME could be tricked into doing that. I came up with another way to keep the graphics at the bottom of the screen from moving. The GIME has a lot of odd graphics modes that never get used. One mode keeps repeating the same line over and over across the whole screen - essentially a 320x1 resolution. It was time to use it! Instead of making the stripes behind the ball extend all the way to top and bottom of the screen, I rigged it so that there was only ONE pixel of stripes above and below the ball. Since the demo was already synchronized with the video so that it could update the palettes at specific scan lines without causing palette glitches on the screen, it wasn't hard to add some extra code to also set the video mode at specific scan lines. What it does, it sets the resolution to 320x1 for the first 'however many pixels the ball is away from the top of the screen' scan lines, then set the resolution to 320x225 for the next 'however many pixels the ball is tall' scan lines, then switch back to 320x1 for the next 'however many pixels the ball is away from the bottom of the screen' scan lines. Once that was done, it just switched it back to 320x225 to display the last 9 scan lines that held the 'non-moving' graphics at the bottom of the screen. Oh, and while it was at it, set the horizontal scroll register back to zero so that the bottom doesn't move left & right along with the ball either. All that was left to do was to make sure that the program always kept track of how many pixels were above the ball, and how many were below, and the whole graphics screen would end up rock-stable while the ball is bouncing all over the place. What about the transparent ball shadow? That just got added in as an afterthought. I figured it wouldn't be any harder to switch palettes for four background colors than it would be to do two. The shadow under the ball is just drawn in two extra colors, one for when the shadow is over a stripe, and one for when the shadow is over black. The palette switching that creates the illusion of a checkerboard background also switches the shadow palettes to create the illusion that the shadow is darkening the checkerboard pattern. The only drawback to reserving 4 colors for the background was that it left only 12 to do the ball's palette animation rotation. That's why the ball seems to turn faster than the other demos - it's because it's cycling through less palettes. Oh yeah, that's about everything except the ball itself. I didn't want to actually draw it, because I figured anything I would draw would look more fake. So, I took a blue rubber ball I had and drew lines on it and painted checkers on it. I used that as a model, and then wrote a basic program to try to generate that ball on the screen. I still have that ball sitting in a drawer in the attic. After some experimentation, I fixed up most of the mathematical glitches that gave me all sorts of screwed up looking balls on the screen. There was one glitch I never fixed, and I decided it was 'good enough'. If you look at the top of the ball, where all the lines join together, you notice that the center point is a bit warped, or off-center. I figured it wasn't distracting enough to be too much of a problem. I just never did figure out the proper math to do a '3D' ball on the screen! If you're wondering, it took that program about an hour to run before completing the image.

Q: What motivates you to make another ground-breaking demo?
A: Time. If I had free time, I'd make another demo!

Q: You have also designed a 4Mhz accelerator for the CoCo3. Can you tell us more about this?
A: The 4Mhz accelerator came about after having a few separate ideas that ended up fitting together. I wanted to try speeding up the CoCo3, so I tried the usual stuff of changing the clock crystal - that works to an extent, but it has side effects. Boosting the clock beyond 2Mhz (normal is 1.79) would push the video rate enough for my CM-8 not to sync with video anymore and also cause my RS232 card to malfunction. After a certain number of tests with various crystals, I concluded that the CPU ITSELF could be over clocked significantly - without blowing up! But the rest of the CoCo could not. The next idea came when I looked at the 6809 cycle-by-cycle operation flow charts. There are a fair number of CPU cycles that execute without addressing memory. If I could only run those cycles faster, I could speed up the CPU without affecting any other part of the CoCo. Looking at the 6809 data sheets, I saw that the AVMA pin on the CPU always gave early warning wether the CPU would or would not be addressing memory in the next cycle. So, the plan was to build a circuit that knew when the CPU wouldn't use memory, and run 'non-memory' cycles at a faster clock. That's actually not too hard, but there's a hitch. If you run one cycle twice as fast, you still have to WAIT for the next cycle to resynchronize with the 1.79Mhz bus. You'd only get any speed increase if there were two 'non-memory' cycles in a row because you could run both within one bus cycle. The next thing I did was actually build something. I just wanted to screw around with the CPU clock and see how dangerous it was to the rest of the hardware to do so. I built a clock divider. For every two E and Q pulses being generated by the GIME, the circuit would only send ONE to the CPU. I booted it up, and PRESTO! It ran as slow as molasses. My CoCo3 just booted up at 0.45Mhz, and the fast poke switched it to a fantastic 0.89Mhz! Other than some jittery video in .45Mhz sometimes, it ran fine. I concluded that you could 'bend' CPU & memory timing quite a bit without it failing. The final idea that made the 4Mhz accelerator possible was this : Instead of 'just' running non-memory cycles at 4Mhz, I could try bending the previous and following cycle's timing to make room for a sneak (burst) cycle. This way, any single non-memory cycle could be executed BETWEEN two normal cycles without the rest of the CoCo ever knowing it happened! What it does is cut the ending phase of the 'before' cycle short, sneak in a burst cycle that doesn't address memory, and then clip the beginning of the 'after' cycle. The result meant that the CPU ran 5 different 'sizes' of cycles, depending when it's using memory and when it isn't, but it also means that it can run FLAT OUT double speed if there is at least one non-memory cycle happening every two CPU cycles. Well, it doesn't happen that there are that many non-memory cycles occurring, so it doesn't actually perform at twice the speed. On average, it runs about 40% faster than stock. Later on, I got a 6309 CPU and the clock doubler also works with that. A 6309 in native mode is already faster than a 6809, and it's faster still with the clock doubler. It's really neat seeing CoCo programs run faster than they were intended to, especially with games. The speed, though is a bit program dependent. Any program that uses few opcodes with non-memory cycles in them won't speed up much, but programs that do have a lot of those opcodes will speed up a lot.

Q: What companies did you work for?
A: I'm currently working for Digital Eclipse. They develop console video games. Before that, I did programming for a medical company, MedNet. The rest wasn't computer related, unfortunately.

Q: Can you tell us any interesting "stories" of your past development days?
A: Most of my experiments were deliberate. I had an idea that something should be possible and I tried it. There was some random hunting for effects - keep poking all sorts of numbers into all sorts of registers (especially video registers) in all sorts of 'times' and see if anything neat happened. By times, I mean things like during the horizontal or vertical video border, or at the end of a scan line or the last line of the screen, etc.. A few neat but fairly useless discoveries were that the 1987 GIME could be tricked into displaying overscan graphics (graphics WITHIN the border) and that the 1986 GIME could be tricked into displaying interlaced video (640x450). I also found out that the interrupt timing between both versions of the GIME is very different, and that's a regular annoyance that interferes with special effects. Luckily, I also found out how to detect which GIME your CoCo has so I could make it so that a program works on both of them. One interesting story is about writing the sound code the Williams Arcade's Greatest Hits. I wrote a translator that converted the original arcade sound program from 6800 to Super Nintendo (SNES) Sound CPU assembly. I made sure that it generated code that ran exactly at the same frequency because all the sounds were algorithmic (A program loop that generated numbers that got fed to a DAC - timing is everything!) and it turned out that everything ran way too slow. Why? Turned out to be a combination of a typo and a half truth in the

Nintendo manual. What they meant to write as 2.048Mhz got written as 2.48Mhz, and 2.048 itself was sort of a lie because every 2nd cycle was stolen by DMA so the CPU really ran at 1.024Mhz when the manual said it was 2.48Mhz! Lets just say it was a lot of trouble to go through all the code by hand and make it go 242% faster.

Q: Are you still developing for the CoCo and why?
A: Yep, I got into it too late to have quit while I was ahead, I guess. The reason I program the CoCo is because I enjoy it. There's a big feeling of satisfaction when you finally get your CoCo to do something really neat. It's also nice when you can program whatever it is you want to program, in any time frame you want.

Q: What are some of your favorite CoCo products of all time?
A: There are a lot of programs that I really liked, either because they were fun, useful or technically impressive. The standards have gone up since the early CoCo days, but that doesn't make some of the older programs any less impressive. Some of my favorites are Zenix, Shanghai, Gauntlet II, V-Term, CoCoMax III, VIP Library, ADOS-3, Dungeons of Daggorath, Wildcatting and Dino Wars. Dragonfire wasn't the best game in the world, but it gets an honorable mention because it managed to coax 8 REAL colors out of the PMODE3 display on a CoCo1. The 8 colors in the CoCo version weren't artifacted. The game switched between the two PMODE3 'palettes' a few times in almost every scan line of the screen. So you had all 8 colors simultaneously on the screen. The trick backfired on the CoCo3 because the GIME had different video timing, and the color changes happened around 8 pixels later than they should have and it made a messy looking screen.

Q: What is your opinion of the CoCo2 and CoCo3 hardware platform?
A: The CoCo 2 was nice in it's day, but it's seriously out of date now. It's still useful for number crunching and interfacing with the real world (measuring and/or controlling things), but as a desktop computer, it's not very useful. The CoCo 3 is pretty cool. The CPU isn't fast enough and the graphics aren't good enough, but with a little bit of work you can still do quite a lot of things with it. The only serious thing that it's lacking today is Internet ability - and even that could be overcome with some work. As far as I'm concerned, the CoCo can do ANYTHING, only slower or not as well as a new machine. It's just a shame that there aren't many programmers left. The CoCo 3 hasn't reached anywhere near it's potential yet.

Q: If you were asked by "Mr. Tandy" to create a CoCo4, what would you include?
A: Some people might think I'm crazy, but I'd keep the 6809 - or rather, the 6309 CPU. My vision of a CoCo4 is a three 6309 CPU machine, where each CPU runs at 4 (not 3.58) Mhz. With three CPUs, this CoCo4 would be slightly more than 8 times faster than a stock CoCo3. The neat thing about 3 CPUs is that you can specifically program each one to do a different task. A game could use one CPU for sound & music, another to draw all the graphics, and the last to process game data. It would be like having one processor and two completely customizable co-processors. Want a sound chip? Write it! Want a graphics processor? Write it! Want a JAVA interpreter? ..you get the idea. It's very versatile, and could do a lot of neat things while still being cheap to make. I wrote a text file describing my vision of a CoCo4, it's on my web page.

Q: Have you any "words of wisdom" to pass on to any budding CoCo programmers?
A: Learn 6809 assembly. Count CPU cycles when programming important sections of your program - the most obvious way to program something usually isn't the fastest. Find an idea that has never been properly taken advantage of on the CoCo and just do it, whether it seems impossible or not - If you think about it for a while, you might be surprised at some of the solutions you can come up with. Oh, use lots of self-modifying code, it's faster!

Classic BASIC Game Series

The Ghana Bwana Patch

By Steve Bjork

-ed.- This is the official Ghana Bwana, Pitfall II, Desert Rider & One On One patch from Steve Bjork (used by permission). Replace the old basic boot program with this new one and you will be all set. Ok, so it's not a game, but it will fix some good ones that most CoCo 3 users haven't been able to use. It also allows for a RGB monitor.

```
10 'NEW PROGRAM BOOT GHANA BWANA, PITFALL II, DESERT RIDER, ONE ON ONE .
20 'THIS PROGRAM WILL GIVE YOU FULL COLOR ON A COLOR COMPUTER 3 WHEN USING A RGB MONITOR.
30 'THIS PROGRAM WILL ALSO FIX THE BUG IN GHANA BWANA ON THE COLOR COMPUTER 3.
40 CLEAR 50,&H3FFE
50 P=PEEK(&H3FFF)
60 X=&H7FFF:POKE X,165
70 POKE X,255-PEEK(X)
80 IF PEEK(X)<>90 THEN 100
90 IF P=PEEK(&H3FFF) THEN 110
100 CLS:PRINT"64K IS NEEDED FOR THIS GAME"
110 CLEAR 500,&H7FFF
120 FOR I=0 TO 70
130 READ A$
140 POKE &H5000+I,VAL("&H"+A$)
150 NEXT I:WIDTH 32
160 DSKI$0,34,4,A$,B$:SUM=0:FOR X=1 TO
128:SUM=SUM+ASC(MID$(A$,X,1))+ASC(MID$(B$,X,1)):NEXT X
170 IF SUM=20423 THEN INPUT"REMOVE WRITE TAB FROM DISK AND PRESS ENTER. ";Q$:MID$(A$,106,1)=CHR$(254):MID$(A$,122,1)=CHR$(254):DSKO$0,34,4,A$,B$: GOTO 160
180 READ S,A,B,C,D:IF S=0 THEN 280 ELSE IF S<>SUM THEN 180
190 POKE &HFFD8,0:PRINT "ARE YOU USING A RGB MONITOR? (Y/N)"
200 Q$=INKEY$:IF Q$<>"Y" AND Q$<>"N" THEN 200
210 IF Q$="N" THEN 280
220 PALETTE 0,A:PALETTE 4,A
230 PALETTE 1,B:PALETTE 5,B
240 PALETTE 2,C:PALETTE 6,C
250 PALETTE 3,D:PALETTE 7,D
260 PRINT "PLEASE HOLD THE CLEAR KEY TILL THE GAME IS BOOTED."
270 IF INKEY$<>CHR$(12) THEN 270
280 EXEC &H5000
290 DATA 86,22,8E,26,00,8D,0D
300 DATA FC,26,00,10,83,4F,53
310 DATA 26,03,7E,26,02,39,34
320 DATA 20,10,BE,C0,06,A7,22
330 DATA 86,02,A7,A4,6F,21,6F
340 DATA 23,6C,23,AF,24,10,BE
350 DATA C0,06,A6,23,81,13,27
360 DATA 12,AD,9F,C0,04,4D,27
370 DATA 06,6C,23,6C,24,20,E9
380 DATA 7F,FF,40,35,A0,4F,20
390 DATA F8
400 DATA 22438,63,0,9,36
410 DATA 20651,63,36,9,0
420 DATA 22631,0,9,36,63
430 DATA 65280,63,36,9,0
440 DATA 0,0,0,0,0
```

Classic BASIC Game Series

Word Wrap

A neat little word game for up to four players.

By Andrew Jackson

This is a pretty neat and fun game for up to four people. The instructions are at the end of the listing and incorporated into the program for display. The text will have to be reformatted to fit the screen when displayed, as it is difficult to count spaces in a printed listing. It is not necessary to type the instructions, of course. You may simply want to make a photo copy of the last part of the listing and keep it with the program. Happy word hunting!

```
10 R=RND(-TIMER):GOTO180
20 POKEPEEK(136)*256+PEEK(137),
96:TM=TM-1:IFTM=0THEN50ELSEIF
SF=1 THEN SOUND240,1
30 A$=INKEY$:IFA$=""THENPOKEPEEK
(136)*256+PEEK(137),255:GOTO20
40 IFA$=CHR$(13)ORA$>CHR$(31)
THENRETURNELSEIFA$<>CHR$(8)
THEN20ELSEIFSA$=""THEN20ELSE
PRINTA$;:SA$=LEFT$(SA$,LEN(SA$)-1)
:GOTO20
50 PRINT@511,STRING$(31,8);"time is
up";:SOUND1,10:RETURN
60 JP=1024:C=96:P4=PEEK(136)*256+
PEEK(137)-1024:IF(JF AND1)=1THEN
JS=0:B=1ELSEJS=2:B=2
70 J=JOYSTK(0):J=INT(JOYSTK(JS+1)/
(63/N2)):K=INT(JOYSTK(JS)/(63/N2)):
POKEJP,C:TM=TM-8:IFTM=0THEN50
80 JP=J*32+K+SP:C=PEEK(JP):POKE
JP,25 5:IF(PEEK(65280)ANDB)THEN70
ELSEPOKEJP,C:SA$=RIGHT$(STR$ (J*
10+K),2):GOSUB160:PRINTSA$;:RETURN
90 FORJ=1TO11*N2+1:J=INSTR(J,F$,
"1"):IFJ=0THEN110
100 IFMID$(F$,H(J1)+1,1)="1"THENJ=J-
1:K=H(J):GOTO140ELSENEXTJ
110 J=RND(11*N2+1)1:IFBL$(J)=""OR
MID$(F$,J+1,1)="1"THEN110
120 IFMID$(F$,H(J)+1,1)="1"THENK=
H(J):GOTO140
130 K=RND(11*N2+1)1:IFBL$(K)=""OR
J=KOR(MID$(F$,K+1,1)="1"ANDK<>H(J))
THEN130
140 SA$=RIGHT$(STR$(J),2):GOSUB
160:PRINTSA$;:RETURN
150 IFA$<"0"ORA$>RIGHT$(STR$(N2),
1) ORLEN(SA$)=2THENRETURNELSE
PRINTA$;:SA$=SA$+A$:RETURN
160 IFVAL(SA$)>9THENRETURNELSE
MID$(SA$,1,1)="0":RETURN
170 FORDL=1TO2000:NEXT:RETURN
180 CLS:CLEAR10000:TM=0:TA=175:
DIMBL$(99),H(99),L(50):N$="0123456789":
W$="WORDWRAP":M=1:D=33:POKE1184,
143: POKE1189,143
190 PRINT@P,MID$(W$,M,1);:C=C+1:M
=M+1:IFC=140THEN205ELSEIFPEEK
(P+D+1024)<>96THEND=(D=33)*-1+
(D=1)*33+(D=-33)+(D=-1)*-33
200 P=P+D:IFM=LEN(W$)+1THENM=1:
GOTO190ELSE190
205 PRINT@224,"COPYRIGHT JULY
1990"," ANDREW JACKSON",," 6825
SUENA DR",," AUSTIN, TX 78741",,,,,
"NEED INSTRUCTIONS? (Y/N)";:GOSUB
20: CLS:IFA$="Y"ORA$="y"THENGOSUB
1000
210 SA$="":JF=1:WF=0:CLS:PRINTW$,,"
GRID SIZES:","(1) 5 BY 5",,"(2) 6 BY 6",,
"(3) 7 BY 7",,"(4) 8 BY 8",,"(5) 9 BY 9",,"(6)
10 BY 10"," (K) USE KEYBOARD"," (N)
NO WORDWRAP",,,"ENTER OPTIONS:";
220 GOSUB30:IFA$="K"THENJF=0:
POKE1252,11:GOTO220ELSEIFA$="N"
THENWF=1:POKE1284,14:GOTO220
ELSEIFA$="\"THEN
210ELSEIFA$<"0"ORA$>"6"THEN220
230 N2=VAL(A$)+3:GC=INT((N2+1)^2):
SL=INT(GC/2):F=GC-INT(GC2)*2:PRINT
A$,,"UP TO FOUR CAN PLAY. HOW
MANY?";
240 GOSUB30:IFA$="\"THEN210ELSEIF
A$<"1"ORA$>"4"THEN240
250 PRINTA$;:NP=VAL(A$):W=NP:FOR
X=1TONP:PS(X)=0:SA$="":PRINT:PRINT"
PLAYER"X"ENTER NAME:";
260 GOSUB30:IFA$="\"THEN210ELSE
IFA$>CHR$(13)THENIFLEN(SA$)<10
THENPRINTA$;:SA$=SA$+A$:GOTO260
ELSE260ELSEIFSA$=""THEN260ELSE
NA$(X)=SA$:IFSA$="COMPUTER"THEN
W=W-1
270 JN(X)=X:NEXT:GOTO290
280 IFTC<GC THEN540ELSEFORX=0
TONP1:NA$(X)=NA$(X+1):PS(X)=PS(X+1):
JN(X)=JN(X+1):NEXTX:NA$(X)=NA$(0):PS(X)=
PS(0):JN(X)=JN(0)
290 FORX=0TO11*N2:BL$(X)="":NEXT:
FOR X=0TOSL:L(X)=1:NEXT:TC=F:SF=1:
P=231:M=1:C=0:T=0:D=1:SP=1255:WW$="":
IFW<2ORWFTHENWW$=STRING$(GC,32):
WF=1:GOTO390
300 FORX=1TONP:WF(X)=0:IFNA$(X)=
"COMPUTER"THENPW$(X)="":GOTO360
310 SA$="":CLS:PRINTNA$(X);" ENTER
UP TO";INT(GCW):PRINT "CHARAC-
TERS FOR YOUR WORDWRAP"
320 GOSUB20:IF(A$=" "OR(A$>"@ "AND
A$<"[")ANDLEN(SA$)<INT(GC/W)THEN
PRINTA$;:SA$=SA$+A$:GOTO320ELSEIF
A$=CHR$(13)THENCLS:Y=1ELSE320
330 Y=INSTR(Y,SA$," "):IFY=0THEN350
ELSEIFY=1THENSA$=RIGHT$(SA$,LEN
(SA$)-1):GOTO330ELSEIFMID$(SA$,Y
+1,1)=" "THENSA$=LEFT$(SA$,Y)+
RIGHT$ (SA$,LEN(SA$)-Y-1):Y=Y-1EL
SEIFY=LEN(SA$)THENSA$=LEFT$(SA$,LEN
(SA$)-1)
340 Y=Y+1:GOTO330
350 IFSA$=""THENX=X1ELSEPW$
(X)=SA$
360 NEXTX:FORX=1TONP
370 R=RND(NP):IFWF(R)THEN370
ELSEWW$=WW$+PW$(R):WF(R)=1:NEXT
380 IFLEN(WW$)<GCTHENWW$=
WW$+WW$:GOTO380ELSEWW$=LEFT$
(WW$,GC)
390 CLS:SF=0:F$=STRING$(SL,48):
C=RND(38)+127
400 T=T+.5:D=(D=1)*-32+(D=32)+(D=-
1)*32+(D=-32)*-1:IFT+M>GC THEN T=
GC-M+1
410 FORY=1TOT:P=P+D:PRINT@P,
CHR$ (ASC(MID$(WW$,M,1))+C);
420 M=M+1:NEXTY:IFM<GC THEN400
430 SP=SP-32:IFPEEK(SP-32)<>96
THEN430
440 SP=SP-1:IFPEEK(SP-1)<>96THEN
440ELSEP3=SP-1024:PRINT@P3-32,
LEFT$(N$,N2+1)
450 SB=0:FORX=0TON2:POKESP-2,X+
112:IFF ANDX=N2 THENN2=N2-1
460 FORY=SPTOSP+N2:BL$(SB)=
CHR$(PEEK(Y)-C):POKEY,99
470 R=RND(SL)-1:N=VAL(MID$(F$,R+1,
1)):IFN=2THEN470
480 N=N+1:MID$(F$,R+1,1)=RIGHT$
(STR$(N),1):IFL(R)=-1THENL(R)=SB
ELSEH(SB)=L(R):H(L(R))=SB
490 R=R-(R>35)*62:BL$(SB)=BL$(SB)+
CHR$(96+R):SB=SB+1
500 NEXTY:SB=10*X+10:SP=SP+32:
NEXTX:IFF THENN2=N2+1:PRINT@Y-
1024,CHR$(PEEK(Y)-C);
510 SP=P3+1024:F$=STRING$ (11*N2
+1,48):PRINT@79,"player";TAB(25)"score":
FOR X=1 TO NP:PRINT@143+(X-1)*64,
LEFT$(NA$(X),8)
520 PRINT@153+(X-1)*64,PS(X):IFSF
THENRETURNELSENEXTX
530 FORX=1TONP
540 PRINT@511,STRING$(95,8);:TM=
TA: SF=0:I FTC=GC THEN790
550 SA$="":PRINT@416:PRINT@416,
NA$(X)" ENTER 1ST CHOICE:";
560 IFNA$(X)="COMPUTER"THENGO-
SUB 90:GOTO590
570 IFJF THENJF=JN(X):TM=INT(TA/1.6)
*8:GOSUB60:IFTM=0THEN 690ELSE IF
INKEY$="\"THEN
800ELSE590
580 GOSUB30:IFTM=0THEN690 ELSE
IFA$="\"THEN800ELSEIFA$>CHR$(13)
THENGOSUB150:GOTO580
590 IFLEN(SA$)<2THEN580ELSEV=VAL
(SA$):IFBL$(V)=""THEN550
600 MID$(F$,V+1,1)="1":P=VAL(LEFT$
(SA$,1))*32+VAL(RIGHT$(SA$,1))+P3:PRINT
@P,RIGHT$(BL$(V),1);
610 SA$="":PRINT@448:PRINT@448,
```

```
"ENTER SECOND CHOICE:";
620 IFNA$(X)="COMPUTER"THENSA$=
RIGHT$(STR$(K),2):GOSUB160:PRINTSA$;:
GOTO650
630 IFJF THENGOSUB60:IFTM=0THEN
680ELSE650
640 GOSUB30:IFTM=0THEN680ELSEIF
A$>CHR$(13)THENGOSUB150:GOTO640
650 IFLEN(SA$)<2THEN640ELSEV2=
VAL(SA$):IFBL$(V2)=""ORV=V2 THEN
610
660 MID$(F$,V2+1,1)="1":P2=VAL
(LEFT$(SA$,1))*32+VAL(RIGHT$(SA$,1))+
P3:PRINT@P2,RIGHT$(BL$(V2),1);
670 IFRIGHT$(BL$(V),1)=RIGHT$(BL$
(V2),1)THEN700ELSEGOSUB170:PRINT
@P2,"#";
680 PRINT@P,"#";
690 PS(X)=PS(X)-2:SF=1:GOSUB520:
NEXTX:GOTO530
700 PS(X)=PS(X)+5:SF=1:GOSUB520:
PRINT@482,"right";:SOUND210,1:GOSUB
170:TC=TC+2:SF=0
710 PRINT@P,LEFT$(BL$(V),1);:PRINT
@P2,LEFT$(BL$(V2),1);:BL$(V)="":BL$(V2)=
"":MID$(F$,V+1,1)="0":MID$(F$,V2+1,1)="0"
720 IFNA$(X)="COMPUTER"ORWF=1
THEN540ELSEFORY=1TONP:IFPW$(Y)=""
ORX=Y THENELSESF=1
730 NEXT:IFSF THENSF=0:TM=TA:
PRINT@480,"WANT TO GUESS WORD
WRAP? (Y/N):";ELSE540
740 GOSUB30:IFTM=0ORA$="N"THEN
540ELSEIFA$<>"Y"THEN740ELSEC=INT
(GC/W)*2:SA$="":PRINT@511,STRING$
(95,8)"ENTER GUESS "NA$(X)":";
750 TM=53:C=C-1:GOSUB30:IFTM=0OR
C<0ORA$= CHR$(13)THENELSEIFLEN
(SA$)=INT(GC/W)THEN
750ELSEPRINTA$;:SA$=SA$+A$:GOTO750
760 FORY=1TONP:IFPW$(Y)=""ORPW$
(Y)<>SA$ORY=XTHEN780ELSEPW$(Y)
="":PS(X)=PS(X)+10:SF=1:GOSUB520
770 PRINT@416,STRING$(32,42);"
THAT'S CORRECT "NA$(X)", YOU","
SOLVED "NA$(Y)"'S WORDWRAP";:FOR
DL=1TO40:SOUNDRND(10)+200,1:NEXT
780 NEXTY:IFSF=0THENPS(X)=PS(X)-
5:SF=1:GOSUB520:PRINT@416,"wrong
guess";:SOUND1,20:GOTO540ELSE540
790 PRINT@384,"END OF WORD
WRAP:"
800 SA$="":TM=0:PRINT@416,"(R)
ESUME GAME","(N)EW GAME","PRESS
ANY OTHER KEY TO QUIT.","WHICH
ONE?";
810 GOSUB30:PRINT@511,STRING$
(95,8);:ONINSTR("RN",A$)GOTO280,210:
END:GOTO540
1000 PRINT" This game is a concentra-
tion type game with a different twist. In
Wordwrap the words actually wrap clock-
wise around and around each other.
Wordwrap requires a good memory, the
ability to type or peck at least ";
1010 PRINT"four words per minute, and
a little luck. It can be played by one to four
people. The computer can play one or
more of the positions. At the beginning of
a new game a menu will print six options,
";:GO SUB2000
1030 PRINT"for the six available grid
sizes and options K and N. If option K
(press K to used keyboard to play game)
and/or N(press N to play game without a
wordwrap) is used they must be entered
before one of the ";
1040 PRINT"grid size options is entered.
At the next prompts the total number of
players in the game should be entered
and each player should enter their name
(enter COMPUTER for any position the
computer will play). While ";:GOSUB2000
1060 PRINT"the menu is still displayed,
to go back to the first prompt to correct any
mistakes, press SHIFT CLEAR at the
same time (if this option is used all previ-
ous menu items must be reentered). If
option N was not used and ";
1070 PRINT"there are at least two real
players, each player will be asked to enter
a wordwrap. A sound should also be
heard, if not the volume should be turned
up until it can be heard. This sound will
prevent anyone from ";:GOSUB2000
1090 PRINT"pressing BREAK or RESET
to find out what any of the other player's
wordwrap is, and then restarting the pro-
gram. All players not entering a wordwrap
cannot look at the monitor or the keyboard.
As long as there ";
1100 PRINT"is an audible sound every-
thing should be alright. Use only space
s and uppercase alpha characters for
wordwrap entries. Wordwraps can be on
any subject, movies, sports, books, au-
thors, states, random selections, etc.
";:GOSUB2000
1120 PRINT" The order of play for each
player will be to the right of the game grid.
At the start of each new game if joysticks
will be used and there is more than two
players in the game, players one and three
must share the ";
1130 PRINT"right joystick, and players two
and four must share the left joystick. To
allow players time to get into position, at
the start of each game the game can be
paused by pressing SHIFT @. Before a
player enters their ";:GOSUB2000
1150 PRINT"first choice, the game can
also be paused at any point during play
by pressing SHIFT CLEAR and if joysticks
are used press the joy button in use. Each
grid position has a match, and the object
of the ";
1160 PRINT"game is to find it. Each player
will have ten seconds and one chance to
find two matching grid positions. If a match
is not found or time runs out the player
loses two points and game control
passes to the next ";:GOSUB2000
1180 PRINT"player. If a match is found
the player scores five points, gets another
turn, two wordwrap characters will be
printed at the two matching grid positions,
and if there is any unsolved wordwraps
he or she will have a ";
1190 PRINT"chance to guess one of the
otherplayer's wordwrap. To guess press
Y, otherwise press N (N will be assumed
if N or Y is not pressed within ten sec-
onds). No points is loss if a guess is not
made. If a wordwrap guess is
";:GOSUB2000
1210 PRINT"made, at least one charac-
ter must be entered every three seconds
and the total number of keystrokes can-
not exceed two times the maximum char-
acters a wordwrap could be (backspace
and other non-printing characters ";
1220 PRINT"will not be considered as
characters or counted as keystrokes). If
time runs out or too many keystrokes is
entered, the computer will check anything
that has been entered for a wordwrap
match. Ten ";:GOSUB2000
1240 PRINT"points is scored for a correct
wordwrap guess. Five points is loss for a
incorrect guess. When playing from the
keyboard, to enter a grid position, first en-
ter it's row number and then it's column
";
1250 PRINT"number, press ENTER when
done. For joystick use move the cursor to
the chosen grid position and press the
joy button. At the end of each game
options (R)ESUME GAME and (N)EW
GAME will be displayed. If R is";:GOSUB
2000
1270 PRINT"pressed the order of play will
change, but the names, scores, and the
original joy positions will be carried over
to the next game. If N is press all scores
and names will be cleared from memory.
At the main menu all ";
1280 PRINT"prompted information must
be reentered. If any other key is pressed
to end the game prematurely, type CONT
and press ENTER to resume game. The
amount of time each player has to make
grid entries can be ";:GOSUB2000
1300 PRINT"adjusted by changing vari-
able TA in line 180. The first TM in line
750 can be adjusted to allow more of less
time for wordwrap guesses. A number
less than zero will allow player to play at
their own pace. Use ";
1310 PRINT"whole numbers only for TA
and TM. ":GOSUB2000:RETURN
2000 PRINT@480,"HIT ANY KEY TO
CONTINUE READING";
2010 IFINKEY$=""THEN2010ELSECLS:
RETURN
```

THE GLENSIDE COLOR COMPUTER CLUB
Presents -
THE EIGHTH ANNUAL "LAST" CHICAGO COCOFEST!

MAY 1st & 2nd, 1999 (SAT. 10-5; SUN. 10-3:30)
AT THE SAME GREAT LOCATION AS LAST YEAR
345 W. RIVER ROAD (A CITY BLOCK FROM I-90 & IL-31),
ELGIN, ILLINOIS
GENERAL ADMISSION: $10.00, WHOLE SHOW
******* CHILDREN 10 AND UNDER - FREE *******

OVERNIGHT ROOM RATE: $65.00 (PLUS 10% TAX)
CALL 1-847-695-5000 FOR RESERVATIONS.
BE SURE TO ASK FOR THE "GLENSIDE" OR "COCOFEST!" RATE.
!!!!! RESERVE YOUR ROOM EARLY — THESE ROOMS WILL BE RELEASED FOR REGULAR RESERVATIONS ON APRIL 14th, 1999, AND WILL NOT, !NOT! BE AVAILABLE TO THE FEST ATTENDEES >>>>>YOU MUST REGISTER UNDER "COCOFEST!" TO GET THIS RATE<<<<<

WHY? - A. TO PROVIDE VENDOR SUPPORT TO THE COCO COMMUNITY
B. TO PROVIDE COMMUNITY SUPPORT TO THE COCO VENDORS
C. TO PROVIDE EDUCATIONAL SUPPORT TO NEW USERS.
D. TO HAVE AN OUTRAGEOUSLY GOOD TIME!!!!!

FOR FURTHER INFORMATION, GENERAL OR EXHIBITOR, CONTACT:
TONY PODRAZA, VP, SPCL EVNTS, GCCCI or BRIAN GOERS, PRESIDENT, GCCCI
847-428-3576, VOICE - 708-754-4921, VOICE
847-428-0436, BBS - TONYPODRAZA@JUNO.COM

Ron Bull Invites You To Attend

PennFest '99

The Third Pennsylvania CoCoFest!

August 21st and 22nd
8am - 4pm Saturday and 8am - 3pm Sunday
Holiday Inn (near the airport)
1406 Beers School Rd.
Coraopolis, PA 15106
(Call 1-800-333-4835 for reservations)

Many vendors have already registered to attend!
Buy software, hardware, meet new and old friends,
learn new tricks, hear the guest speakers,
and most of all, *have fun!*

Admission is $5 per person per day or
$15 for a "Family Pass" good for both days.

For more information contact:
Ron Bull
Phone: 717-834-4314
ronbull@aol.com
www.stg.net/bullsbarn

Coming Attractions!

Coming soon in "the world of 68' micros" -

Jokes and Quotes - a new section

GWBASIC to DECB - part 3

Adventures In Assembly - part 4

"Great CoCoist" Interview Series - #2

Where Is OS9 Today?

All About CoCozilla!

Dumping Carts To Disk

Neurocomputing for the CoCo

...and much more!

ADVERTISER'S INDEX

What are you waiting for?

Get your friends to subscribe to the only magazine that still supports the Tandy Color Computer... *"the world of 68' micros"*!
The more people who want the support, the longer it will be here!