

For your  
TANDY  
Color Computer

PNG K5.95  
NZ \$3.95

\$4.50

AUSTRALIAN

# RAINBOW

July, 1986

Registered by Australian Post — Publication No. QBG 4009 **No. 61**



*A Symphony of Sound*

TANDY ELECTRONICS DEALER. (No 9320)

# TANDY COMPUTERS & ACCESSORIES

best  
prices!

FREE DELIVERY THROUGHOUT AUSTRALIA

90 DAYS WARRANTY

bankcard  
welcome here

DISK DRIVE 0 FOR CoCo  
40 TRACK DSDD  
DISK ECB 1.4  
AUTO LINE NUMBERING, SUPPORTS FLEX & OS9.  
6MS Access time  
Inc Controller and Manual \$599

## BAYNE & TREMBATH

3 Boneo Rd., Rosebud, Victoria 3940  
Ph: (059) 86 8288. A/H: (059) 85 4947

Bankcard &  
Cheque Orders  
accepted

## DATALOG

SOFTWARE SPECIALISTS

Presenting  
the DTX 2001 Monitor.  
\$460.00 + Tax (20%)

- \* Sharp brilliant colour
- \* Green and Amber modes
- \* NSTC input
- \* Sound

Perfect Brand Disks.

5.25" (Box of 10)

SSDD ..... Call!!

DSDD ..... \$27.00

DSQD ..... \$63.00

DSHD (1.2MB) ..... \$90.00

3.5" (Box of 5)

SSDD ..... \$28.50

DSDD ..... \$45.00

Unit 2, 278 Newmarket Rd.,  
Wilston, Qld. 4051.

07-300-1978

07-300-5463

## Hardware/Software Specialists

For All Your CoCo Needs

**AUTO ANSWER \$399.00**

### INFO CENTRE

THE FIRST BULLETIN BOARD SYSTEM  
for Tandy's computers

(02) 344 9511 — 300 BPS (24 Hours)

(02) 344 9600 — 1200/75 BPS

(After Hours Only)

### SPECIAL!

Avtek Mini Modem + Cable + CoCo  
Tex Program — the total Viatel System —  
\$279.00

We also have the largest range of Software for  
OS-9 and Flex operating systems.

## PARIS RADIO ELECTRONICS

161 Bunnerong Rd., Kingsford, N.S.W. 2032

(02) 344 9111



CoCoConf'86 .....	P 3
Letters .....	P 5
Reviews	
CD Editor .....	P 6
Hall Of The King .....	P 6
Super Duper Utilities .....	P 7
Typing Tutor .....	P 7
Algebra .....	P 8
SCRDMPIO's .....	P 8
OS-9 Goodies .....	P 9
XSCREEN .....	P 10
RAMDisk .....	P 10
BLEEPS, BLOOPS, BELLS and WHISTLES	
..... by William Mitchell	P 11
CoCo SYNTHESIZER .....	by Martin and Jeremy Spiller P 13
MUSIC+ .....	by Bob Ludlum P 20
LANGUAGE IDIOMS .....	by Steve Blyn P 22
ROBOTS .....	by Michael Plog P 23
CASTLE OF DOOM .....	by Scott Halfman P 24
Correction ..	CoCo Zone ..... P 26
COLOUR CALC .....	by C. Bartlett P 27
GREAT TRANSFORMATION .....	by Marty Goodman P 30
COMPARE .....	by E. Pozzi and K Paterson P 35
BARDEN'S BUFFER .....	by William Barden P 37
Forth Forum .....	by John Poxon P 44
Virtually Done .....	by John Redmond P 45
16 Bit! .....	by Jerome Siappy P 47
Basic 09 .....	by Jack Fricker P 48
New Text Formatter OS9 .....	by Dale Puckett P 49

## Computer Hut Software

Music Box ..... Tape only \$34.95  
 Learn to Read Music ..... Tape or Disk \$33.95  
 CoCoTex ..... Tape or Disk \$79.95  
 Use your CoCo to contact Goldlink on Viatel.

Special!!

64K upgrades for SOME CoCo's ..... \$40.00  
 Phone or message us on Viatel for details.

T1000 Games:

    Sea Search ..... \$49.95  
 Shenanigans ..... \$49.95

See us on Goldlink this month!

We accept :-  
 \* BANKCARD  
 \* MASTERCARD  
 \* VISACARD

VIATEL NUMBER  
 778622200

Computer Hut Software  
 21, Williams Street.  
 Bowen, Qld. 4805.  
 Phone (077)86-2220

ALL ORDERS SHIPPED SAME DAY

Some people have asked me, "Who are you and what is your role in the office?". I can't blame them for asking. Everyone has their own story of how they got into something or another. So I'll do my best to explain to you how I got into this mess!

First I'll describe myself. My name is Alexander (I prefer "Alex.") Hartmann, born 23/6/1968 in a town called Mannheim, West Germany. I'm about 179cm, fair complexion, brown hair and eyes.

Now, how did I get myself involved in all this?

Sometime in September 1984 Michael and Rainer (Horn) drove up and asked me, "Did you know that there's this computer meeting on tonight?". I said I didn't and decided to come along.

The meeting place then was a Fishing Club hall. There I met Kevin and Graham. They were nice enough people alright.

So nice were they that they had ~~suckered~~ asked Michael to make tapes for the magazine!

For Michael it had paid off well. He would get the odd free copy of the magazine for making a million odd tapes of Rainbow and CoCoOz. I thought (at the time) that was pretty neat-o. So I found myself going to 'the office' to see if I could do some odd job to get a magazine and maybe a few programs.

The house. They said "you can't miss it! It's an old house with a death trap as stairs".

As I fell through the stairs, I decided I must be in the right place.

I was briefly introduced to everyone there; there was a Sonya, a Martha, and an Annette. Sonya, I believe, was the typist. They were running behind in finishing the magazine (as usual) and she was in the middle of some typing.

Graham sat me down in front of a CoCo and gave me some lengthy letters. He said one word: "TYPE!". So I did. According to them, I was the fastest two fingered typist they'd ever seen.

In those days all I did was the typing.

Nowadays I do some or all of the following:

Typing, tape-making, banking, programming, hardware and software repairs, advisor, editor, coffee making, magazine-wrapper, 'receptionist', mail-opener, tape-checker, hardware upgrades.

Not bad, huh?

Anyway, I'll tell you about this month's edition.

One of my hobbies is making music. And that's what this edition is all about.

The CoCo has two ways the average user can make music without setting up lengthy machine language programs.

The first is SOUND. This command is alright, but the biggest grudge I have with SOUND is that you can't do much with it. Sure, you can play music, but it has only one voice and anyway, there are better ways to make music (if you have Extended Color Basic). That better way is called PLAY.

PLAY lets you enter notes, octaves, volume, and pauses all in one statement. Again it only has one voice, although you could play the music so fast that you make it sound like it has two voices. But that won't do if you're really into making and playing music.

If you want to delve into machine language, there's either something like Tandy's Speech Pak or Super Voice, which can play 3 voices at the same time.

Alright for the person who's happy with three voices.

After the three voice generation, enter the four voice generation. They are somewhat more popular,

because most people will think, "Hey, four's better than one!".

There are many of these programs out from both commercial and magazine sources. Some commercial ones from the top of my head are Musica and Musica 2.

If you want to go all the way, there's Symphony 12. Now this is BIG. It plays 12 voices at the same time! Hardware requirements are a keyboard from the same firm, or an interface, so that you can play from your home organ.

Then there's Synther 77. That lets you fool around with the ADSR (Attack Delay Sustain Release), vibrato, volume, and a few other things. Also, don't forget the stereo pak. One places it in the side of the CoCo and you connect the other side of the cable to your stereo.

On to CoCoConf'86.

Our very own Annette even at this moment, is roaming the streets to chase up the best in accommodation and prices! Her article appears this month. Use it to help you choose a place to stay.

Above all, remember that the Gold Coast is quite long - if you book into lodgings at Surfers Paradise, you'll have a three quarter hour trip to CoCoConf'86 and back each day. Try to get something in Tweed Heads or Coolangatta.

You've probably heard it all before, but why not come to CoCoConf'86? It'll be fun, interesting, educational, and different. You'll meet interesting personalities like Martha, Graham, Annette, Jim Rogers, me, and a few other people you've never heard of before.

How do you book? Easy. Fill out the form and send it with a cheque to Goldsoft as quickly as possible. We'll look forward to seeing you on the 30th!

For those of you who have been reading the past few editions of "PRINT#-2", you've probably noticed that we've had this competition going on.

It's the GAMES COMPETITION!!

I've been sitting at this desk opening the mail and what do I get? That's right! No games! Lots of utilities for the Utilities Competition, but no games! So how about it? Send something in and who knows - your's might be the winner!!



# COCOCONF'86

As most of you are by now no doubt aware, CoCoConf'86 is being held at the end of next month on the Gold Coast in Queensland.

We already have a good number of bookings for the conference - many more than at this time last year, so it all seems set to be a top weekend. One look at the list of speakers at the Tutorials confirms this!

## CoCoConf'86. Tutorials

Basic BASIC .....	Johanna Vagg.
Advanced BASIC .....	Mike Turk & Alex Hartmann.
FORTH .....	John Redmond & John Poxon.
OS-9 .....	Graeme Nichols, Ron Wright, & Jack Fricker.
68000 .....	Ron Wright, Jerome Slappy, & Jackie Cockinos.
MS DOS .....	Brian Dougan, Barry Cawley, Paul Fulloon.
Education .....	Ross Eldridge, Bob Horne, & Bob Delbourgo.
Games .....	Michael Horne, Andrew White, Nicholas Merantes, & Tony Evans.
Viatel .....	Ron Wright.
The Future .....	Mike Turk & Ken Allen (Tandy).
MC 10 Computing .....	Jim Rogers & friends.

## Proposed Weekend TimeTable.

Sat AM	Sat PM	Sun AM	Sun PM
Basic BASIC (J Vagg)	Advanced BASIC (A Hartmann)	Hardware (G Fiala)	Games
Advanced BASIC (M Turk)	Viatel (R Wright)	FORTH (Redmond et al)	68000 Computers
OS-9 (J Fricker)	OS-9 (G Nichols)	OS-9 (R Wright)	
MS DOS (B Dougan)	MS DOS (B Cawley)	MS DOS (P Fulloon)	The Future (M Turk & K Allen)
Education (R Eldridge)	Education (B Horne)	Education (B Delbourgo)	
	MC 10 (J Rogers)	MC 10 (J Rogers)	

In addition to the items shown in the tables, your admission to CoCoConf also pays for your meal on the Saturday night at which the presentations for the various awards will be made.

(Last year we had so much food even Tino Delbourgo got stopped - this year the banquet will be bigger!)

During the year we have had an extensive series of contests running, including our Games Contest and the Utilities Contest. In addition to these contests, prizes will be awarded in a number of other categories including OS-9, MS DOS, and our major award, the Greg Wilson Award for Services to the Computing Community. (Nominations for this award from the Users' Groups are still open.)

On the Sunday, in addition to the Tutorials, a number of our advertisers will have goods on sale. I understand there will be some real bargains!

The other reason to be at CoCoConf this year relates to an event of interest to most of our readers, about which I can not be specific, but which I know is close to your hearts!

Should the object in question arrive on time (& currently it is), CoCoConf will be its first public showing!

I want to also thank Tandy Australia Pty Ltd for the generous offer of a number of prizes - over \$1300 worth in fact!

The prizes include a Disk Drive, Computer, and a number of peripherals.

These prizes are in addition to Bayne and Trembath's offer of a complete EARS package (Speech Recognition Unit), Blaxland Computer Centre's donation of a 68000 Class Computer, Paris Radio's Donation of CoCoTex programs for Viatel, and The Computer Hut's donation of an extensive selection of software for both the CoCo and the MS DOS computers.

Like some of the previously mentioned tutorial leaders, some of the other advertisers don't know that they are donating prizes either (eg. GT Computing) - but I've embarrassed enough people for one issue, so I'll keep the embarrassing of advertisers till next month!

Following last year's article on travel to the Gold Coast, many readers will know that I am biased towards the train. This year the journey is quicker and the rolling stock is more modern.

Train is the safest, most relaxing mode of travel. You see a lot of the country, you get to move about, and you don't get thrown about like you do in a bus.

The train takes you as far as Murwillumbah, where you have to get the Greyhound bus, which will take you the 30 Km to Tweed Heads.

Seagulls RLC is on Gollan Drive in Tweed Heads, and this road has just been made part of the Pacific Highway. There are a number of motels on this road, and these, as well as hotels and other accommodation

will be the subject of an article in next month's magazine.

The Gold Coast Airport is also quite close to Tweed Heads, so it is feasible to fly in and get a Taxi to a nearby motel.

If you decide to come up by road, then you should consider the joys (there are none - except price) of bus travel, and then decide to drive.

At night, if you are an experienced driver, if you get a clear road and if you can handle continuous winding road at speed, you can do Sydney - Tweed in 8 - 9 hours.

Any change to the above circumstances usually results in the trip being more like 12 hours - day time travel is especially s-l-o-w!

12 hours on the road at one time is about the limit, so why not stop off on the way up.

Some top places to spend a night or a day include Port Macquarie, Nambucca Heads, Valla Beach, Hungry Head, Coffs, and Iluka.

If you have a little more time, also see Timber Town at Wauchope, the amazing Dorrigo Mountains and their new - but not yet open - steam railway, a night sky on the Tyringham to Grafton road (dirt but go anyway), Red Rock, and Pottsville.

The villages of Pottsville, Red Rock, Hungry Head and Valla Beach will not appeal if you do not enjoy miles of deserted uncluttered natural beach.

If you are coming from the north, then Tweed Heads is about 90 minutes down the new highway from Brisbane. There is still no train to the coast - one is being put back in (!), but Skennars, Greyhound, VIP and virtually every other bus company in the country pass through the place most days!

Travel from the west is accelerated by making the turn at Aratula on the Cunningham Hwy and following

the signs which take you through Beaudesert, Canungra and Nerang.

Travel time from Toowoomba is about the same whether you come via the Cunningham, or if you go through Brisbane.

Finally, if you travel at least part of the journey on the New England Hwy, you will not be sorry.

This highway is fast and much easier to drive than the Pacific. Unfortunately, to the Coast, it is also a little longer (as opposed to Sydney - Brisbane, for which it is shorter).

The offset is the beautiful scenery, especially if you can afford the time to take some journeys off the highway.

The waterfalls of the Armidale region are ALL recommended, especially Dangar, Wollomombi, Gara, Ebor, and Australia's tallest, Marengo.

If you get the impression that I think the New England region is special - you are dead right. That area from the New England Highway to the coast is one of THE spots in Australia to see!

We're excited so many have indicated they will come up this year; we're certain we'll all have a very useful and enjoyable time; and we're especially impressed by the line up of tutorials and tutorial leaders that will be presented this year.

CoCoConf is a resource for the whole Tandy community, so whether you are a user, a hobbyist, a Tandy employee or dealer, a teacher, or you just have a thirst for knowledge about computers, CoCoConf is a great, different way, to spend a weekend.

Graham.

# COCOCONF '86

**WHAT'S HAPPENING:-** Tutorials on Advanced BASIC, Basic BASIC, Educational use of computers, OS9, MS DOS/GW BASIC, FORTH, The CoCoConnection HARDWARE mods include:- high K upgrades (128,256,512,1mb) AND THAT'S JUST SATURDAY!! Saturday night we have our dinner and prize session. (this is included in your registration fee) SUNDAY continues with MORE tutorials plus the opportunity to browse/buy the large range of software and hardware available for the CoCo and T1000. There will be lots of bargains!

**SPEAK UP!:-** Now is your chance to suggest your ideas for any tutorials we may not have mentioned. (participants only).

**LOCATION:-**  
SEAGULLS RUGBY LEAGUE CLUB  
TWEED HEADS.

**DATE:-** Sat 30th & Sun 31st August 1986.

## REGISTER NOW!!

We can only accept a limited number of people this year. DON'T MISS OUT! on a top weekend of FUN, FRIENDSHIP and LEARNING.

Name: .....

Address: .....

Phone: .....

No. People attending: .....

\$39.95 per person/1st family member

\$20.00 per additional family member

\$9.95 dep. balance by 15/8/86

Cost includes:- tutorials, dinner Sat. night, morning and afternoon tea.

Tutorials likely to attend: .....

.....

Please find enclosed:

chq/money order/bankcard/visa/mastercard

Card No. ....

Signature: .....

# LETTERS

Dear Graham,

I have October Rainbow which gives upgrade information from 16K to 64K. But does this upgrade also Extend Color Basic?

G. Harrip  
Port Arlington. Vic.

G.

No, you need a separate ECB chip which Tandy or your local software supplier will be happy to sell to you.

Graham.

\*

Dear Graham,

I hope that you can help me in the program, "Brotan the Blue" (December 1985 Rainbow). As I am new at the computer game it took me 2.5 hours to type the program in for my children.

After finishing and running the program the title screen ran alright up to line 8 and then I could get no more to run - it printed FC ERROR IN 2011.

I have tried taking out the POKES from lines 12 and 2011 but it made no difference, and as far as I can see the POKES should make no difference because we are using a TRS-80 color computer 64K.

I checked my typing of line 2011 and found no errors. However the first line of line 2011 reads in part "T8V3102BAGAB;" -- which I think should probably read "T8V3102BAGA;B;" -- however, I also changed this just to make sure that this was not the fault and it made no difference.

I think there may be an error in one of the numbers printed or programmed or maybe it is a simple thing I'm doing wrong.

I hope someone can help so that I do not waste the effort I put into the program.

Awaiting your reply,  
S. Stride.

P.S. I bought the January '86 issue to see if there were any corrections, but nothing was listed.

Dear S.

There are many reasons why you could have got a FC ERROR IN 2011. The best answer I can give you is to re-type the line printed below.

2011

```
POKE65494,0:PLAY"T8;V31;02;B;A;G;A;B;P1;
B;P1;B;B;B;B;B;P1;B;A;G;A;B;P1;B;P1;B;B;
B;B;B;P1;A;P1;A;P1;B;P1;A;P1;G;G;G;G;G;G;
G;G":POKE65495,0
```

The first POKE in the line (POKE65494,0) slows the computer down to normal speed, and the second POKE (POKE65495,0) speeds the computer up again.

Please note that the "0" in the line (in the part of "T8;V31;02;B;" is NOT a zero.)

Anyway, hope that it works (it WILL work) and happy computing!

Alex (for Graham).

\*

Dear Graham,

Here is a little tip I just discovered to turn a ML program into an auto execute program.

Using Edtasm!, insert a line

"ORG \$182 XXXX

where XXXX equals the start address of your ML program.

Then another ORG for the program itself, and put 2 lines at the beginning:

"LDA #\$39 sta \$182.

If the stack will interfere with the program, then just move it to a different location with a

"LDS XXX.

Don't forget the stack will move down fromXXX, so leave enough room for it!

Gordon Thurston

\*

Dear Graham,

In the January '86 issue of Australian Rainbow, you have a hardware project called CoCo Conversation (by Larry Landwehr).

Larry said that he used a terminal package that he got from The Color Computer Magazine (terminal by Richard Campbell). I could not find this magazine. Can you provide details?

How do I hook up the Tandy Armatron to the CoCo?

Jorge Echegaray.  
Leichhardt. NSW.

Jorge,

The magazine you require is no longer available, but any terminal package will probably work anyway.

The Tandy Armatron is not as easy to hook up as you would at first think.

The Wagga Wagga group I believe have done it, so perhaps you should contact them.

Graham.

\*

Dear Graham,

Being interested in education and in programs that can be used by groups of children I was anxious to try Dean Hodgson's "Treasure Island" page 6 of July's issue of Australian Rainbow. Unfortunately I can't get it to run properly even though have carefully examined the listing many times. Lines 96 and 133 have errors. Line 96 reads "IF", something has obviously been omitted. Line 133 on the other hand is not critical.

When I run the program I begin at the first location but am unable to move to any other location. Because I couldn't move I tried using the TRON command. Following the path of the program for all possible commands gives the expected result except for the directions NORTH, EAST, and WEST (these are the directions available to move from the first location), which all give YOU NEED THE RAFT AND PADDLE TO GET TO THE SOUTH ISLAND as their result (line 97).

I waited for the September issue to see if there were any corrections for this program. As there were none I am writing to you in the hope that you or one of the magazine's readers has the solution to this problem.

On an entirely different matter - I know that the CoCo is an excellent machine but it irks me to read magazines such as "Your Computer", "Australian Personal Computer", "Classroom Computing" (the last magazine for primary school

teachers) and see many advertisements and programs for Apple, Commodore, Microbee, etc. but barely a mention of anything to do with CoCo. What can be done to redress this situation?

In Victoria the Education Department does not have CoCo as one of its recommended systems nor does it mention it in Department publications.

Somehow we (CoCo owners) are going to have to do something so that generalist computer magazines and government departments give our computers the representation they deserve. If any readers have strategies or ideas they feel could be useful they should put them into effect. It would be nice to see CoCo mentioned as often as machines like the Apple and Commodore 64.

Keep up the great work you have done since the untimely demise of Greg.

Ian Pengelly  
KILSYTH, VIC.

Ian,

I placed the Treasure Island program in the magazine believing that it was only me who couldn't make it work. Dean is always very careful with his programs and instructions - and as I hate adventures more than any other type of program, you can possibly see how easy it was for me to make too many assumptions!

We had word from Dean that his copy works, so we returned the copy he sent us, and he is currently working on the problem.

As soon as possible, we'll get a revised version in the magazine, and this time I'll get one of the Adventure Desperates around here to try it first!

Many changes are currently taking place to the thinking of several of the Education departments in relation to the type of computers they have in their schools.

In Victoria we did some work recently which I think proved the viability of using CoCo in the classroom, and more will be heard in the new year.

In several other states, there are people looking for the first time in years at CoCo, and drawing the conclusions we drew some time ago - that CoCo is an ideal classroom computer because of

- \* its price;
  - \* the price and availability of software;
  - \* the backup;
  - \* its ability to handle more than just the traditional classroom programs;
  - \* its durability - both physical and in terms of its stay in the market place.
- Don't worry about the other magazines - they'll catch up in a year or so!

Graham.



### CD Editor Provides Invisible Convenience

— Chuck Wozniak

Using Extended BASIC's line EDIT function to make changes in a block of program lines can, at times, be frustrating. I usually end up listing all the lines in the section of interest and then noting those needing changes or corrections. Each line is then edited one by one. Until now, there has been no easy way of working on programs one screen at a time. *CD Editor* from C & D Computer Products provides some relief from the frustration of program editing. This utility program provides many of the features of a full-screen editor without using one extra byte of memory in a 64K CoCo.

*CD Editor* requires a 64K CoCo with at least one disk drive and should work on all versions of JDOS or Disk BASIC. The program comes on disk in the form of a short BASIC program that copies the BASIC and Extended BASIC ROMs into the upper 32K of RAM. It then checks for the type of Disk BASIC that is being used and loads in new machine code from disk to the upper RAM locations containing the original line editor. The editor fits in the same amount of space that was taken up by the original EDIT function. If desired, BASIC's standard OK prompt can be changed to anything you want, up to five characters, such as READY. The loader program then clears the screen and erases itself. The copy that I originally received failed to load properly with JDOS BASIC 1.21. A phone call to C & D Computer Products isolated the problem. A short time later I got a new disk that loaded and ran properly with JDOS BASIC 1.21 and Disk BASIC 1.1.

The program is transparent to BASIC and permits BASIC programs to run as if it did not exist. The cursor becomes a blinking black square that turns into a blinking white square when the editor is active. Typing in the command EDIT turns the program on and off.

*CD Editor* works on any displayed portion of a program that has just been typed in or listed. A movable cursor determines where on the screen any changes are being made. Cursor movement is controlled by the four arrow keys. (The JDOS version uses the shifted up- and down-arrows for vertical motion.)

To delete characters, place the cursor on the first character to be deleted and press the shifted left arrow once for each character that is to be deleted. To add characters, place the cursor at the point of insertion, add blank spaces with the shifted right-arrow key, and then type the new data over the blank spaces. Typing at the current cursor position causes the new text to overwrite the existing text.

None of the changes are actually entered until the cursor is moved to the end of the line being edited and the ENTER key is pressed. Pressing ENTER in the middle of the line causes the last portion of the line to be lost. I had a tendency to forget this and often pressed ENTER immediately after making changes, and not at the end of the line.

The editor also allows two or more program lines to be combined into one. Program lines may also be duplicated by changing the line number of the line that is to be duplicated. I found this last feature quite handy in writing

programs that use many lines of nearly identical code. I just duplicated the lines as many times as required and then went back and made the unique changes to each line.

Another program on the disk lets you save the modified BASIC and Extended BASIC into EPROMs to make the screen editor and new prompt a permanent part of the CoCo. I could not try this because I do not own an EPROM programmer. However, I did try the EPROMs loaned to me by C & D Computer Products and had no problems.

The disk comes with a three-page instruction sheet which covers loading and using the program. A copy of the instruction manual is on the disk in the form of a BASIC program. The disk also contains a catalog of other programs from C & D. None of the programs are copy protected so backups can be made. The programs, however, are copyrighted.

### Hall of the King Challenges Avid Adventurers

— Barbara Combes

If you like Adventure games, and enjoy programs that show how far Color Computer programming has advanced in recent years, you should experience *Hall of the King* to see how good it gets.

Available from Prickly-Pear Software, *Hall of the King* is one of the best Adventure programs I have experienced to date. I wish I could report my victory but thus far I have been unable to solve the game. *Hall of the King* is a challenge for even the most avid Adventure player.

The opening credits are impressive and make you feel as though you're watching a show on television because the special effects are special. Next, you receive an in-depth background on the situation you're becoming involved in. You can review the scenario a page at a time at your own speed. The authors have taken time to research the topic while remaining imaginative so you're primed when the Adventure begins.

*Hall of the King* consists of two disks, and gives you a lot more playing time that you might need when an Adventure is so complex and challenging. Although there are two disks, only one disk drive is required, but you need to have 64K.

The graphics are many and well-done. *Hall of the King* is 100 percent high resolution graphics in detailed color. I enjoyed wandering through the *Hall of the King* admiring the programmer who spent so much time polishing all the fine details.

The response time to commands is almost immediate, except for a brief wait between commands while new graphics are drawn, which I didn't mind at all. There are SAVE and LOAD commands that make it possible for you to resume where you left off between games with ease. The packaging of the program is a vinyl container making it handy to grab and load. The documentation is thorough and well-done.

Good Adventures like *Hall of the King* keep the Color Computer's future bright.



## Super Duper Utilities Packs a Punch

— Jerry Semones

Utilities are some of the most popular pursuits of the CoCo hobbyist. Human nature drives us to seek new ways of doing things quicker and easier. Our computers provide the opportunity. The new offering from Microcom Software definitely does some new things quicker. *Super Duper Utilities* is supplied on an unprotected disk and is written for a 32K ECB system. The author, Kishore M. Santwani of *500 Pokes Peeks & Execs* fame, has done a good job. He has used his expertise to make CoCo do some pretty handy tasks, which I have listed here:

**40K Disk Basic** — A utility that gives 64K disk users 40K instead of the usual 32K of memory. Remember, Disk BASIC needs 2K for its own use, so there is only 38K to work with. Two versions are available, one for Disk BASIC 1.0 and one for version 1.1.

**Alphadir** — This utility reads the disk directory, then sorts and rewrites it in alphabetical order. This is handy for locating program names as your disk library grows larger.

**Basic Search** — Enables you to find all BASIC lines where a specified string is located. Very handy to find all those high speed pokes, as an example.

**Banner Creator** — Lets you create a large banner with seven-inch letters. Baud rate is selectable from the menu, ranging from 600 to 9600.

**Disk Encryption** — Provides password protection for BASIC programs on disk and keeps unauthorized people from accessing them.

**EZ Disk Master** — Lets you copy, kill and rename disk files, and to determine the starting, ending and executing addresses for ML programs. You can also run and execute programs directly from the menu.

**Function Keys** — Allows you to program any numeric keys (0-9) with strings of up to 250 characters each. This can be a very helpful feature during programming. For example, you could press a key and automatically insert "print #-2" in the BASIC line.

**Graphics Zoom** — This utility is very impressive and easy to use. A menu allows you to look at the picture in memory and to select the area to be magnified four times. You can then modify the magnified area using arrow keys to move the cursor and the space bar for turning the cursor on or off and for modifying the picture.

**Large Screen Dump** — This program dumps PMODE 3 or 4 screen images to your DMP printer. The printout runs sideways and is twice the size of the graphics screen.

**List/Dir Pause** — I really liked this one! By pressing SHIFT/CLEAR, you can make the list pause in full screens instead of flashing by. A second SHIFT/CLEAR returns to the normal mode.

**Mailing List** — A handy mailing list right where you need it. You can delete or modify the records and sort by ZIP code. It prints to either screen or printer.

**Program Packer** — Removes all spaces and REM statements from BASIC programs. Reduces the memory requirements of BASIC programs.

**Super Input/Line Input** — This is a very useful utility and one which most programmers will love. It modifies the keyboard input routine to allow editing without having to

access the EDIT command each time. Load it in and edit directly with a combination of arrow and CLEAR keys.

**Disk Zapper** — This utility allows you to change the data on the disk and recover most of the data in case of a crashed disk.

## Super Tutor — A Typing Tutor for Young Children

— Gabriel Weaver

*Super Tutor* is designed to teach letter and number recognition to children ages 2 to 6. With the aid of parents, *Super Tutor* can be expanded to teach spelling.

The program arrived on disk. Actually there are four versions of *Super Tutor* on the disk. *Super Tutor* is divided into three learning levels. Each learning level has its own program. In addition, there is a main program that runs all three levels. The main program allows you to quickly switch between learning levels.

Level one teaches letter and number recognition. Each time a letter or number key is pressed the letter or number is drawn on the screen in large block form. The letters and numbers are drawn on a black background and colored white. Up to five rows of eight characters each can be displayed at one time. When a key is pressed the parent should tell the child the name of the letter or number. After a letter or number is displayed a short melody is played, which is usually a couple of notes designed to get the child's attention.

Level two teaches the child to recognize a letter or number on the keyboard. A random character is displayed and the child presses the appropriate key to cause the character to be displayed again. Nothing happens to the display until the proper character is pressed. When the child gives the proper response, the character is echoed to the screen and a short tune is played. In level two you can select training on letters or numbers only, or both letters and numbers.

Level three can be used to teach spelling. At the start of level three, enter the largest word length to be displayed, which can be from one to eight letters. In level three, words are displayed on the screen. The child must press each letter of the word in the proper order. When a correct letter is pressed the letter is displayed. When a word is entered properly an ear-catching melody is played. The parent must work along with the program to teach the child word pronunciation and spelling.

The *Super Tutor* package includes two pages of operating instructions. The instructions are straightforward and easy to understand. Included with the operating instructions is information on modifying and adding words to level three. The *Super Tutor* programs are written in BASIC. Words used in level three are located in lines 7000 and above. Up to 250 words can easily be placed in the data dictionary. *Super Tutor* comes with 50 words in the dictionary. You must know how DATA statements are written to modify or add words to the dictionary.

*Super Tutor* is easy to operate and performs exactly as described in the instructions. The author's telephone number is included in the instructions. Parents need to participate in the training in order for *Super Tutor* to work effectively. If you are looking for a program to teach young children the alphabet, numbers and early vocabulary, *Super Tutor* may fit the bill.

## Algebra Simplifies and Solves Equations

— John McCormick

*Algebra*, to the best of my knowledge, is the only program that solves equations as *equations*, rather than numerically, although it sometimes says zero is one solution to a particular equation, even while it continues to complete the solution algebraically.

It is somewhat difficult to describe just what it does but a few examples may help make it clear.

If you enter  $(X^2-6X)^2-2(X^2-6X)=35$ , *Algebra* displays  $X^4-12X^3+34X^2+12X-35$ . Enter  $3-X^2=2X^2+1$ , and you get  $3X^2-2=0$ .

$5X-2X^2=2$  produces  $-2X^2+5X-2$ ; ready to substitute into the standard binomial solution (very hard to write in a recognizable form here, but which goes like this:  $-B$  plus or minus the square root of  $B$  squared, minus  $4AC$ , all divided by  $2A$ ; a formula that stirs memories).

Here is an equation generated by the computer (the previous ones came from a math text):  $(F+A-(C-E)-B)*E*A=00$ ; one solution is  $A=0$  and the general solution is  $A=B-E-F+C$ .

Here is one last equation, this one also generated by the computer, which I did not verify:  $C/(C+F*B/(F+E-C)/(C+D))=0$  and the solution is  $C=0$  and  $C=E+F$ . I spent about 10 minutes on this one and gave up!

Briefly put, *Algebra* simplifies or solves certain equations for any specified variable.

The only changes the user must make to his input equation is to specify every number as a letter, e.g.,  $23X-3Y$  must be written as  $AX-BY$  because the program treats everything, numbers included, as string variables.

If you don't change all numbers to letters, you often get incorrect-appearing solutions because the answer is written as "ABX" or something similar and, if 'A' and 'B' are left as numbers, you could get something like "12X" when the correct answer is "2X" (1 times 2X).

By specifying the numbers as letters, it is always obvious that the numbers are to be multiplied in the final answer.

This is not made clear in the documentation and led to my initial conclusion that there was something wrong with the program (there wasn't).

When starting the program you select three different speeds which turn out to be our old friends, the CoCo speed pokes. These pokes don't work with all CoCos but will work with most and really speed up the operation of this program.

On normal speed I could always solve the equation faster than the computer, and I even kept up with it at high speed, but I am certain my accuracy would suffer if I solved several equations in a row, a problem the computer doesn't have.

The program's author is currently translating this program into machine language, which should greatly speed up execution, although it is now faster than many people who are inexperienced with this sort of problem.

The new version should be sent to the purchasers of the BASIC version when completed.

After selecting the speed the computer can handle, you either enter an equation to solve and then specify which variable to solve for, or you have the computer generate random equations and solve them as a demonstration.

Don't run the auto equation generator on the fastest speed. If you do, you won't be able to tell what is happening.

If you get "stuck" in the fastest speed (where the display is garbage during calculations), press the BREAK key and rerun the program. Even though the screen is scrambled, this restarts the program without using the Reset button. When running your own equation in the fastest speed, the program stops at the end of a solution.

I found no bugs in this program and am impressed with the idea behind it.

I have seen better documentation but I can't remember ever seeing worse. The bad grammar, poor spelling, lack of information and generally sloppy appearance of the brief documentation that comes with this fine program are discouraging but shouldn't prevent you from purchasing the product because the program itself is very easy to use.

This is a very interesting program, probably unique in its function, that deserves attention. With some changes to the documentation this would be a fine program for anyone who has the need to solve linear equations or convert higher order equations to a form that can then be looked up in various tables.

It is a bargain and so easy to use that most people will find the instructions more of a minor annoyance than an obstacle. This program isn't flashy or complicated to use; it just sits there and lets your CoCo perform a task it has never been able to do before.

*Algebra* is the answer to many students' prayers for a program that would "really solve" some of those jumbled masses of numbers and letters that teachers always assign as homework.

## Modest Packaging Can't Hide SCRDMPI0's Usefulness

— John Ogasapian

Let's get the negative thing out at the beginning. *SCRDMPI0* comes modestly packaged on an average quality commercial cassette. The documentation accompanying the cassette is two paragraphs that very briefly describes what the program does and how to get it up and running.

But don't be discouraged and don't be fooled. Once it's fired up, you discover that behind those humble trappings is one slick screen dump program that reproduces a PMODE 3 or 4 screen through a DMP printer with a minimum of fuss and bother.

*SCRDMPI0* comes on a cassette in the form of a BASIC driver. When the program is run, you are provided with a pair of prompts. The first is for your printer's graphics control codes (ENTER defaults to those of the DMP-100, which worked fine with my DMP-110). The second prompt asks for a loading address, and here you may run into a snag, since the documentation is neither clear nor helpful (hint: Begin the routine from a cold start — POKE 113,0:EXEC40999, and if you have a DMP-100 or one of its cousins, try entering 14848). The actual ML program is poked in and you are given the choice of saving it to tape or disk.

Now you're home free. After loading or drawing the graphics screen, simply enter EXEC. You are prompted for the choice of background color (to reverse the printout) and the Baud rate for the printer. Position the paper at the top of the head, push ENTER, and away you go! It's as easy as that.

As soon as the print is finished, you are offered the choice of rerunning (again with the option of reversing the colors) or returning to BASIC to draw or enter another graphics screen and repeat the cycle.

The finished format is six and a half by seven inches sideways, and, as might be expected, there is a degree of distortion in the printout, relative to what is on the screen. I also discovered that I could not copy the ML program between drives. So, unless you are luckier than I, you'll have to use the Backup routine and then copy whatever else you might want on the disk with it. I'm not into drawing with my CoCo, but I did try the program as a tool for printing out graphs generated by Tom Mix's *Teachers Data Base* and billboards of some other pieces of software in my library. I had no problems at all.

The main problem with this software is its poor documentation. If you can get by that and the modest packaging, you have a quick and easy, black-and-white screen dump.

## Software Review

# Advanced Utilities — Five OS-9 Goodies

— Mark Sunderlin

I was once asked, "What do you buy for the computer when you already have a word processor, a spreadsheet and a database?" I told the questioner to buy utilities. Utilities are those wonderful little programs that exist only to make the computer do more for us in an easier way. To this end, Computerware has released a set of five OS-9 utilities packaged together as *Advanced Utilities*.

The five utilities included in the package are *Kshell*, a direct, more powerful replacement for Shell; *Cpy*, a more powerful version of Copy; *Archive*, a backup utility to back up hard disks to floppies; *Flink*, a program to define what file to use as the boot file on the next reboot; and *Unload*, used to remove a program from memory by recursively unlinking it. The five utilities come on one disk, which is unprotected. Also included is a nine-page manual that explains the utilities and has examples for each. The explanation for *Kshell* is wonderful and covers all aspects of this utility. The other utilities seem to suffer a little, though.

*Kshell* is the showpiece of the package. This program takes the place of the shell to give a more powerful command interpreter. Its most powerful feature is its automatic wild card extensions. Any place on an OS-9 command line you would place a filename, you can use a wild card. Use it to match any file in the directory that corresponds to a pattern. The "\*" character matches any set of zero or more characters. The "?" character matches any single character. For example, the command "del \*.bas" deletes all files in the current directory that end with ".bas." The command "del file?.txt" deletes files such as "file1.txt," "file2.txt," "filez.txt" and any other file that fits the pattern. This works with all OS-9 commands that let you give a list of filenames on the command line. It does not work on OS-9 commands that only use the first filename found on the command line. For example, the ident command only reports the first file no matter how many are stated. Thus, a wild card used with ident still only gives one report.

Another feature of *Kshell* is its PATH variable. You may

assign a value to PATH to tell OS-9 where to look for commands. The command PATH="/d0/;mds;/ho/cmds" tells OS-9 to look for a command first in /c0/cmds and then in /h0/cmds. You can give it as many paths to search as wanted. Although not stated in the manual, I am sure there is some limit on the number of characters the PATH can have.

In addition to the PATH variable, *Kshell* gives four user-defined variables or macros. These are called \$1 through \$4. You may give them any value and substitute them anywhere in an OS-9 command line. If you set \$1 equal to /d1/docs/reviews/advutil.txt, all you have to do to edit that file is enter edit \$1. In addition to saving repetitive typing, macros can be passed to procedure files. The command proc \$2="program" sets \$2 to the string "program" for the duration of the procedure proc. In the procedure file you would use \$2 in any place you would normally use a filename.

*Kshell* has also borrowed some ideas from the UNIX operating system. One of these is the use of the single opening quotation mark (') character. If a command is placed within single opening quotes, its first line of output is substituted at that point. For example the command echo Current directory is 'pwd' generates this: Current directory is /d1/docs/reviews. You can mix this with the macros. \$1='pxd' sets \$1 equal to the current execution directory. Also from UNIX, *Kshell* accepts either the standard CHD and CHX commands or the aliases of CD and CX. Comment lines under *Kshell* may begin with either an "\*" as in the normal shell or with a "#" as in UNIX.

The *Kshell* has a few other features. The command prompt can be set to whatever is wanted. The command "-p="Ksh">" would replace the OS9: prompt with Ksh>. Error reporting can be turned on or off. The command "-e" turns on full error messages like PRINTERR does. The difference is that a -ne command turns off error reporting. You can also specify what file to take the error messages from if you want to use something other than the system default. *Kshell* has improved upon the standard shell's redirection capability. The output of a command may be redirected to a file, to append to an existing file, or to overwrite an existing file.

To get all these extra features, you must give up something; what you give up is memory. *Kshell* reduces available memory by about 3.5K. This may be a problem if memory is tight. Using *Kshell* and a Hi-Res screen utility only left enough room for me to edit an 8K document with my word processor, as opposed to a 12K document under the standard shell. Those using the standard 32 by 16 screen or an 80-column hardware card may not miss that 4K as much.

If it seems that the other four commands are getting the short end of the stick here, you're right. They also get short-changed in the manual. While *Kshell* is covered in full detail with several examples for each command, the rest of the utilities are covered in less detail.

The *Cpy* utility is an improvement upon the Copy command. In addition to the standard Copy features, it can copy multiple files to a directory. Used with *Kshell*'s wild cards it can be very useful. The command "Cpy /d1/source \*.bas" copies all BASIC programs in the current directory to /d1/source. *Cpy* is a little confusing though. To copy one file to another, the syntax is "Cpy source destination," while to copy multiple files it is "Cpy Destdir source1 source2 source3 . . ." I have no idea why the author used two separate syntaxes on the same program.

The *Archive* utility is used to back up hard disks or large

floppies to smaller media. It allows copying these large media to several smaller ones without splitting files over two different smaller medias. This command worked exactly as the documentation said it would and without any problems.

The *Unload* utility is a recursive version of *Unlink*. *Unload* removes a module from memory by repetitively unlinking it until it disappears from memory. This also worked as described.

The final utility, *Flink*, is an interesting one. It allows you to state what file the system is to boot from on the next reboot. Thus, you could have two or more boot configurations on a disk and choose which to use. This also seemed to work just as it is described.

What you get with *Advanced Utilities* is a very good replacement shell and four good utilities. The *Kshell* itself is worth the price. All five utilities were tested under both OS-9 Version 1.01 and Version 2.0 without any problems.

## Software Review

### **XSCREEN Gives High Resolution for OS-9**

— Mike Piotrowski

Have you ever wanted something besides the 32 characters per line on a green screen for your OS-9 system? Unhappy with inverse video instead of lowercase letters? Well here it is, and at an affordable price.

*XSCREEN* is a high resolution screen package for the OS-9 operating system of the Color Computer. Choose from 51, 64 or 85 characters per line. The display can be white or green characters on a black background, or black characters on a white or green background. All of these combinations have 24 lines per page. *XSCREEN* also has real lowercase letters.

The 85 characters per line is nearly impossible to read on a television. However, it is readable on a monitor. The 64 characters per line is available in two character sizes; wide and narrow. The wide characters seemed easier for me to read, but judge this for yourself. At 64 characters per line with either the wide or narrow characters, my eyes got tired after 15 minutes of work on the television. The 51 characters per line was easy to work with for long periods of time on a television or a monitor.

After *XSCREEN* is copied to the command directory, activate it by typing *XSCREEN*. You are presented with a menu for selecting the characters per line, and the foreground and background colors. It then returns to OS-9. If you need to change to a different style screen or want to quit *XSCREEN*, return to the menu by pressing the *CLEAR* and *BREAK* keys at the same time. If you quit *XSCREEN* and want the high resolution screen back again, reboot the system before executing *XSCREEN* again. This is stated in the manual in large bold letters.

*XSCREEN* uses about 12K bytes of user memory. This leaves about 28K bytes of memory for applications. If using *BASIC09*, you will have about 7,000 bytes of memory for your application program. With the OS-9 editor you will have just over 24K bytes of memory for the text you are editing.

All of the OS-9 display functions are supported by *XSCREEN*. In fact, *XSCREEN* has additional display functions which make it much easier to write screen editing routines. These codes allow erasing to end of line, turning the cursor on or off, scrolling down and erasing to end of screen. There are also several codes to change the fore-

ground and background colors of the display and change the number of characters per line. To get these additional display codes, *XSCREEN* uses some of the OS-9 graphics display codes. This may sound like a problem if you have existing programs that make use of these codes. It is not, and here is why. *XSCREEN* does not use the standard output */TERM*. Instead it uses a driver called */H1*. To perform the standard OS-9 graphics display functions, direct the display command to */TERM*. OS-9 will process the display code the same as if *XSCREEN* was not present.

I tried *XSCREEN* with OS-9 version 2.0.0. It did not operate predictably. I hope the makers of *XSCREEN* will make the necessary changes so it will work properly with version 2.0.0.

*XSCREEN* is well worth the price. It is nice to be able to get more than the 32 characters per line. If you cannot afford 80-column hardware, *XSCREEN* is the way to go.

## Hardware Review

### **Super RAMDisk Provides Mega-memory**

— Dan Downard

"More memory!" they cried. "Give us mega-memory like the competition!" Well, CoCo users, now you have it. Spectrum Projects is distributing a 256/512K memory expansion from *DISTO*.

To use the *DISTO* Super RAMDisk you need a 64K CoCo and a Multi-Pak Interface. Why the Multi-Pak? It enables the *DISTO* upgrade to be used with the CoCo 2, something that has been hindering previous upgrade kits. The unit is housed in an attractive white metal case and plugs into slot number 2 of the Multi-Pak. Software for both *Disk BASIC* and OS-9 is provided on disks to use the memory upgrade as a RAM disk. With OS-9, up to three RAM disks can be connected at one time.

What is a RAM disk and do I need one? At present, the only software that exists for the Super RAMDisk is drivers that make the memory expansion act as another disk drive. It's not a normal disk drive, though. First, it is super fast. The total time to read or write a sector is less than 5/1,000 of a second. One of the uses could be storage of graphics pages for fast recall during games. Can you write longer programs? Not with the present software. The problem is not with the memory expansion, but with the software. As you are probably aware, *Disk BASIC* only recognizes 32K of user memory.

To write longer programs, you could split them into small ones, but you can do the same thing with a regular disk drive. You will have to find a way to pass parameters (variables) between programs if necessary. A suggestion is to store them in a file and recall them when necessary. Remember, we are talking about a quiet, fast disk drive that consists of RAM.

The use of a RAM disk with OS-9 is another story. Regarding longer programs, as long as modular programming concepts are used, *BASIC09* already has the facility to pass parameters between programs. Considering the disk intensive nature of OS-9, it's a welcome relief to transfer the commands directory to the RAM disk and watch it fly. Commands execute almost instantaneously. No noise either.

continued on Page 47

# Bleeps, Bloops, Bells and Whistles!

By William Mitchell



This program, *Soundbase*, represents an inventory of sounds used in games and educational programs I have written. They have been developed over several months and saved so they can be appended to any program, then called for subroutines as needed. When the program is completed, the unused subroutines are deleted, but I usually use all of them in most programs. *Soundbase* can be used to form a basis for many programs.

(Editor's Note: Unplug the disk controller, if you have one, before loading the program.)



180	.....68
330	.....249
450	.....125
650	.....193
840	.....129
END	.....217

```

170 RETURN
180 '
ZAP OF VOLTAGE
190 PLAY"V20;L255;O1;1;2;3;2;3;4
;3;4;5;4;5;6;5;6;5;4;5;4;3;4;3;2
;3;2;1"
200 RETURN
210 '
HISS OR FIZZ
220 PLAY"L255;O1;V30;1;V5;2;V6;3
;V7;2;V5;3;V4;4;V3;5;V3;4;V4;3;V
6;4;V4;5;V5;6;V4;5;V3;4;V2;5;V3;
6;V2;7;V3;8"
230 RETURN
240 '
BLIPP
250 PLAY"L255;O1;V30;1;V26;2;V22
;3;V18;4;V14;5;V12;6":SOUND 1,1
260 RETURN
270 '
280 '
SPRING
290 PLAY"O2;L255;V15;1;O3;V10;1;
V9;2;V8;3;V7;4;V6;3;V5;4;V4;3;V3
;2;V2;1;V1;1;2;3;2;1;2;3;2;1;2;3
;2;1;2;3;2;1;2;3;2;1;2;3;2;1"
300 RETURN
310 '
TAUNT
320 PLAY "V3;O3;L16;10;10;7;12;L
8;10;7":RETURN
330 '
CHARGE
340 FORX=1TO2
350 PLAY "L4;V4;O4;L16.;1;L32;1;
L16.;1;L32;1;L16.;1;L32;1;L16.;1
;L32;5;L16.;8;L32;5;L16.;8;L32;5
;L16.;8;L32;5"
360 NEXTX:PLAY"1":RETURN
370 '
BIG SPRING

```

```

380 FORV=31TO1STEP-1:PLAY"O1;V"+
STR$(V)+";L255;8":NEXTV:RETURN
390 '
BOUNCING BALL
400 PLAY"T2;L255;O1;V31;1;V20;1;
V10;1;P2;V9;1;P5;1;V8;P10;1;V6;P
15;1;V4;P20;1;V2;P25;1;V2;P30;1;
P35;1;P45;1;P60;1;P80;1"
410 RETURN
420 '
SIREN
430 PLAY"T255;L255;O4;V1;1;2;V2;
3;4;V3;5;6;V4;7;8;V5;9;10;V6;11
;L4;12;L255;V5;10;9;V4;8;7;V3;6;
5;V2;4;3;V1;2;L1;1"
440 FOR DLAY=1TO200:NEXTDLAY:RET
URN
450 '
WOLF WHISTLE
460 PLAY"T255;L255;O4;V2;2;V2;2;
V3;3;V4;4;V5;5;V6;6;V7;7;V8;8;V9
;9;V10;10;V11;11;V12;12;O5;V13;1
;V14;2;V15;3;V16;4;V17;5;V18;6"
470 FORDLAY=1TO100:NEXTDLAY:GOSU
B430:RETURN
480 '
MACHINE GUN
490 FORX=1TO6:PLAY"O1;L255;4;3;2
;1":NEXTX:RETURN
500 '
HOORAY FOR THE RED WHITE & BLUE
510 PLAY "O4;V5;L8;C;C;O3;L16;A#
;A;L4;A;V5;L8;G#;A;L2;A":RETURN
520 '
FOR SCREEN DISPLAY
530 KOLOR=RND(7):PATTERN=RND(15)
540 FACTOR=128+(16*KO)+PA
550 FOR T=1TO20
560 PRINT STRING$(32,CHR$(FA));

```

## The listing: SOUNDBSE

```

10 ' SOUNDBASE
COPYRIGHT BY WILLIAM L. MITCHELL
104 CLUBVIEW RD
ENTERPRISE, AL 36330
NOV 1985

20 CLEAR 200
30 POKE359,0
40 GOSUB610
50 GOTO650
60 ' SOUND SUBROUTINES
70 '
BUZZER
80 FORV=30TO2STEP-2:PLAY"V31;L25
5;O3;5":NEXTV:RETURN
90 '
DEPTH SOUND
100 PLAY"T2":FOR V=20TO0STEP-5:P
LAY"O4;V"+STR$(V)+";L4;12"
110 FORDL=1TO20:NEXTDL
120 NEXTV
130 FORDL=1TO300:NEXTDL
140 RETURN
150 '
SIREN WARNING
160 PLAY "V30;L200;O4;1;2;3;4;5;
6;7;8;9;10;11;12;O5;1;2;3;4;5;6;
7;L100;8"

```

continued on Page 26

# CoCo SYN

Even if you don't know much about music, banging away on the keys of a piano can be a lot of fun. If you have always wanted to compose music, but didn't have an instrument or the time to learn how to play, now you do. The CoCo Piano-Synthesizer/Composer makes composing tunes easy. It turns the CoCo's keyboard into a piano keyboard, remembers the notes played and plays them back at any tempo you choose. It provides a powerful editor for correcting or altering the stored music data.

The CoCo piano allows you to save each composition on tape or disk, and to load them as stand-alone machine language program that EXEC without the help of the BASIC language driver. Furthermore, it automatically create PLAY statements complete with line numbers and stores them on tape or disk for merging into BASIC programs. This function is in addition to, and totally separate

from, the ability to store the music as synthesized ML programs. You may reload your ML composition back into the BASIC editor at a later date and add to or edit the composition. Please note that the CoCo piano does not play chords, only single notes.

The CoCo Piano-Synthesizer/Composer can be programmed to sound like a wide variety of instruments (as well as non instruments). It contains two envelope tables that can be programmed by the user. With a bit of experimentation, it can sound like a jazz piano, an organ, an echo chamber or like no instrument ever heard before. You can toggle from one envelope to the other at any time while composing. This change is recorded in the musical data and, on playback, taggles a change to the other envelope. When the composition is saved, any changes made in the envelope tables are automatically saved along with the music data.

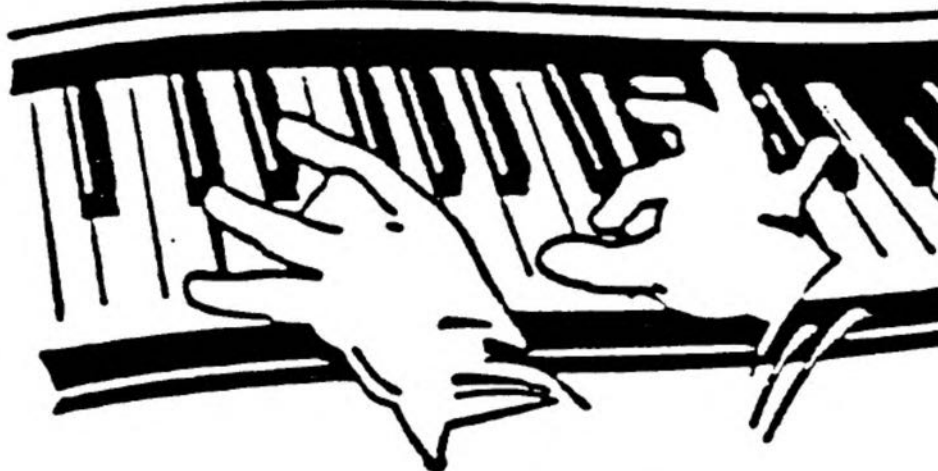
## HOW TO USE THE PROGRAM

There are two separate program listings. The first one boots the machine language synthesizer into memory and saves it on tape or disk as PIANO/BIN. Line 2 of the booter contains checksums for each of the data lines. Provided all of the entries in Line 2 are correct, the program tells in which line you have made a mistake in typing the rest of the data statements. **DO NOT RENUMBER** this program! If you get an error report and can't find the error in the line reported, check to see if the checksum in Line 2 is correct.

The second program listing is the BASIC language utility that allows easy management of the ML program. While keying it in, it is best to include only those spaces between commands that you see in the listing. We economized on memory to allow 16K users to use the piano, and extra spaces use extra memory. Save this program on disk or tape as PIANO/BAS. PIANO/BAS loads PIANO/BIN each time it is run, so both programs should be on the same disk or tape. Run the BASIC program. A prompt asks if the ML program is to be loaded from tape or disk. Once the ML program has been loaded, the main menu appears.

Now press "1". This puts you in the play/composite mode. Begin playing music and the computer stores the notes and their lengths. The length of the note depends mostly on the length of time your finger remains on the key. However, the program does record information on the length of time between keystrokes up to a maximum of one second.

In this mode, a text screen



# THE SIZER

by Martin and Jeremy Spiller

representation of the keyboard appears on the TV. The keyboard looks like a two tiered organ keyboard. All of the keys in the second row and most of the keys in the bottom row are mapped to resemble the white keys on a piano. They are each marked with the note that the key plays when depressed. The keys above them represent the black keys on the piano. Notice that some of the keys in the first and third rows are colored blue on the map. These keys produce no sound when pressed, and are not recorded in music memory. They represent sharps and flats that do not exist on the chromatic scale; their omission gives the keyboard the appearance of a piano keyboard.

Three octaves are represented, starting at middle 'C'. They go from the up arrow key to the 'Y' key, from the 'U' key to the right-arrow key, and from the 'Z' key to the "<" key.

Some of the keys are colored red - these are control keys. They are the CLEAR, ENTER, SHIFT, "?" and spacebar. These can be pressed at any time while in the play / compose mode. The SHIFT key exits the synthesizer and returns to the main menu.

In order to simplify playing the keyboard, consider using black and white self adhesive tabs to cover the appropriate keys. These can be obtained at any stationery store. If you can't find these, then try small pieces of electrician's tape.

## Playing versus Composing

The machine code program was originally formulated to allow experimentation. The object was to allow the user

to modify a pure tone by manipulating its volume over time. This is done by specifying the volume at discrete intervals in what is known as an envelope pattern. Some envelope patterns, such as the piano envelope, sound lovely if stretched out over a long period (a high envelope delay) and if played only once per key press. Others can give interesting effects if the delay is shortened and if the envelope pattern is repeated for as long as the key is pressed.

As the program evolved, however, the keyboard developed into a piano and we decided to store the notes and their lengths in memory. We discovered that when repeating patterns were used, memory filled up too fast. For this reason, we decided to allow composing only in the non-repeating mode. While you get meaningful note data in the repeating mode, the length data may not be correct. However, even when using a repeating pattern, only one note byte and two length bytes are stored per key strike, so memory is conserved.

The program is configured at execution for composing. We have chosen an arbitrary envelope delay (representing tempo) of 1100, and a non-repeating piano envelope as default. Whenever a note key is pressed in the play/compose mode, note and length data is stored in memory whether you want to keep the data or not. You will store meaningful length data as long as you do not switch to a repeating envelope pattern by pressing the space bar. If you mistakenly press the space bar, press it again to return to a non-repeating envelope pattern.

Now that you are in the play/com-

pose mode, go ahead and try playing some music. For 16K users, about 1,000 notes can be stored before running out of room; 32K users should be able to bang away for most of the day before hitting the top of RAM. If this happens, don't worry. The binary program checks to see if the limit has been reached and returns to the menu. Nothing has been lost, and you can still play back, alter and save the stored data.

Each time a note key is pressed, the CoCo remembers the note and the length of time your finger was on the key. It also keeps track of the time between keystrokes up to a second. If you are inexperienced and spend a lot of time looking for the next key to press, the program takes no notice beyond one second. For experienced piano players, the only problem is getting used to the keyboard itself. Remember that if you press a second key while the first is still pressed, there will be no response until you take your finger off the first key.

## The Control Keys

Correction of mistakes in the play/compose mode is limited to the CLEAR key. Whenever this key is pressed, the last note played is eliminated. Pressing it twice or three times eliminates the last two or three notes played. The entire composition can be eliminated this way. Any notes eliminated are replaced with the next note keys pressed.

When you are finished composing and want to return to the main menu, simply press either SHIFT key. If you mistakenly exit the play mode, return to the same position in the composition by pressing '2' at the menu. Beware. If the '1' is pressed, the program assumes you

want to compose a new piece and dumps any music already in memory.

The computer keeps track of pauses that last up to one second. This creates a problem with longer or multiple pauses that normally occur in musical notation. The ENTER key makes it possible to place pauses in the music. It works the same way the note keys work. The pause lasts for as long as you press the ENTER key. During composition, this creates a high pitched noise that indicates something is being pressed. This noise does not occur on playback.

Of course, if you are only fooling around with the synthesizer and do not care about what is stored in memory, you do not need the pause button at all. You will, however, want to use the space bar. This control toggles between repeating envelopes and single envelope strikes. Remember that repeating envelopes do not give meaningful note length data and should be avoided when composing.

The final control key is the question mark-slash key. The synthesizer program contains two programmable envelope tables. The question mark key toggles between the two tables and allows playing of different parts of a composition with different sounding envelopes. Any time this control key is pressed, a code is stored in the music memory that triggers a corresponding shift during playback.

### Envelope Delay and Play Delay

How long is a whole note? The longest note stored in memory is a whole note and is proportional to the envelope delay. Press a note key in the default mode and listen carefully. The sound trails off to nothing. The length of time it takes for the sound to fade away totally is the longest note length stored in memory. Keeping your finger on a key longer does not lengthen the time between that note and the next during playback.

If you are playing a very slow tempo piece and need a longer whole note, simply increase the envelope delay. If you are playing a very fast piece and want to hear more of a fadeoff during short keystrokes, shorten the envelope delay. It is best to alter the envelope delay before storing data to keep. A short keystroke with a long envelope delay causes the program to store only a part of the entire envelope. Increasing the envelope delay after the data has been stored lengthens the note, but it does not increase the proportion of the envelope played. The envelope delay may be altered from the main menu by pressing the "change tempo" option.

Numbers between one and 65,535 will work. Try the default envelopes using an envelope delay of 500.

The play delay can also be altered from the main menu by pressing "change tempo." While the envelope delay has an effect in both the play and playback modes, the play delay operates only in the playback mode. It accounts for a constant pause between notes when they are played back. It is included as a fine adjustment for playback timing. The default value is 50.

This is the first parameter to alter if the playback is too fast or too slow. Values between one and 65,535 are accepted. Small increases or decreases don't make much difference, so don't be afraid to change it by factors of 1,000. You may later alter the play delay or the envelope delay at any time, even after composing a piece. They do not affect the stored music data once it has been placed in memory, but they affect the playback of that data.

### Programming the Envelope Tables

Before programming an envelope, you must know something about synthesizing periodic sounds. The CoCo has no sound-generating circuitry. In order to produce sounds, the microprocessor must minutely manipulate the voltage output to the loudspeaker. (That it can do this quickly enough to produce a fantastic array of sounds is a testament to the extreme versatility of the 6809 and the Color Computer.)

In order to produce the sound of a particular instrument, most hardware synthesizers produce a particular sine wave electrical output varying between zero and five volts. This pure tone is then modified to produce the same general pattern of electrical output as the sound output of the instrument it is trying to mimic. The unmodified sine wave produces a pure tone of a particular volume depending upon the maximum voltage allowed by the circuitry. The envelope modifications are mainly constraints on the volume of the sine wave over time.

To produce the sound of a piano, it begins with a maximum volume when the key is struck, falling off rather quickly at first and then more slowly until it fades out entirely. This is exactly what the *CoCo Piano-Synthesizer* does: It produces a square wave instead of a true sine wave, the maximum voltage of which is controlled by the values in the envelope table.

Go back to the main menu and press '6'. When the prompt asks which table to use, press '1' and ENTER. Now remember what we said about the fast

falloff and a gentle fadeout? The graph shown is the envelope used to produce the piano default sound. This envelope can be altered or a totally new one created by using the left joystick and firebutton. For example, position the cursor at the bottom left side of the screen and press the firebutton. The original point disappears and is replaced by the new one. Now move the cursor one position to the right and four above the one just set and press the firebutton again. Continue this process, depositing points three or four higher in each succeeding position, until it reaches the top of the screen. Then, for the next cursor position, move the cursor to the bottom of the screen and repeat the same process.

Do this until you reach the last cursor position on the right. You have created a series of upward sloping lines. To be really fancy, make the top of each succeeding line several positions lower than the last one to create a trail-off effect. When you are finished creating the envelope, press any key to return to the menu.

Now try playing any selection still in memory. If there is no stored data, just play something from the compose mode. Try changing the tempo. Type in an envelope delay of 100. Press '1' at the menu and get into play mode. Play a few notes and then press the space bar. Now play the same notes over. Interesting!

A word about coherent values. While you can type in literally any envelope pattern wanted, the best ones follow a pattern. Each succeeding value should bear a rational relationship to its neighbors rather than being a random scattering of points. Maybe the points follow some sort of curve, or a pair of curves. Maybe every third point is offset from the curve by some fixed amount. The possibilities are endless, and when combined with various envelope delays and repeating patterns, quite a collection of different sounds can be created. Both envelope tables are programmable. An organ-like effect can be produced by using an envelope that is a straight horizontal line at the top of the screen.

For those interested in experimenting with sound, try a rational envelope using very short delays (say 50) and a repeating pattern. Note that different keys vary in tonal quality as well as in pitch. The reason for this involves the interference patterns produced as the envelope delay gets nearer to the frequency delay (the delay used to produce the desired pitch). You may hear different "beat frequencies" with different keys.



## Saving and Loading Compositions

Once you have composed and perfected the music, *PIANO/BAS* provides the means to save it on disk or tape. This is done from the menu and is self-prompting. It is saved with any altered envelope tables, envelope delay and play delay that are *POKED* in while composing, so each composition has its own unique sound. You do not have to run the BASIC driver in order to play the music. Just *LOADM* or *CLOADM* whatever filename is used to store it. Then type *EXEC* and the piece plays.

If it needs further editing, run the BASIC driver and load the previously stored piece from the menu. Add to it, change the envelopes, the delays, or alter the note data and then resave the changes.

## The Editor

We have done our best to confine the BASIC program in order to allow 16K owners to use the piano. The editor is simple, but adequate for manipulation of program data. It windows any 15-note segment of your composition, and allows replaying any part of that segment from the beginning to the cursor position.

Play something from the compose mode and press '9' at the main menu. The first column on the left is the note position. It starts at zero and increments for each note in the composition. The other columns represent actual note data. Each note is represented in memory as three data bytes. The second editor column translates the first data byte into an octave and a note. The third column contains the note length. These numbers go from one to 32; 32 is the longest note played. (If most of these numbers are less than 15, you might consider resetting the envelope delay to a smaller number and replaying the piece. That way, more of the envelope is heard during each keystroke.)

The last column is the pause interval that keeps track of the time between keystrokes. It is a number between one and 255. The higher the number, the longer the pause. This pause is not to be confused with the pauses intentionally placed in the data by pressing the *ENTER* key. Those pauses are stored just like any other note, with a note length and an interval pause.

The cursor can be moved up or down by pressing the appropriate arrow keys. Notes may be changed, inserted or deleted by positioning the cursor at the note position to be changed or deleted. Insertions occur at the cursor position, and the remainder of the composition is moved one position higher in memory. Inserting and deleting may take

some time in long compositions since the entire data array above the cursor position must be relocated, and this relocation is done from BASIC.

Insert intentional pauses by typing *PAU* in place of a note. Pauses are like any note in that you must specify a note length and pause length. An envelope table switch may be inserted by typing *ENV*.

For the inexperienced piano players, we have included an option that allows the user to change both the note lengths and interval pauses en masse. Pressing '9' while in the editor allows you to specify numbers that are automatically inserted into all note positions from the top of the screen to the cursor position. Since any note position can be specified as the first note in each segment (the top of the screen), you can make large-scale changes in tempo or timing with reasonable precision.

## Assembling PLAY Statements

Pressing '8' at the main menu causes the computer to build BASIC *PLAY* statements. These are assembled directly to disk or tape as a BASIC program complete with line numbers. Load them and type *RUN* to hear them. Those with disk drives can merge them into their own programs after adjusting the line numbers using Extended BASIC's *RENUM* function.

Line 5 is a tempo line. This line may be altered to speed up or slow down the playback. Since these are run from BASIC without the help of the *ML* synthesizer, envelope changes are ignored. (Actually, there is a way to get a "pseudo envelope" in BASIC. Those interested should contact Jeremy Spiller for information on how to obtain the program for this.) The *PLAY* statements offer another method of editing compositions as they can be manipulated using Extended BASIC's editing functions.

## Tuning Your Piano

While we feel reasonably competent to write computer programs, neither of us play the piano, nor do we even know much about music. We tuned the piano by ear and if you can do better, give it a shot. The key table is located at &H3180. The assembly listing shows it from lines 16100 to 31300.

Look carefully at the assembly listing. Each key is represented by three bytes. The name of the key is commented to the right of its first byte. (Don't confuse these with the three data bytes stored in memory while composing. These begin at &H334C.) The first two bytes are the frequency delays and these account for the pitch of the key.

The third byte is the note "name" and is the value stored in the first of the three data bytes while you are composing. The higher the frequency delay values, the lower the sound produced by that key. Note that in most cases, both delay values are the same. Keeping them equal or close to equal keeps the wave square.

The wave shape could be altered toward triangular by drastically reducing one while increasing the other. The wave doesn't have to be square. The delay values can be from one to 255. To sharpen the sound of a key, locate that key in the comment column of the key table and reduce the numbers in the frequency bytes. (The actual memory address is the hexadecimal number in the column farthest to the left in the assembly source code listing. As an example, to sharpen the sound produced by the *BREAK* key, *POKE* numbers lower than 52 into addresses &H3216 and/or &H3217.)

To lower the entire keyboard an octave, *POKE* higher numbers into the frequency bytes of all the keys in the table. Do this by trial and error, sharpening or flattening each position until it sounds right. Once you get the table the way you want it, play a tune and save it as *PIANO/BIN*. Then, whenever you run the BASIC driver, it automatically loads the modified tables and becomes the new default. Have fun!

## Key Memory Addresses

&H3000	Execution address of play/compose segment.
&H325A	Execution address of playback segment.
&H330D and &H330E	LSB and MSB of address of last note played. <i>POKE</i> another address here to end playback at another note position.
&H334C	Address of first note of compositions.
&H3262 and H3263	Holds LSB and MSB of address of first note played (usually holds &H334C). <i>POKE</i> another address here to begin playback at another note.
&H3180	First byte of key table.
&H3219	First byte of envelope Table 1.
&H323A	First byte of envelope Table 2.
&H317E and &H317F	Holds top-of-RAM (&H7F80 in 32K systems).

```

180 .....237
280 .....161
410 .....247
480 .....33
590 .....91
750 .....23
860 .....71
960 .....163
8014 .....164
END .....183

```

### Listing 1: PIANOBIN

```

0 GOTO1000
1 CLEAR200,&H2FFF
2 DATA 1847,2952,2629,2549,2518,
1653,1806,1698,1525,2938,1478,17
37,1948,2446,2333,1796,1916,1631
,1711,1186,867,1187,1180,1006,16
98,879,1382,1025,1322,1544,1951,
2067,2059,1813,1907,2319,2268,16
09,1754,2414,1878,2275,637
4 DIM SUM(43):FOR Z=1 TO 43:READ
SUM(Z):NEXT Z
10 DATA 8D,11,7F,31,78,8E,32,19,
BF,31,76,30,88,1F,BF,31,7A,20,1B
,B6
20 DATA FF,1,84,F7,B7,FF,1,B6,FF
,3,84,F7,B7,FF,3,B6,FF,23,8A,8
30 DATA B7,FF,23,1A,50,39,1C,FE,
BE,31,76,BF,31,73,FE,31,7C,86,FF
,B7
40 DATA FF,2,C6,FF,5C,79,FF,2,B6
,FF,0,8A,80,81,FF,26,6,C1,7,26
50 DATA EF,20,DB,34,4,B7,31,75,C
6,FF,5C,46,25,FC,86,8,3D,EB,E4,3
5
60 DATA 2,C1,2F,26,23,7C,31,78,B
6,31,78,44,25,E,8E,32,19,BF,31,7
6
70 DATA 30,88,1F,BF,31,7A,20,C,8
E,32,3A,BF,31,76,30,88,1F,BF,31,
7A
80 DATA C1,37,10,27,2,1E,10,8E,3
0,A2,E1,A0,27,94,10,8C,30,AA,27,
A
90 DATA 20,F4,1,6,B,1C,20,23,27,
37,C1,1F,26,31,7C,31,79,B6,FF,0
100 DATA 8A,80,81,FF,26,F7,16,FF
,71,B6,FF,0,8A,80,81,FF,26,F7,BE
,33
110 DATA D,86,20,A7,1,17,2,3F,E7
,2,30,3,BF,33,D,BC,31,7E,24,69
120 DATA 16,FF,4F,C1,31,26,F,BE,
33,D,30,1D,8C,33,4C,25,C4,BF,33,
D
130 DATA 20,BF,8E,31,80,86,3,3D,
3A,A6,2,A7,9F,33,D,EC,84,34,6,A6
140 DATA 9F,31,73,84,FC,E6,E4,8D
,7,4F,E6,61,8D,2,20,EF,5A,26,2F,
8A
150 DATA 2,B7,FF,20,B6,FF,0,8A,8
0,B1,31,75,27,1F,32,64,FC,31,73,
B3
160 DATA 31,76,BE,33,D,E7,1,17,1
,D9,E7,2,30,3,BF,33,D,BC,31,7E
170 DATA 24,3,16,FE,E9,39,33,5F,
11,83,0,0,26,C6,FE,31,7C,BE,31,7
3
180 DATA 30,1,EC,31,7A,27,5,BF,3
1,73,20,B4,B6,31,79,44,25,5,32,6
4
190 DATA 16,FF,52,BE,31,76,BF,31
,73,20,A1,32,19,BF,32,19,0,0,32,
38
200 DATA 4,4C,7F,FF,3F,3F,14,1,1
,80,1F,1F,20,25,25,1D,27,28,1C,9
0
210 DATA 90,6,1,1,80,20,21,1F,1D
,1E,21,53,53,F,1A,1B,23,1,1,80

```

```

220 DATA 15,16,26,19,19,24,1C,1C
,22,4B,4B,11,47,47,12,AC,AC,3,80
,80
230 DATA 8,2D,2D,1A,71,71,A,5E,5
E,D,22,22,1E,98,98,5,2A,2A,1B,65
240 DATA 65,C,30,2F,19,C1,C1,1,1
,1,80,37,39,16,31,31,18,0,0,0
250 DATA 1,1,80,B6,B7,2,A2,A2,4,
1,1,80,88,88,7,78,79,9,6B,6B
260 DATA B,1,1,80,58,59,E,4F,4F,
10,43,43,13,13,13,28,17,17,25,3B
270 DATA 3B,15,14,14,27,1,1,64,5
,5,FF,1,1,64,34,34,17,FF,CA,AA
280 DATA 91,7E,6B,5B,50,46,3D,35
,2D,28,22,1D,1A,17,15,12,10,F,D,
C
290 DATA A,9,8,7,6,5,4,3,2,1,FF,
89,44,4,E6,95,41,4,D2,91
300 DATA 41,4,C0,79,41,4,AD,70,3
4,4,9A,63,2C,4,86,54,29,4,74,48
310 DATA 22,0,17,FD,B6,7F,31,78,
10,8E,33,4C,8E,32,19,BF,31,76,FE
,31
320 DATA 7C,BE,31,76,BF,31,73,10
,BC,33,D,27,37,E6,21,BE,31,76,3A
,BF
330 DATA 32,B1,E6,A4,8E,31,80,C1
,64,26,1D,7C,31,78,B6,31,78,44,2
5,A
340 DATA 8E,32,19,BF,31,76,31,23
,20,CC,8E,32,3A,BF,31,76,31,23,2
0,C2
350 DATA E1,2,27,7,30,3,20,F8,39
,32,33,EC,84,34,6,A6,9F,31,73,E6
360 DATA A4,C1,FF,26,1,4F,E6,E4,
8D,7,4F,E6,61,8D,2,20,EA,5A,26,2
8
370 DATA 8A,2,B7,FF,20,BE,31,73,
BC,32,B1,10,23,FE,66,8D,55,CE,0,
32
380 DATA 33,5F,11,83,0,0,26,F8,3
0,1,BF,31,73,32,64,31,23,16,FF,7
2
390 DATA 33,5F,11,83,0,0,26,CD,F
E,31,7C,BE,31,73,30,1,BF,31,73,2
0
400 DATA C0,33,4C,C6,1,7F,FF,2,B
6,FF,0,8A,80,81,FF,10,26,FE,26,4
F
410 DATA 10,21,FD,A,10,21,FD,6,4
C,27,2,20,F3,5C,C1,FF,10,27,FE,1
1
420 DATA 20,DE,5F,4F,10,21,FC,F2
,10,21,FC,EE,4C,26,F5,5C,E1,22,1
0,27
430 DATA FF,66,20,EB,D
435 Z=0:SUM=0:L=10
440 FOR X=&H3000 TO &H334C:READA
$:A$=&H"+A$:A$=VAL(A$):POKE X,A:S
UM=SUM+A:Z=Z+1:IFZ=20 OR X=&H334
C THEN 450 ELSE NEXT X
450 PRINT"WORKING ON LINE #":L:
IF SUM<>SUM(L/10) THEN CLS3:PRINT
TE257,"ERROR IN LINE #":L:END
460 SUM=0:L=L+10:Z=0:IF X=&H334C
THEN 470ELSE NEXT X
470 CLS:PRINT"ENTER SYSTEM SIZE
(16 OR 32)":INPUT A
480 IF A=16 THEN POKE&H317E,&H3F
:POKE&H317F,&H80:GOTO 510
490 IF A=32 THEN POKE&H317E,&H7F
:POKE&H317F,&H80:GOTO510
500 GOTO 470
510 CLS:INPUT"(C)ASSETTE OR (D)IS
K":D$:IF D$="D"THEN SAVEM"PIANO"
,&H3000,&H334C,&H325A:END
520 CSAVEM"PIANO",&H3000,&H334C,
&H325A
1000 PCLEAR1:GOTO1

```

```

80 .....92
180 .....54
320 .....85
410 .....226
END .....5

```

### Listing 2: PIANOBAS

```

5 GOTO20000
10 CLS:PRINT"PLEASE REMOVE JOYST
ICK FROM RIGHT JOYSTICK PORT
":PRINT
14 CLEAR255,&H2FFF
20 INPUT"(C)ASSETTE OR (D)ISK":D$
:IF D$="C"THENLOADM"PIANO"ELSEL
OADM"PIANO"
40 DIMN$(12)
50 CLS:PRINT@14,"MENU":PRINT
60 PRINT" 1) COMPOSE MUSIC"
70 PRINT" 2) ADD ON TO MUSIC"
80 PRINT" 3) PLAY MUSIC"
90 PRINT" 4) SAVE MUSIC"
100 PRINT" 5) LOAD MUSIC"
110 PRINT" 6) CHANGE ENVELOPE"
120 PRINT" 7) CHANGE TEMPO"
130 PRINT" 8) ASSEMBLE TO PLAY
COMMANDS"
140 PRINT" 9) EDIT YOUR COMPOSIT
ION"
150 PRINT@480," ":A$=INKEY$:IFA
$=" "THEN150
160 B$=INKEY$:FORX=&H152 TO&H152
+7:IFPEEK(X)=255THENNEXT:GOTO170
ELSE160
170 ONVAL(A$)GOTO300,310,320,330
,340,360,440,8000,500
180 GOTO50
190 CLS:A$="c d z f g a z c d z
f g a":B$="C D E F G A B C D E F
G A B":FORX=1TOLEN(A$):C$=MID$(
A$,X,1):IF C$="z"THENMID$(A$,X,1
)=CHR$(175)
200 IFC$=" "THENMID$(A$,X,1)=CHR
$(128)
210 NEXT:FORX=1 TOLEN(B$):C$=MID
$(B$,X,1):IFC$=" "THENMID$(B$,X,
1)=CHR$(138)
220 NEXT:CLS0
230 PRINT@4,A$::PRINT@67,B$:A$=
CHR$(175)+CHR$(170)+CHR$(175)+CH
R$(170)+A$:A$=LEFT$(A$,22):A$=A$
+STRING$(4,255)+CHR$(128)+CHR$(2
55):PRINT@132,A$:B$=STRING$(3,2
55)+CHR$(133)+B$:B$=LEFT$(B$,21
):B$=B$+CHR$(128)+CHR$(255)+CHR$(
128)+STRING$(3,255)
240 FOR X=1TOLEN(B$):IFMID$(B$,X
,1)=CHR$(138)THENMID$(B$,X,1)=CH
R$(133):NEXT ELSE NEXT
250 PRINT@196,B$::PRINT@267,STRI
NG$(11,255);
260 PRINT@384,"ENTER = PAUSE : ?
= NEW ENVELOP"
270 PRINT"SHIFT = MENU : CLEAR=
BACKSPACE"
280 PRINT" SPACEBAR=REPEAT T
OGGLE
290 RETURN
300 GOSUB190:POKE&H330D,&H33:POK
E&H330E,&H4C:EXEC&H3000:GOTO50
310 GOSUB190:EXEC&H3000:GOTO50
320 GOSUB190:EXEC&H325A:GOTO50
330 CLS:S=&H3000:E=PEEK(&H330D)*
256+PEEK(&H330E):PRINT"SAVE":INP
UT"ENTER SONG NAME":F$:INPUT"(C)
ASSETTE OR (D)ISK":D$:IF D$="D" T
HEN SAVEM F$,S,E,&H325A:GOTO50
335 CSAVEMF$,S,E,&H325A:GOTO50
340 CLS:PRINT"LOAD":INPUT"ENTER
SONG NAME":F$:INPUT"(C)ASSETTE
OR (D)ISK":D$:IFD$="C"THENLOADM
F$ ELSELOADMPS
345 GOTO50

```



```

350 IFC=0THENRESET(A,B):RETURNEL
SESET(A,B,3):RETURN
360 CLS:INPUT"WHICH ENVELOPE DO
YOU WANT TO CHANGE (1 OR 2)";
E
370 IFE=0THENE=1
380 IFE=1THENE=&H3219 ELSEE=&H32
3A
390 CLSO
400 FORX=0TO63STEP2:A=PEEK(E):SE
T(X,31-INT(PEEK(E)/8),3):E=E+1:N
EXT:E=E-32
410 Z=JOYSTK(0):A$=INKEY$:IFA$<>
""THEN50ELSEA=JOYSTK(2):A=INT(A/
2):A=A*2:B=JOYSTK(3):B=INT(B/2):
C=POINT(A,B):SET(A,B,3):FORX=1TO
20:NEXT:GOSUB350:IFPEEK(&HFF00)=
253THEN430
420 GOTO410
430 POKEE+A/2,255-(B*8):FORY=0 T
O31:RESET(A,Y):NEXT:SET(A,B,3):G
OTO410
440 PLYDLY=PEEK(&H32E2)*256+PEEK
(&H32E3):ENVLDY=PEEK(&H317C)*256
+PEEK(&H317D)
450 CLS:PRINT"CURRENT PLAYDELAY="
";PLYDLY:PRINT"HIGHER OR LOWER V
ALUES WILL LENGTHEN OR SHORT
EN PAUSES BE- TWEEN NOTES ON PL
AYBACK ONLY. ENTER NEW VALUE,
OR <ENTER> TO LEAVE THE SAME
"
460 INPUTA:IFA<>0THENB=INT(A/256
):C=A-256*B:POKE&H32E2,B:POKE&H3
2E3,C
470 CLS:PRINT"CURRENT ENVELOPE D
ELAY=";ENVLDY:PRINT"LOWER VALUES
WILL COMPRESS THE ENVELOPE INT
O A SHORTER TIME AND INCREASE
THE TEMPO OF BOTH COMPOSITION
AND PLAYBACK. ENTER NEW VALUE OR
<ENTER> TO LEAVE THE SAME"
480 INPUTA:IFA<>0THENB=INT(A/256
):C=A-256*B:POKE&H317C,B:POKE&H3
17D,C
490 GOTO50
500 CLS:S=&H334C:P=0:FS=0:CP=0:M
=256*PEEK(&H330D)+PEEK(&H330E):L
P=(M-S)/3:GOSUB510:GOTO520
510 N$(1)="C":N$(2)="C#":N$(3)="
D":N$(4)="D#":N$(5)="E":N$(6)="F
":N$(7)="F#":N$(8)="G":N$(9)="G#
":N$(10)="A":N$(11)="A#":N$(12)=
"B":RETURN
520 FORX=0 TO448STEP32
530 IFF=LP THENFORZ=X TO448STEP3
2:PRINTSTRING$(16,CHR$(175)):NEX
TZ:GOTO620
540 N=PEEK(S+3*P):IFN=255THENN$=
"PAU":PRINT@X,P:PRINT@X+5,N$:GOT
O570
550 IFN=100THENN$="env":L1=0:L2=
0:GOTO580
555 GOSUB560:GOTO570
560 O$=RIGHT$(STR$(INT((N-1)/12)
+2),1):Z=N-12*INT((N-1)/12):N$=N
$(Z):N$=O$+N$:RETURN
570 L1=PEEK(S+3*P+1):L2=PEEK(S+3
*P+2)
580 PRINT@X,P:PRINT@X+5,N$:IFN=1
00THEN600
590 PRINT@X+10,RIGHT$(STR$(L1),2
):L2
600 P=P+1
610 NEXT X
620 'EDIT MENU
630 PRINT@17,"1.PLAY SEGMENT":PR
INT@51,"TO CURSOR":PRINT@81,"2.P
LAY START":PRINT@115,"TO CURSOR"
:PRINT@145,"3.PLAY CURSOR":PRINT
@179,"TO END":PRINT@209,"4.NEW S
EGMENT":PRINT@241,"5.CHANGE NOTE
":PRINT@273,"6.INSERT NOTE":PRIN
T@305,"7.DELETE NOTE"
640 PRINT@337,"8.MAIN MENU":PRIN
T@369,"9.QUICK CHANGE":PRINT@403
,"TEMPO BYTES"
650 X=4:P=FS:CP=0
660 PRINT@X,CHR$(128);
670 A$=INKEY$:IFA$=""THEN670
680 IFA$=CHR$(94)OR A$=CHR$(10)T
HEN720
690 IFVAL(A$)<10R VAL(A$)>9 THEN
670
700 ONVAL(A$)GOTO790,800,810,820
,845,948,980,710,1000
710 GOTO50
720 IFA$=CHR$(10)THEN740
730 IFX=4THEN670ELSEPRINT@X," ";
X=X-32:CP=CP-1:GOTO660
740 IFX=4520R FS+CP+1=LP THEN670
ELSEPRINT@X," ";X=X+32:CP=CP+1:
GOTO660
750 M1=PEEK(&H330D):M2=PEEK(&H33
0E):RETURN
760 A1=INT((S+3*FS)/256):A2=(S+3
*FS)-A1*256:RETURN
770 A1=INT((S+3*FS+3*(CP+1))/256
):A2=(S+3*FS+3*(CP+1))-A1*256:RE
TURN
780 EXEC&H325A:FORZ=1 TO300:NEXT
Z:PRINT@X,CHR$(255);FORZ=1TO255
STEP20:SOUNDZ,1:NEXT:POKE&H3262,
&H33:POKE&H3263,&H4C:POKE&H330D,
M1:POKE&H330E,M2:GOTO660
790 GOSUB750:GOSUB760:POKE&H3262
,A1:POKE&H3263,A2:GOSUB770:POKE&
H330D,A1:POKE&H330E,A2:GOTO780
800 GOSUB750:GOSUB770:POKE&H330D
,A1:POKE&H330E,A2:GOTO780
810 GOSUB750:GOSUB770:Z=256*A1+A
2:Z=Z-3:A1=INT(Z/256):A2=Z-A1*25
6:POKE&H3262,A1:POKE&H3263,A2:GO
TO 780
820 CLS:PRINT"PLEASE TYPE POSITI
ON NUMBER FOR NEW SEGMENT":INPUT
Q
840 FS=Q:P=Q:GOTO520
845 GOSUB850:GOTO520
850 FOR Z=16TO464STEP32:PRINT@Z,
STRING$(15," "):NEXTZ:Z=S+3*FS+3
*CP:PRINT@17,"ENTER NEW NOTE":PR
INT@49,"ENTER TO EXIT":PRINT@81,
"";INPUTN$
860 IFN$=""THENP=FS:RETURN
865 IF N$="ENV"THENPOKEZ,100:POK
EZ+1,0:POKEZ+2,0:RETURN
875 IF N$="PAU"THENPOKEZ,255:GOT
O930
890 PRINT@113,"OCTAVE?":PRINT@14
5,"";INPUTO:IFO<20RO>4THEN890EL
SEFORX=1 TO12:IFN$(X)THENNEX
TELSEN=(O-2)*12+X:POKEZ,N:GOTO93
0
892 IF N$="PAU"THENPOKEZ,255:GOT
O930
894 IF N$="ENV"THENPOKEZ,100:GOT
O930
900 GOTO845
930 PRINT@177,"NOTE LEN(1-32)":P
RINT@209,"";INPUTN:IF N>32OR N<
1 THEN 930
935 POKEZ+1,N
940 PRINT@241,"PAUSE? (1-255)":P
RINT@273,"";INPUTN:IFN<10RN>255
THEN940
945 POKEZ+2,N:P=FS
947 RETURN
948 M=M+3:GOSUB950:GOTO960
950 PRINT@433,"thinking":M1=INT(
M/256):M2=M-M1*256:POKE&H330D,M1
:POKE&H330E,M2:RETURN
960 FOR Z=M TOS+3*FS+3*CP+1STEP-
1:A=PEEK(Z-1):POKEZ+2,A:NEXT
970 GOSUB850:LP=(M-S)/3:GOTO520
980 M=M-3:GOSUB950:FORZ=S+3*FS+3
*CP TOM+2:A=PEEK(Z+3):POKEZ,A:NE
XT:LP=(M-S)/3:GOTO520
1000 :CLS:PRINT"THIS OPTION CHAN
GES ALL NOTE LENGHTS AND/OR P
AUASE BYTES FROM BEGINING OF
SEGMENT TO THE CURSOR":PRINT:PR
INT"ENTER NOTE LENGTH (1-32)
<ENTER> TO LEAVE UNCHANGED"
:INPUTA:IFA=0THEN1030ELSEIFA<10R
A>32THEN1000
1010 FORZ=S+3*FS TOS+3*FS+3*CP S
TEP3:POKEZ+1,A:NEXT
1030 PRINT:PRINT"ENTER NOTE PAUS
E <ENTER> TO ESCAPE":INPUTA:
IF A=0THEN520ELSEIFA<10R A>255TH
EN1030
1040 FORZ=S+3*FS TOS+3*FS+3*CP S
TEP3:POKEZ+2,A:NEXT:GOTO520
8000 'PLAY STATEMENTS
8002 L=0:I=0:A=&H334C:E=PEEK(&H3
30D)*256+PEEK(&H330E):GOSUB510
8010 D=1:CLS:PRINT"PREPARE TAPE
RECORDER OR DISK TO RECEIVE PL
AY STATEMENTS":PRINT"INPUT"FILE
AME";F$=INPUT"(C)ASSETTE OR (D)I
SK";D$=IFD$="C"THEN"OPEN"O",-1,F
$:D=-1:GOTO8012
8011 F$=F$+"BAS":OPEN"O",#1,F$
8012 B$="5 PLAY"+CHR$(34)+"T6"+C
HR$(34):PRINT#D,B$
8014 L=L+10:L$=STR$(L):L$=RIGHT$(
L$,LEN(L$)-1):A$=L$+" PLAY"+CHR
$(34)
8020 FORX=A TOA+27STEP3:IFX=E TH
EN8100
8030 N=PEEK(X):IFN=255THEN8040EL
SEIFN=100THEN8160ELSEGOSUB560:N$
="O"+N$:IFRIGHT$(N$,1)="#"THENN$
=LEFT$(N$,LEN(N$)-1):N$=N$+"#"
8040 L1=PEEK(X+1):L2=PEEK(X+2):I
FL1=0THENL1=1:IFL2=0THENL2=1
8050 IF L1>16ANDL1<20THENL1=16EL
SE IF L1>19ANDL1<28THENL1=3:GOTO
8060
8055 L1=INT(1/(L1/32))
8060 IF L2>128ANDL2<160 THEN L2=
127 ELSEIFL2>127ANDL2<224 THEN L
2=3:GOTO8070
8065 L2=INT(1/(L2/255))
8070 L1$=STR$(L1):L2$=STR$(L2):L
1$=RIGHT$(L1$,LEN(L1$)-1):L2$=RI
GHT$(L2$,LEN(L2$)-1)
8075 IF N=255THEN8180
8080 N$="L"+L1$+N$+"P"+L2$
8090 A$=A$+N$:NEXTX
8100 A$=A$+CHR$(34):PRINT#D,A$:P
RINTA$
8114 IFX=E THENCLOSE#D:GOTO50
8120 A=A+30:GOTO8014
8160 IFA=E THEN50ELSENEXTX
8180 N$="P"+L1$+"P"+L2$:GOTO8090
20000 PCLEAR1:GOTO10

```

Listing 3: PIANO

3999	8D	11	99199	ORG	99999
3999	8D	11	99299	START	BSR INITLZ
3992	7F	3178	99219	CLZ	ENVTOG
3995	8E	3219	99599	LDX	ENVVTAL
3998	8F	3176	99699	STX	ENVTAB
3998	39	88 1F	99799	LEAX	+31,X
3998	8F	317A	99899	STX	ENDTAB
3911	29	18	99999	BRA	INKEY
3913	86	FF91	91999	INITLZ	LDA \$FF91 INITIALIZE PIA'S
3916	84	87	91199	ANDA	#8F7

3918	87	FF91	91299	STA	\$FF91
3918	86	FF93	91399	LDA	\$FF93
391E	84	87	91499	ANDA	#8F7
3929	87	FF93	91599	STA	\$FF93
3923	86	FF23	91699	LDA	\$FF23
3926	8A	98	91799	ORA	#8
3928	87	FF23	91899	STA	\$FF23
3928	1A	59	91999	ORCC	#859
392D	39		92999	RTS	
392E	1C	FE	92199	INKEY	ANDCC #8FE KEYBOARD POLLING
3939	8E	3176	92299	LDX	ENVTAB
3933	8F	3173	92399	STX	ENVPTR
3936	FE	317C	92499	LDU	ENVLDY





## NEW COMMANDS, NO BUGS

## MUSIC PLUS



by Bob Ludlum

Since MUSIC+ appeared in Rainbow ("Making Four-Part Harmony Easier") I've received a large response from MUSIC+ users with questions and requests for additional features. As a result, I've fixed a minor bug and added two new commands to the program.

If you're not familiar with MUSIC+, it's an enhanced version of Larry Konecky's CoCo Composing. It is a BASIC program that loads a machine language music synthesis program. A screen editor facilitates the entry, editing and playing of four-part music. It requires a 32K Color Computer with Extended Color Basic and runs without modification on both tape and disk systems.

I want to answer some questions I received repeatedly. First, is it possible to add more voices and octaves? Yes, relatively simple modifications to the editor and synthesis programs are all that is required, but a tradeoff exists between the added complexity and the quality of the sounds produced.

MUSIC+ synthesizes the music wave form by summing the contributions from the four voices at equally spaced time intervals. The result is a sampled approximation of the desired wave form. The accuracy of the approximation depends on how often the samples are calculated (the sampling rate). The theoretical minimum rate required is two samples per cycle of the highest frequency component in the wave form. In practice, much higher rates are needed.

If the sampling rate is too low, unwanted frequency components appear in the wave form, a phenomenon known as "aliasing". MUSIC+ calculates a new wave form sample every 145 microseconds (6,896 samples per second), which is already marginal.

The second question frequently asked was, "Why do I get a 'C' note when I enter a 'B' sharp and why do the notes jump from 'B' in one octave to 'C' in the next?"

The note table in MUSIC+ implements the equally tempered chromatic scale with a standard pitch of 440 Hertz (cycles per second) for 'A' in the

fourth octave. Each octave begins with the note 'C' and is made up of 12 pitch intervals (half-steps). There is one half-step between 'B' and 'C', between 'E' and 'F'. There are two half-steps between the rest of the notes with the sharps falling on the half-steps between.

For example, beginning with the third octave, the notes are C3, C3#, D3, D3#, E3, F3, F3#, G3, G3#, A3, A3#, B3, C4, C4# etc. To raise a note one half-step, its pitch is multiplied by the twelfth root of two (approximately 1.0595).

The original MUSIC+ program had a bug that showed up when the music was saved following use of the (M)ove command. The (M)ove command changes the point (actually a branch instruction offset) to the start of the music data allowing a portion of a song to be played. Moving and then saving caused the wrong start location to be saved and the entire song would not play when executed. Playing before saving prevents the problem. Adding POKE A9,0:POKE A9+1,128: to the original program fixes the bug.

The first of the new commands is (H)dcopy, which is used to dump the music data between specified note columns to a printer. The command simply lists each column number followed by the note length and the four note names for that column. The POKE150,18 in Line 9600 sets the baud rate to 2400. Change it to match your printer, if necessary.

Turn the printer off when playing music. The synthesis program generates a byte (eight bits) to the output port that drives the CoCo's six-bit digital to analog converter. One of the lower order bits appears on the serial port while music is playing and will cause your printer to do strange things!

The other new command is (W)form. It allows changing the waveform table to produce sounds with different timbres. The program prompts for the percentages of the fundamental and the first four overtones of the music wave form. The 256 values for the new wave form table are calculated (in BASIC) by summing the scaled sinusoidal fundamental and the second through the fifth

harmonics.

The new wave form table is in effect for played and saved music until MUSIC+ restores the table after RUN. The original MUSIC+ organ wave form has 50 percent fundamental and 25 percent each for the second and third harmonics. The sum of the percentages should equal 100.

The machine language program is located immediately above the BASIC screen editor in order to maximize the amount of memory available for holding music data. Adding the new commands require either relocating the machine language program (which would have destroyed compatibility with existing MUSIC+ music files) or shrinking the BASIC program.

I decided on the latter and removed the unnecessary spaces and packed the lines. Unfortunately, doing so makes describing the necessary steps to update the original MUSIC+

program very difficult. I'll be happy to make a copy of the latest version of the program if you send me a tape or (preferably) a disk in a self-addressed, stamped return mailer. My mailing address is 226 Pine Ridge Drive, Panama City, FL 32405.

I've been very pleased with the positive responses to MUSIC+ and hope the new commands will be useful. I'm especially grateful to all who were kind enough to send me some outstanding samples of their music. I encourage you to share your efforts with the readers of RAINBOW.

Editor's Note: Due to the length of the new and improved MUSIC+ program, we're unable to print the listing in The RAINBOW. We will, however, include the modified MUSIC+ program on this month's Rainbow On Tape or Disk.

The following contributors have sent us their compositions using the original Music+ program. We have dumped the first portion of the music data from each song using Music+'s (H)dcopy command and have printed it for your enjoyment. Both songs will be provided in their entirety on this month's RAINBOW ON TAPE, immediately following the Music+ program listing. Simply CLOADM and EXEC to play each song.

**Scott Joplin's "The Entertainer"**  
By Bill Kast

39: 16	,E6	,G5	,E5	,G2	37: 16	,D5	,A4	,D4	,F3#
40: 16	,C6	,E5	,C5	,G2	38: 8	,D5	,B4	,G4	,G3
41: 16	,D6	,F5	,C4	,G3	39: 16	,D5	,A4	,F4#	,D3
42: 16	,E6	,G5	,C4	,G3	40: 16	,D5	,A4	,F4#	,D3
43: 16	,E6	,G5	,E5	,G2	41: 8	,D5	,B4	,G4	,D3
44: 16	,B5	,D5	,B4	,G2	42: 16	,D5	,A4	,D4	,F3#
45: 8	,D6	,F5	,B3	,G3	43: 16	,D5	,A4	,D4	,F3#
46: 8	,C6	,E5	,C5	,C3	44: 8	,D5	,B4	,G4	,G3
47: 8	,C6	,E5	,C4	,G3	45: 16	,D5	,A4	,F4#	,D3
48: 8	,C6	,E5	,C4	,G3	46: 16	,D5	,A4	,F4#	,D3
					47: 8	,D5	,B4	,G4	,D4
					48: 8	,D5	,A4	,D4	,F3#
					49: 8	,C5#	,G4	,E4	,E3
					50: 8	,D5	,F4#	,A3	,D3
					51: 8	,D5	,E4	,A3	,A3
					52: 8	,C5#	,E4	,A3	,A2
					53: 8	,D5	,F4#	,A3	,D3
					54: 8	,A5	,D5	,A4	,F3#
					55: 8	,G5	,C5#	,A4	,E3
					56: 8	,F5#	,D5	,A4	,D3
					57: 4.	,E5	,A4	,C4#	,A3
					58: 8	,A4	,A4	,E4	,A4#
					59: 8	,F5#	,A4	,D4	,D4
					60: 8	,E5	,A4	,C4#	,A3
					61: 8	,D5	,A4	,C4#	,A3
					62: 8	,D5	,A4	,C4#	,A3
					63: 4.	,E5	,A4	,C4#	,A3
					64: 8	,A4	,A4	,E4	,C4#
					65: 8	,F5#	,A4	,D4	,D4
					66: 8	,E5	,A4	,C4#	,A3
					67: 8	,D5	,A4	,C4#	,A3
					68: 16	,E5	,A4	,E4	,C4#
					69: 16	,E5	,A4	,E4	,C4#
					70: 8	,F5#	,A4	,D4	,D4
					71: 16	,E5	,A4	,C4#	,A3
					72: 16	,E5	,A4	,C4#	,A3
					73: 8	,D5	,A4	,C4#	,A3
					74: 16	,E5	,A4	,E4	,C4#
					75: 16	,E5	,A4	,E4	,C4#
					76: 8	,F5#	,A4	,D4	,D4
					77: 16	,E5	,A4	,C4#	,A3
					78: 16	,E5	,A4	,C4#	,A3
					79: 8	,D5	,A4	,C4#	,A3
					80: 8	,E5	,A4	,E4	,C4#
					81: 8	,F5#	,A4	,D4	,D4
					82: 8	,E5	,A4	,E4	,C4#
					83: 8	,D5	,A4	,F4#	,B3
					84: 8	,D5	,G4#	,D4	,B3
					85: 8	,C5#	,A4	,E4	,A3
					86: 16	,D5	,A4	,C4#	,E3
					87: 16	,D5	,A4	,C4#	,E3
					88: 8	,D5	,A4	,D4	,F3#
					89: 8	,D5	,A4	,C4#	,E3

**Handel's "Hallelujah Chorus"**  
By Dave Greenfield

The listing: HALELUJA

COL: LEN	,V1	,V2	,V3	,V4
1: 8	,D5	,A4	,F4#	,D3
2: 8	,D5	,G4	,D3	,E3
3: 8	,D5	,A4	,D3	,F3#
4: 8	,A5	,F5#	,D3	,D3
5: 8	,B5	,G5	,D5	,G3
6: 8	,A5	,F5#	,D5	,D3
7: 8	,D5	,A4	,F4#	,D3
8: 8	,D5	,A4	,F4#	,D3
9: 8	,D6	,A5	,F5#	,D3
10: 8	,D6	,A5	,F5#	,E3
11: 8	,F5#	,D5	,A4	,D3
12: 8	,G5	,C5#	,G4	,E3
13: 8	,F5#	,D5	,A4	,D3
14: 8	,D5	,A4	,F4#	,D3
15: 8	,D5	,A4	,F4#	,D3
16: 8	,A5	,D5	,A4	,F3#
17: 8	,G5	,C5#	,A4	,E3
18: 8	,F5#	,D5	,A4	,D3
19: 8	,E5	,D5	,A4	,A3
20: 8	,E5	,C5#	,G4	,A2
21: 8	,D5	,A4	,F4#	,D3
22: 8	,A4	,D4	,F4#	,D3
23: 8	,B4	,D4	,G4	,D3
24: 8	,C5#	,D4	,E4	,D3
25: 4.	,D5	,A4	,F4#	,D3
26: 8	,A4	,A4	,D4	,F3#
27: 8	,B4	,G4	,D4	,G3
28: 8	,A4	,F4#	,D4	,D3
29: 8	,D5	,A4	,F4#	,D3
30: 8	,D5	,A4	,F4#	,D3
31: 4.	,D5	,A4	,F4#	,D3
32: 8	,A4	,A4	,D4	,F3#
33: 8	,B4	,G4	,D4	,G3
34: 8	,A4	,F4#	,D4	,D3
35: 8	,D5	,A4	,F4#	,D3
36: 16	,D5	,A4	,D4	,F3#

The listing: ENTERTAIN

COL: LEN	,V1	,V2	,V3	,V4
1: 16	,D6	,D5	,D4	,D3
2: 16	,E6	,E5	,E4	,E3
3: 16	,C6	,C5	,C4	,C3
4: 8	,A5	,A4	,A3	,A2
5: 16	,B5	,B4	,B3	,B2
6: 8	,G5	,G4	,G3	,G2
7: 16	,D5	,D4	,D3	,D2
8: 16	,E5	,E4	,E3	,E2
9: 16	,C5	,C4	,C3	,C2
10: 8	,A4	,A3	,A2	,A1
11: 16	,B4	,B3	,B2	,B1
12: 8	,G4	,G3	,G2	,G1
13: 16	,D4	,D3	,D2	,D1
14: 16	,E4	,E3	,E2	,E1
15: 16	,C4	,C3	,C2	,C1
16: 8	,A3	,A2	,A1	,A0
17: 16	,B3	,B2	,B1	,B0
18: 16	,A3	,A2	,A1	,A0
19: 16	,G3#	,G2#	,G1#	,G0#
20: 8	,G3	,G2	,G1	,G0
21: 8	,D3	,D2	,D1	,D0
22: 8	,G5	,D5	,B4	,G2
23: 16	,D4	,B3	,G3	,D2
24: 16	,D4#	,B3	,G3	,D2
25: 16	,E4	,C3	,D3	,D2
26: 16	,C5	,C3	,D3	,D2
27: 16	,C5	,C4	,G3	,E3
28: 16	,E4	,C4	,G3	,E3
29: 16	,C5	,G3	,G2	,D2
30: 16	,C5	,G3	,G2	,D2
31: 16	,E4	,C4	,A3#	,G3
32: 16	,C5	,C4	,A3#	,G3
33: 8	,C5	,F3	,F2	,D2
34: 8	,C5	,C4	,A3	,D2
35: 16	,C5	,D4	,D3	,E3
36: 16	,C6	,E5	,C5	,E3
37: 16	,D6	,F5	,C4	,G3
38: 16	,D6#	,F5#	,C4	,G3

# Teaching Language Idioms

By Steve Blyn

This month's program is a playful one designed mainly for those in the middle grades. It's good for all of us to occasionally take a break from more serious educational programs. This program points out idioms, one of the peculiarities of our language. We are going to have fun with some idioms that refer to bodily figures of speech. We have included such expressions as "crossed fingers," "toe the mark" and "nose to the grindstone."

If your students are motivated to discover the derivation of these idioms, then we have accomplished even more than we set out to do. We'll demonstrate how to add to the list to make it more comprehensive. Our main purpose, though, is enjoyment. We intend to show students that the computer can easily produce fun and educational programs.

While testing the program with middle school students, we found that a great source of amusement was the errors made — some deliberately. Even after the students learned the idioms, they had a lot of fun entering answers other than the correct ones. "With tongue in cheek" became "with toe in cheek." "Feet of clay" became "nose of

clay" and so forth. This experiment inspired a jovial atmosphere — laughing girls and boys, chuckling teachers, animated discussion, thinking out loud — it was delightful.

Lines 40 and 50 set the dimensions at 15 questions and answers. 'N' was set at 15 simply because we ran out of body-part idioms. If you can think of others, add more DATA lines and adjust the number on Line 40 accordingly. Lines 60-80 read these questions and answers from the DATA lines.

Line 100 chooses a random question and answer (variable 'R'). The program gives six answers from which to choose. The variable 'J' in Line 110 subtracts a number between one and five from the correct answer. The six answers printed start at the true number (R) less 'J' and include five more choices. The true answer is ensured a place among the six listed. Lines 170-190 print out the choices.

The only problem is the 'J' variable may fall below number one, or the 'J'-plus-five amount may exceed the 15 listed answers. If these situations occur, we encounter several BS errors. This indicates there is no such string. To prevent these problems, we set further

restrictions on the 'J' values in lines 120 and 130.

Line 200 asks the question and Line 210 waits for the answer. Lines 220 and 230 evaluate whether the answer is correct. Line 240 prints the correct answer if the student gives an incorrect response. Lines 250-270 wait for the user to press the ENTER key to continue. If 'E' is pressed, the program ends.

We assumed that players would soon master this program completely since there are only 15 questions. Therefore, we did not include a scorecard. If it is needed, you could display the score at the bottom of the screen at all times. We included an extra variable (CR) on Line 220 to count the correct answers.

We hope your child or student enjoys learning these idioms. Perhaps you or they will be creative and produce a similar program with other idioms. Colors would be a good possibility, using questions such as "-- as a beet" or "-- with envy" or "feeling sad and --." We here at Computer Island always enjoy hearing from readers about their experiences with the programs in this column. □

## The listing: IDIOMS

```

10 REM"UNUSUAL USE OF OUR LANGUA
GE"
20 REM"STEVE BLYN,COMPUTER ISLAN
D,NY,1986
30 Z$=STRING$(32,255)
40 N=15
50 DIM A$(N),B$(N)
60 FOR T= 1TO N
70 READ A$(T),B$(T)
80 NEXT T
90 CLS:PRINT"      OUR STRANGE LA
NGUAGE"
100 R=RND(N)
110 J=R-RND(5)
120 IF J<1 THEN J=1
130 IF J>10 THEN J=10
140 PRINT@32,Z$
150 PRINT@288,Z$;
160 PRINT@320,"";
170 FOR T=J TO J+5

```

```

180 PRINTB$(T),
190 NEXT T
200 PRINT@64,A$(R):PRINT:PRINT"N
AME THE BODY PART - ";
210 LINEINPUT G$
220 IF G$=B$(R) THEN PLAY"O3L100
CCDCDEFFGGGG":CR=CR+1:GOTO 250
230 IF G$<>B$(R) THEN PLAY"O1L10
0FFF"
240 PRINT:PRINT"      THE ANSWER I
S "B$(R)
250 PRINT:PRINT"      PRESS ENTER
TO GO ON";
260 EN$=INKEY$
270 IF EN$=CHR$(13) THEN 90 ELSE
IF EN$="E" THEN END
280 GOTO 260
290 DATA ----- OF CONTENTION.,B
ONE
300 DATA SPLITTING -----,HAIRS
310 DATA ----- TO THE WHEEL.,SH

```

```

OULDER
320 DATA WITH ----- IN CHEEK.,T
ONGUE
330 DATA TURN THE OTHER -----,
CHEEK
340 DATA WITH ----- CROSSED.,FI
NGERS
350 DATA ----- THE MARK.,TOE
360 DATA ARMED TO THE -----,TE
ETH
370 DATA ----- GREASE.,ELBOW
380 DATA ----- IN GLOVE.,HAND
390 DATA STAB IN THE -----,BAC
K
400 DATA ----- TO THE GRINDSTON
E.,NOSE
410 DATA DON'T STICK YOUR -----
OUT.,NECK
420 DATA IN ONE ----- AND OUT TH
E OTHER.,EAR
430 DATA ----- OF CLAY.,FEET

```



# Robots: Their Place in Education

By Michael Plog, Ph.D.

A young woman was spending a rainy summer vacation with a group of artistic people, including some major literary "names." The group, restricted to indoor activities, told eerie stories for amusement. One of the venerable members of the group suggested that everyone write a ghost story. The young woman, named Mary Godwin, wrote a horror story based on a dream she had a few nights after the suggestion. She later married one of the members of that group, Percy Bysshe Shelley. Whether or not you have heard of Percy or Mary, you certainly know Mary's horror story, *Frankenstein* (or, the *Modern Prometheus*).

Mary Shelley's book became a prototype for horror stories, particularly those concerning robots. Mary had never heard the term "robot." Her book was written in 1818 and the word robot came into being in a 1920 play by Karel Capek titled *R.U.R.* The word robot comes from the Czech word "robotnik," for worker or serf. Capek wrote about people creating mechanical beings to do work for humans.

Science fiction writer Isaac Asimov has been called "the father of robotics" because of the many stories he has written about the mechanical creatures. And for another important reason — Asimov's robots are not creatures who turn on their creators (like Frankenstein), but are manufactured by engineers to fit exacting specifications. The most important of these specifications is that robots may not harm human beings. Their circuits simply do not allow such an action. Thus, modern robot stories eliminate the fear (the Frankenstein complex) people have about mechanical intelligence.

Modern robots are industrial automations that perform a series of steps to complete a task. Robots are not yet made in the general shape of humans

and have extremely limited intelligence. My daily life seldom brings me in touch with industrial robots, but I have a few contacts with other types. For example, I interact with a robot when making long distance telephone calls. I simply dial an 800 number, enter my access number, then dial the number I want to reach. This all takes place with the aid of robots.

Besides industrial robots, there are robot "toys" for the home. Some of these machines are built in a similar fashion to the *Star Wars* robots and can perform a variety of tasks. The home robots, as well as industrial robots, need to be programmed. Indeed, a robot has a computer "brain" to allow human programming.

Since robots use computers, and may be considered as a subcomponent of the field of computer science, it is only natural that robots function in schools as well as factories and homes. Generally, they are used in computer classes and electronics courses. Students learn about robots by building the mechanical workers and programming them to perform a task.

When computers first appeared in schools, educational leaders wondered and debated about their use. In the beginning, they were used in classrooms to teach *about* computers. More recently, myriad uses have been made of computers in schools. Students use them for a multitude of purposes other than learning BASIC programming. In fact, students can be declared computer literate without ever knowing about binary addition and subtraction. Computers are being used more and more as learning tools in classrooms. Students use word processing packages to write reports, database programs are used to examine information from science experiments, and the list goes on.

The educational community has

spent over a decade debating the computer's role in elementary and secondary education. The debate continues even today, although most educational professionals consider the computer to be an additional (and very important) tool for students in the classroom, with a wide variety of purposes. There is no reason to expect the debate over the educational role of robots to be any less active than the debate over computer uses. What are appropriate activities for robots in the school? Should students simply learn about them, then consider the utility of robotics finished? Should robots be used as another tool for students, in the same way computers are an educational tool?

Despite similarities, robots and computers are not the same thing. Computers tend to be more oriented toward mental activities. Robots, on the other hand, tend to interact with the physical environment in a much more direct way than computers. For example, consider students working with a word processing package. The actual printing of a page is a physical activity, but is much less important than the mental activity of the student creating the document.

When you sit down in front of the Color Computer to write a program, most of the activity is the relationship between your mind and the screen. Not so with a robot. The observable activity of a robot performing a task deals with physical objects. Screws are tightened, materials are moved from one place to another, objects are assembled, and so on. The programming of a robot may involve the same mental activity as the programming of the Color Computer, but the end result differs.

continued on Page 26

# Castle of Doom

By Scott Halfman

You are traveling in a land far away, seeking fortune. You gain loot by passing from castle to castle, each time picking up all the objects on the different levels (floors) of the castle. After cleaning out each castle, you move on to a new one, the difficulty increases accordingly.

Castle requires 32K Extended Color Basic, however, it will run in 16K ECB with the disk controller unplugged. Now load in castle and type RUN. The title screen appears. To continue press the fire button on the right joystick. The castle door opens and you enter the castle. A skill level prompt appears. Type a number between one and four (1-easy, 4-hard). There is a short pause, and then the computer reveals what object are to be picked up on that level. Push the fire button to play.

The game board appears. All the objects are laid on the castle floor, the bonus score and the number of men you have are displayed at the top of the screen. Your man is then lowered onto the game board.

To clear the board, you must pick up all the keys (or other objects) before the bonus countdown runs out, without falling off the path.

When the board is cleared, move to the (white) elevator platform. You are then lifted off that castle level to the next.

After clearing level six, your man makes his way to the castle exit, where you are either prompted to either quit or go on. The number of castles finished, your score and the high score are displayed. Press 'Q' to quit or 'C' to continue.

If you dare to continue, your man leaves the castle and a new castle approaches.

## The Listing: CASTLE

```
10 CLEAR130:POKE65495,0:DIMG(3,1
),A$(21),B$(21):GOSUB8000:GOSUB7
140:PMODE3,1
15 BO=0:T1=9:T2=9:T3=9:GU=4:A=20
0:RESTORE:PLAY"ABCDEFGH":CLS:PRIN
T@6,,:INPUT"SKILL LEVEL(1-4)";SK
:I=SK*.2:IFSK<1ORSK>4THEN15 ELSE
L=17-SK*.5:GOTO45
20 PMODE3,1:PCLS:DRAW"BM14,185C3
R205E20L20E20L20E30L10H10E20H10E
10H10E10U10H10L10E10L100G20L10E2
0L40G10L10E10L40G20D10F10G10R30G
10L10G10D20R10G20R30F10R10G10L20
G20F10G10R10BM55,120R40E10L10E10
L10G10L10E10L10G20"
25 DRAW"BM34,175R100E10L90E30L10
G20L20G10F10BM154,175E10R30E10R1
0G15R10G5L50BM84,155R70E20L10G10
L50E20L10G30BM134,135R10E10L10G1
0"
30 DRAW"BM34,115R10E40L30G10L10D
10R10D10G10BM64,65R40G10R10E20L2
0E10L10G10L20G10BM84,45E20L5G20R
5BM69,35L10L15E10L5G10D10F10R10E
5L5E15L5"
35 DRAW"BM89,85G5R10E5L10BM147,2
5R5G20L5E20BM114,85R40G20R10G10L
15E10L10E10L5H10BM124,75R20E10L1
0E20R10E10L25G10R5G30BM154,115E1
5L10E15R10G5R35E5R10G20F15G10L10
E10H15L20G10L15"
40 DRAW"BM180,115R10G5R10G5L20E1
```

```
0BM189,140R10G5R5G10L15E15":PAI
N
T(38,22),3,3:DRAW"BM199,60C4R5U6
0D60G5L5U65D65E5":PAINT(201,61),
4,4:LINE(190,3)-(210,9),PRESET,B
F:RETURN
45 CR$(1)="C2R3D3L4U3R9D2L2U2":C
R$(2)="C2U4R4D6L4U4R3D4L1U6R2D2C
3U1":CR$(3)="C3R3U2C2R2U1D1R1D2R
1L1D1L2D1U1L1U2L1R1U1":CR$(4)="C
3R2U2C2D2F2E2U2L2D6R2L4":CR$(5)=
"C3R2U2C2R6L2D4R2L6R2U4R2D4":CR$(
6)="C3R2U4C2F4G4H4E4D2R2L4R2D1L
2R4L2D1R4L8R4D2L2R4L2"
50 DATA 36,96,162,224,36,104,120
,104,80,28,24,64,176,104,160,204
,216,184,132,160
55 DATA 24,20,24,40,56,40,48,80,
128,100,180,180,180,180,128,140,
104,110,140,80
60 FORX=1TO20:READA$(X):NEXTX:FO
RX=1TO20:READB$(X):NEXTX
70 PR$(1)="PICK UP ALL KEYS":PR$(
2)="LOCK ALL DOORS":PR$(3)="PIC
K UP ALL RINGS":PR$(4)="PICK UP
ALL CUPS":PR$(5)="PICK UP ALL SC
ROLLS":PR$(6)="PICK UP ALL DIAMO
NDS"
80 N$(1)="C1R2C3G2E2D6R2L4":N$(2
)=="R4D3L4D3R4":N$(3)="R4D3L3R3D3
L4":N$(4)="D3R4U3D6":N$(5)="R4L4
D3R4D3L4":N$(6)="D6R4U3L4":N$(7)
=="R4D2G4":N$(8)="R4D3L4U3D6R4U3"
:N$(9)="D3R4U3L4R4D6":N$(0)="R4D
6L4U6":GOSUB5010
```

```
100 PLAY":A=A+(JOYSTK(0)-32)/L
:B=B+(JOYSTK(1)-32)/L:PUT(A-1,B-
3)-(A+1,B+3),G,NOT
105 T3=T3-1:ON T3+2 GOSUB200:LIN
E(130,3)-(134,9),PRESET,BF:DRAW"
BM130,3"+N$(T3)
110 PUT(A-1,B-3)-(A+1,B+3),G,NOT
:ONPOINT(A,B+3)GOTO1000,2000,10
0,3000
200 T3=9:T2=T2-1:ONT2+2GOTO210:L
INE(120,3)-(124,9),PRESET,BF:DRA
W"BM120,3"+N$(T2):RETURN
210 T2=9:T1=T1-1:LINE(110,3)-(12
4,9),PRESET,BF:ONT1+2GOTO220:DRA
W"C3BM110,3"+N$(T1)+"BM120,3"+N$(
T2):RETURN
220 T1=5:T2=9:T3=9:FORX=1TO3:PLA
Y"L100;1;2;3;4;5;6;7;8;9;10;11;1
2":NEXTX:PUT(A-1,B-3)-(A+1,B+3),
G,NOT:GOTO1000
1000 IN=3:FORX=B TO191STEP2:X=X+
IN:PUT(A-1,X-3)-(A+1,X+3),G,NOT:
IN=IN+1
1010 PLAY"AV"+STR$(INT(31-(X/8)))
:PUT(A-1,X-3)-(A+1,X+3),G,NOT:N
EXT:ONGU GOTO 6000:GU=GU-1:PUT(G
U*10-1,3)-(GU*10+1,6),G,NOT
1020 B=10:PLAY"V31":IN=0:FORA=2T
O199STEP2:IFA<139THENB=B+1
1025 PUT(A-1,B-3)-(A+1,B+3),G,NO
T:PLAYSTR$(INT(B/4)):PUT(A-1,B-3
)-(A+1,B+3),G,NOT:NEXTA
1030 PUT(A-1,B-3)-(A+1,B+3),G,NO
T:PLAY"O3V15L255":FORX=1TO10:PLA
```

```

YSTRS(RND(12)):PAINT(A,B),1,3:PA
INT(A,B),4,3:PAINT(A,B),2,3:NEXT
X:PAINT(A,B),3,3
1040 DRAW"BM110,3;" +N$(T1)+"BM12
0,3" +N$(T2)+"BM130,3" +N$(T3):GOT
O100
2000 CR=CR-1:PAINT(A,B+3),3,3:GO
TO2010
2005 Y=LEN(SC$):FORX=2 TO Y:LINE
(156+10*X,3)-(166+10*X,9),PRESET
,BF:DRAW"BM"+STR$(156+10*X)+",3"
+N$(VAL(MID$(SC$,X,1))):NEXTX:RE
TURN2010 PLAY"L255ABC":SC$=STR$(
VAL(SC$)+10):GOSUB2005:ONCR GOTO
2030
2020 GOTO 100
2030 PUT(A-1,B-3)-(A+1,B+3),G,NO
T:FORV=1TO30STEP6:PLAY"V"+STR$(V
):FORN=1TO12:PLAYSTR$(N):NEXTN,V
:PUT(A-1,B-3)-(A+1,B+3),G,NOT:GO
TO100
3000 IFPOINT(A-1,B+3)=3ORPOINT
(A+1,B+3)=3THENGOTO100ELSEB=63:P
LAY"CDEFGAB":X=B'ELEVATOR ROUTIN
E
3010 A=200:FORE1=1TO5:PLAY"O"+ST
R$(E1):FORE2=1TO12:X=X-1:PUT(A-1
,X-3)-(A+1,X+3),G,NOT:PLAYSTR$(E
2):PUT(A-1,X-3)-(A+1,X+3),G,NOT:
NEXTE2,E1
3015 IF CR>1THENGOSUB5030
3020 PLAY"O3":ON CR GOSUB5000:GO
TO100
5000 IFBO=6THENBO=0:GOSUB7000'dr
aw board routine
5005 SC$=STR$(VAL(SC$)+T1*100+T2
*10+T1):GOSUB2005:T1=1:T2=1:T3=1
:DRAW"C3":GOSUB200:LINE(110,3)-(
134,9),PRESET,BF:FORX=1TO30:PUT(
172,3)-(166+LEN(SC$)*10,9),G,NOT
:PLAY"O1A":NEXTX
5010 T1=9:T2=9:T3=9:PCLS:DRAW"BM
110,3" +N$(9)+"BM120,3" +N$(9)"BM
130,3" +N$(9):FO=BO+1:GU=GU+1:PCL
S:CLS:PRINT@268,"PHASE":BO:PRINT
@288+(32-LEN(PR$(BO)))/2,PR$(BO)
:GOSUB20:IFGU>5THENGU=5
5013 FORX=110TO130STEP10:DRAW"C3
BM"+STR$(X)+",3" +N$(9):NEXTX
5015 DRAW"C4":LINE(172,2)-(245,2
),PSET,B:LINE(172,10)-(245,10),PSE
T:GOSUB2005:PLAY"O1ABCDEF"
5020 PRINT@484,"PRESS FIRE BUTTO
N TO PLAY":IFPEEK(65280)=126 OR
PEEK(65280)=254 THENSCREEN1,0:PO
KE65314,248:CR=21 ELSE GOTO5020
5025 FORX=1TO20:PLAY"O1CDEF":DRA
W"EM"+AS$(X)+", "+B$(X)+CR$(BO):NE
XTX:FORX=10TOGU*10-10STEP10:PUT(
GU*10-1-X,3)-(GU*10-X+1,6),G,NOT
:PLAY"O3CDEF":NEXTX:X=3
5030 FORE1=5TO1STEP-1:PLAY"O"+ST
R$(E1):FORE2=12TO1STEP-1:X=X+1:P
UT(A-1,X-3)-(A+1,X+3),G,NOT:PLAY
STR$(E2):PUT(A-1,X-3)-(A+1,X+3),
G,NOT:NEXTE2,E1:B=X
5040 PLAY"O3L255V15":FORA=199TO2
10:PUT(A-1,X-3)-(A+1,X+3),G,NOT:
PLAY"12":PUT(A-1,X-3)-(A+1,X+3),
G,NOT:NEXTA:DRAW"C3":RETURN
5500 '?score and all that stuff
5510 GOTO 5510
6000 LINE(90,88)-(178,120),PRESE
T,BF

```

```

6010 DRAW"BM100,90C4R4L4D6R4U3L2
BM109,90:D6U3R4U3L4R4D6BM116,90;
D6U6R3D3U3R3D6BM127,90R4L4D3R2L2
D3R4BM140,90R4D6L4U6BM149,90;D4F
2E2U4BM157,90R4L4D3R2L2D3R4BM165
,90;D6U6R2F2G2L2R2F2"
6020 DRAW"BM97,100D6U3R4U3D6BM10
4,100R4L2D6L2R4BM113,100R4L4D6R4
U3L2BM120,100D6U3R4U3D6BM137,100
R4L4D3R4D3L4BM144,100R4L4D6R4BM1
53,100R4D6L4U6BM160,100D6U6R2F2G
2L2R2F2BM169,100R4L4D3R2L2D3R4"
6030 IF VAL(SC$)>VAL(HS$)THEN HS
$=SC$:PLAY"V31CDEFGABBAGFEDCCCC
V15"
6040 Y=LEN(HS$)*10:DE=117-Y/2:LI
NE(127-Y/2,108)-(127+Y/2,118),PR
ESET,BF:FORX=2TOY/10:DRAW"C3BM"+
STR$(DE+X*10)+",110;" +N$(VAL(MID
$(HS$,X,1))):NEXTX
6050 POKE178,RND(255):LINE(90,88
)-(178,120),PSET,B:IFPEEK(65280)
=126ORPEEK(65280)=254THEN6060 EL
SE 6050
6060 GOTO 15
7000 CLS' intermission
7010 PCLS:DRAW"C3BM190,45F10L30G
100F5R100H10R40F30L40H10L130H15E
15R10E100R20F10":PAINT(185,45),3
,3:DRAW"C4BM190,160R6F6L6H6BM190
,175D16U16F6D10U10R6D10BM175,45U
45D45R6F6U51D51L6U51D51H6"
7015 L1=L1+1:L$=STR$(L1):IF L1>9
9THEN L1=1 ELSE IF L1<10THENL$="
"+L$
7020 DRAW"C4S40BM10,10" +N$(VAL(M
ID$(L$,2,1)))+ "BM60,10" +N$(VAL(M
ID$(L$,3,1)))+ "S4":LINE(150,2)-(
245,2),PSET,B:LINE(150,10)-(245,10
),PSET,B:LINE(150,3)-(245,9),PRESE
T,BF:GOSUB2005
7025 SCREEN1,1:POKE65314,248:O=5
:P=8:FORX=5TO43:PUT(180,X-3)-(18
2,X+3),G,NOT:P=P-1:IF P=0 THEN P
=8:O=O-1
7026 PLAY"L255O"+STR$(O)+", "+STR
$(P):PUT(180,X-3)-(182,X+3),G,NO
T:NEXTX
7030 PRINT"YOUR SCORE ";SC$:PRIN
T:PRINT"HIGH SCORE ";HS$:PRINT:P
RINT"YOU HAVE COMPLETED LEVEL";L
1:PRINT:PRINT"PRESS <<C>> TO CON
TINUE":PRINT"PRESS <<Q>> TO QUIT
"
7040 AS=INKEYS:IFAS="Q"THENSCEE
N1,0:POKE65314,248:GOTO6000
7045 PLAY"L255O"+STR$(RND(5))+";
"+STR$(RND(12))+"O3"
7050 IF AS<>"C"THEN7040
7060 SCREEN1,0:POKE65314,248:Y=4
5:FORX=181 TO 55STEP-1:PUT(X-1,Y
-3)-(X+1,Y+3),G,NOT:PLAY"A":PUT(
X-1,Y-3)-(X+1,Y+3),G,NOT:IFX<160
THENY=Y+1
7070 NEXTX:FORY=Y TO Y+10:PUT(X-1
,Y-3)-(X+1,Y+3),G,NOT:PLAY"A":P
UT(X-1,Y-3)-(X+1,Y+3),G,NOT:NEXT
Y
7080 FORX=55 TO195:PUT(X-1,Y-3)-
(X+1,Y+3),G,NOT:PLAY"AA":PUT(X-1
,Y-3)-(X+1,Y+3),G,NOT:NEXTX:FOR
Y=Y TO191:PUT(X-1,Y-3)-(X+1,Y+3)
,G,NOT:PLAY"A":PUT(X-1,Y-3)-(X+1
,Y+3),G,NOT:NEXTX

```

```

7090 PMODE0,2:PCLS:PMODE0,1:PCLS
:SCREEN1,1:Y1=40:Y2=151:X1=0:X=1
60:UX=0:SCREEN1,1:IN=-3:DE=1:GOT
O7115
7100 X=X+IN:PLAY"O3V31L255B":PMO
DE0,2:PCLS:LINE(97,191)-(97-X/2,
20+X),PSET,B:LINE-(157+X/2,20+X*.8
),PSET,B:LINE-(157,20),PSET,B:LINE
-(97,20+X*.8),PSET,B:LINE(97,20)
-(97-X/2,20+X*.8),PSET,B:LINE(157,
191)-(157+X/2,20+X),PSET
7110 PCOPY2TO1:PMODE0,1:SCREEN1,
1:IFX=DE THEN 7111 ELSEGOTO7100
7111 FORX=31TO1STEP-1:PLAY"O1AV"
+STR$(X):NEXTX:RETURN
7115 GOSUB7100
7120 IG=0:B=0:IN=0:PCLS:FORX=1TO
94:IG=IG+1:PMODE0,2:PCLS:B=B+3+I
N:IFB>120THENIN=IN-.4ELSEIN=IN+.
2
7125 IF IG>47THENIN=IN+.2
7130 LINE(B-X/3,B-X)-(B+X/3,B+X)
,PSET,B:PCOPY2TO1:PMODE0,1:SCREE
N1,1:NEXTX
7140 PMODE0,2:PCLS:SCREEN1,1:X=0
:IN=3:DE=192:GOSUB7100
7150 FORX=1TO20STEP2:PMODE0,2:PC
LS:LINE(97-X,20-X)-(157+X,181+X/
5),PSET,B:LINE(0,181+X/5)-(255,1
81+X/5),PSET,B:LINE(0,181)-(97-X,2
0-X),PSET,B:LINE(157+X,20-X)-(255,
181),PSET
7160 PCOPY2TO1:PMODE0,1:SCREEN1,
1:NEXTX
7170 FORX=1TO76STEP4:PMODE0,2:PC
LS:LINE(177+X,181+X/2)-(76-X,0)
,PSET,B:LINE-(0,181+X/2),PSET:LI
NE(177+X,0)-(255,181+X/2),PSET:L
INE(0,181+X/2)-(255,181+X/2),PSE
T
7180 PCOPY2TO1:PMODE0,1:SCREEN1,
1:NEXTX
7190 PLAY"O3ACDDDEFAABV15":RETUR
N
8000 PMODE3,1:POKE179,53:PCLS:SC
REEN1,1:POKE65314,248:POKE179,3:
PAINT(0,191),1,4
8010 DRAW"BM36,70R32L32G30R32E5L
22E15R22E10R1C1R5;C4G30R6E20R10G
20R6E30L20R7G1C1G2C4R5G5L5E6C1E2
C4R15C1R5;C4R32G10L22G6R22G14L32
E6R22E4L22E20R33C1R3;C4R32G10L12
G20L6E20L12E10R32C1R7;C4R6G10L2G
14R24G6L32E30R10C1R28"
8020 DRAW"C4R32G10L22G5R22G5L22G
5R22G5L32E30BM96,105R6G6L6E6R7C1
R3C4R6L6G3R3L3G3E6R7C1R12;C4R4D3
G3L9E6R5C1R8;C4R6G6L6E6R7C1R3;C4
R6G6L6E6R7C1R4;C4G6E6R3G3E3R4G6"
8030 FORX=50TO230STEP32:PAINT(X,
75),3,4:NEXTX:PAINT(70,75),3,4
8035 POKE65494,0:PLAY"V1501L9CP3
0CP3CP30CP3":PLAY"L2CL9DP2L2CL9D
P2L2CL4DL3E-L2CL20":FORX=15TO1ST
EP-1:PLAY"GL30G-V"+STR$(X)+"L20"
:NEXTX:POKE65495,0
8040 FORX=1TO255:POKE178,X:LINE(
3,65)-(250,120),PSET,B
8050 IFPEEK(65280)=127ORPEEK(652
80)=255THENNEXTX:GOTO8040ELSELIN
E(5,65)-(250,120),PRESET,B:POKE1
78,53:PAINT(0,0),2:POKE178,3:RE
TURN

```

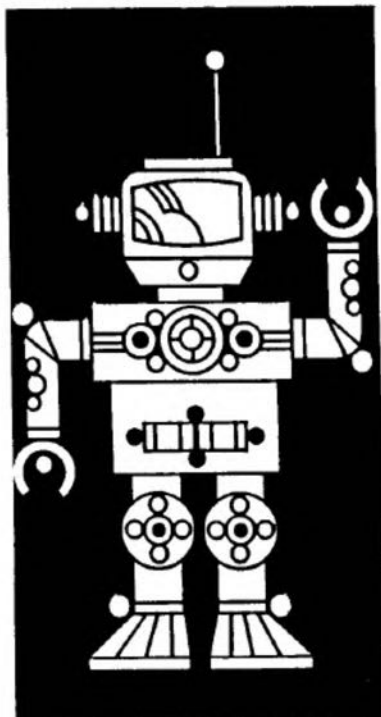
# Robots:

continued from Page 23

Another feature of modern robots is their level of intelligence is not (and may never be) that which exists in fiction. A robot teacher may exist in science fiction stories, but will not be a reality in a classroom during my lifetime. A robot message carrier, however, is a tangible reality.

With current technology, we can reasonably expect robots to perform any task a well-trained pet can perform. Before the end of this century, we may see a robot collecting lunch counts from teachers and delivering this information to the school cafeteria. It is not unreasonable to expect a robot to sweep the gym floor between classes, or to inform the administration when a child wanders away from the playground. But, I seriously doubt a robot will decide the grade a student should receive in math class, or referee an intramural basketball game.

The future of educational robots will probably be more interesting than the present fiction. The current reality of computers in education is much more than past science fiction writers ever dreamed. I would enjoy hearing about the uses of robots in your school. If there is such a creature (even if it deserves the nickname "Frankenstein") in your school, please let me know of your experiences. My address is 829 Evergreen, Chatham, IL 62629.



# Bloops, Bells

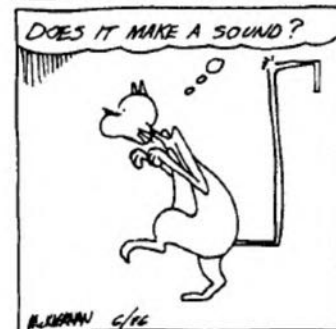
continued from Page 11

```

570 NEXTT
580 RETURN
590 '
600 'TO GET SCREEN LINES
610 DIMS$(17)
620 FORX=1TO17
630 READ S$(X)
640 NEXT:RETURN
649 '
650 PG=1:SCREEN0,1:GOSUB530:PRIN
T@32, " " PAGE
    ONE "":P=98
660 FORX=1TO8:PRINT@P,S$(X);
670 P=P+33:NEXT
680 PRINT@416,CHR$(255);
720 GOTO790
740 '
750 PG=2:SCREEN0,1:GOSUB530
760 PRINT@32, " "
    PAGE TWO " "
;
770 P=99
780 FORX=9TO17:PRINT@P,S$(X);:P=
P+33:NEXT
790 PRINT@416,CHR$(255);
800 INPUTN
810 GOSUB1060
811 IFN=8THEN750
812 IFN=17THEN650
830 ON N GOSUB 80,100,160,190,22
0,250,290,1050,490,320,340,510,3
80,400,430,460,1040
840 IFPG=2THEN750
850 IFPG=1THEN650
855 PRINTPA:PRINTPA:PRINTPA:STOP
860 GOTO750
870 DATA BUZZER-----1
880 DATA DEPTH SOUND----2
890 DATA SIREN WARNING--3
900 DATA BLAST OF LASER-4
910 DATA HISS OR FIZZ---5
920 DATA BLIPP-----6
930 DATA SMALL SPRING---7
940 DATA * SECOND PAGE--8
950 DATA MACHINE GUN----9
960 DATA TAUNT-----10
970 DATA CHARGE-----11
980 DATA HOORAY SONG---12
990 DATA BIG SPRING----13
1000 DATA BOUNCING BALL-14
1010 DATA SIREN-----15
1020 DATA WOLF WHISTLE--16
1030 DATA * FIRST PAGE--17
1040 PG=2:GOTO750
1050 PG=1:GOTO650
1060 SCREEN0,0:RETURN
    
```



CoCo Cat



## CORRECTION

"The CoCo Zone" (May 1986, Page 11): There are no reported problems with the actual program CoCo Zone. There have been, however, numerous calls from readers who are having trouble creating a working copy of the game. RUNNING the CoCo Draw program draws and saves the graphics screens, Zone 0 through Zone 9 to the tape or disk MUST have these files in the following order: Boot, Zone 0, Zone 1, ..., Zone 9 and CoCo Zone. Then, CLOAD and RUN Boot, answer all prompts and you'll soon be ready to play the adventure.

NOTICE that CoCo Draw has nothing to do with the actual play of the game.

# COLOUR CALC

by C. Bartlett

This program is a 25 x 31 column spreadsheet.  
Press the '/' key to display the following prompt  
at the top of the screen:-

"COMMAND: C/D/F/G/I/P/R/S/X"

These commands will now be explained in order.

'/C' This erases in-memory information. Loading  
another file into memory will have the same effect.

'/D' This will delete a column or row. It will  
display "DELETE: (R/C)". Respond with 'R' or 'C' and  
it will ask "COL:" or "ROW:" depending on reply to  
the previous question. Type in the row or column to  
be deleted and the program will display "DELETING  
COL: x" (or "ROW: x").

Columns to the right of rows below the deleted one  
will be moved up or to the left. The program will  
attempt to adjust any moved SUMS though you should  
check these to make sure that they are correct as  
program logic in this area is fairly simplistic due  
to memory restrictions.

'/F' This will fix ROW 1 as a heading so that it  
does not scroll when you move down the page. The  
program will ask "FIX (Y/N)" reply 'Y' to FIX or 'N'  
to remove a previously fixed heading.

'/G' This will move to screen display directly to  
the specified point, it will ask:-

"GOTO: (C:R)"

Respond with the column and row you wish to move  
to.

'/I' This is the reverse to the '/D' command and  
the same warning applies in relation to moved SUMS.  
It will display "INSERT: (R:C)". Reply with 'R' or  
'C' at which point it will display either "COL:" or  
"ROW:". Reply with the column or row to be inserted.

'/P' This will send the columns and rows to the  
printer. It will print from 'A1' to the current  
cursor position.

'/R' This will replicate any complete column or  
row. When selected it will ask:-

"REPLICATE: (C:R)"

Reply 'C' or 'R' and it will display either "COL:"  
or "ROW:". Reply with a column or row number and it  
will show:-

"COL: A: TO:"

Now respond with the destination column or row and  
the column or row will be copied to the destination.

'/S' This is the STORAGE command. Reply 'S' to  
load or 'S' save the program. It will ask for a  
filename, reply with a legal name. During a save the  
screen will revert to green for a moment, this is  
normal and can be ignored.

'/X' This exits the program and returns you to  
BASIC, deleting Color Calc from memory.

CLEAR KEY: Pressing this key causes a screen dump  
to the printer of the rows and columns on the  
screen.

"@SUM" Pressing the key will put you into the SUM  
mode, the display will show SUM at the top of the  
screen. Now type in, first the column row taking  
part in the sum.

"@SUM A1"

Now type an operator:

- + Add
- Subtract
- \* Multiply
- / Divide
- . Add from here to destination

Now type the other column/row taking part in the  
sum.

"@SUM A1+B1"

Next type an equals sign (=) followed by the  
column/row that is to contain the answer.

"@SUM A1+B1=C1"

To force the display of trailing zeros in a  
decimal display add the following symbol: #

"@SUM A1+B1=C1#"

Use this symbol only when necessary, use of this  
symbol will confuse the cursor routine if the cursor  
is placed over a sum block. The block may contain  
two decimal points. To correct this, force a  
recalculation of the sum by retyping the value in  
one of the blocks associated with the sum; better  
still, don't place the cursor on a sum containing  
the '#' sign.

An entry can be made to a block by moving the  
cursor to that block. You can indicate that you have  
completed an entry in a block either pressing ENTER  
or by pressing an arrow key. If you press ENTER,  
what you typed will be so obscured until you move  
the cursor.

Sums may reference other sums, all sums are  
recalculated each time as entry is made.

## The Listing: COLORCAL

10 ' COLOR CALC (C) 11/11/85  
C. BARTLETT

20 POKE359,57:SCREEN0,1:CLS  
30 POKE248,50:POKE249,98:POKE250  
,28:POKE251,175:POKE252,126:POKE  
253,173:POKE254,165:POKE410,126:  
POKE411,0:POKE412,248  
40 POKE65495,0:CLEAR200,32758:FO

RX=32758T032765:READY:POKEX,Y:NE  
XT:EXEC32758:CLEAR7000,32765:DIM  
A\$(27,33),A(27,33):F2\$="##"+CH  
R\$(239):TL\$=STRING\$(8,239):ML\$=S  
TRING\$(28,128):DV=0:DATA204,14,1  
,31,2,126,150,165  
50 SX=1:SY=1:CP=68:C\$=STRING\$(9,  
175):K=1:K2=1:BL\$=""":P  
X=1:PY=1:CC\$=BL\$:F\$="#####.##":  
N1=0:MV=1  
60 CLS:FOR T=0 TO 6:PRINT@T#64+3  
5,CHR\$(239):ML\$:NEXT T  
70 POKE1535,239:PRINT@483,STRING

\$(28,239);  
80 GOTO120  
90 IN\$=INKEY\$:IF IN\$=""THEN 90 E  
LSE SOUND235,1:RETURN  
100 IF N1=0 THEN RETURN  
110 Z1=1:PRINT@64,USINGF2\$;Z1;:Y  
=1:GOSUB940:RETURN  
120 GOSUB380  
130 PRINT@CP,C\$;  
140 GOSUB1940:PRINT@0,"ready";  
150 PRINT@CP,C\$;  
160 OK=-1:SOUND200,1  
170 IN\$=INKEY\$:IF IN\$="" THEN 17

```

0
180 IF ASC(IN$)=3 THEN PRINT@0,"
break disabled";GOTO 170
190 SOUND235,1
200 IF ASC(IN$)=12 THEN DV=2:POK
E65494,0:PRINT#-2,CHR$(15);:GOSU
B330:DV=0:POKE65495,0:GOTO 170
210 GOSUB530:IF RT=0 THEN GOSUB1
940
220 PRINT@CP,C$;
230 IN=ASC(IN$):IF IN=10 THEN OK
=0:PRINT@CP,BL$;:CP=CP+64:PY=PY+
1:K2=K2+1:IF K2=8 THEN K2=7:CP=C
P-64:SY=SY+1:GOSUB1730:GOSUB380
240 IF IN=94 THEN OK=0:PRINT@CP,
BL$;:CP=CP-64:PY=PY-1:K2=K2-1:IF
K2=0 THEN K2=1:SY=SY-1:CP=CP+64
:IF SY=0 THEN SY=1:GOSUB380 ELSE
GOSUB380
250 IF IN=9 THEN OK=0:PRINT@CP,B
L$;:CP=CP+9:K=K+1:PX=PX+1:IF K=4
THEN K=3:CP=CP-9:SX=SX+1:GOSUB1
700:GOSUB380:GOSUB100
260 IF IN=8 THEN PRINT@CP,BL$;:O
K=0:PX=PX-1:CP=CP-9:K=K-1:IF K=0
THEN K=1:CP=CP+9:SX=SX-1:IF SX=
0 THEN SX=1:GOSUB380:GOSUB100 E
LSE GOSUB380:GOSUB100
270 IF PX=0 THEN PX=1 ELSE IF PY
=0 THEN PY=1 ELSE IF PX=26 THEN
PX=25 ELSE IF PY=32 THEN PY=31
280 GOSUB530
290 IF IN=47 THEN GOSUB1940:OK=0
:PRINT@0,"command";:GOSUB2060
300 IF IN=64 THEN GOSUB 560:OK=0
310 IF OK THEN GOSUB 420
320 PRINT@CP,C$;
330 IF OK THEN OK=0:GOTO210
340 GOTO160
350 IF MID$(A$(T3,T4),7,1)=". " O
R MID$(A$(T1,T2),7,1)=". " OR TX$
="# " THEN DP=-1
360 TT$=" "+STR$(TT):TT$
=RIGHT$(TT$,9):IF MID$(TT$,7,1)<
>". " AND DP=-1 THEN TT$=TT$+".00
":TT$=RIGHT$(TT$,9)
370 TX$="":DP=0:RETURN
380 PRINT@36,"";:FOR Y=SX TO SX+
2:PRINT TL$;CHR$(X+96);:NEXTX:PR
INT CHR$(239);:IF DV=2 THEN PRIN
T#-2," ";CHR$(SX+64);"
";CHR$(SX+65);" ";CH
R$(SX+66);CHR$(13)
390 Z=1:FOR Y=SY+N1 TO SY+6:PRIN
T@((Z+N1)*64),"";:PRINT#-DV,USIN
G F2$;Y;:GOSUB940:Z=Z+1:GOSUB400
:NEXT Y:RETURN
400 IF DV=2 THEN PRINT#-2,CHR$(1
3)
410 RETURN
420 GOSUB1940:AD$=IN$:PRINT@CP,A
D$;:FOR T=1 TO 8:IF AD$=CHR$(13)
THEN AD$="":GOTO 490
430 GOSUB90
440 IN=ASC(IN$):IF IN=13 OR IN=9
OR IN=94 OR IN=10 THEN 490
450 IF IN$=CHR$(8) AND T>1 THEN
AD$=LEFT$(AD$, (LEN(AD$)-1)):T=T-
2:PRINT@CP,AD$;CHR$(175);:GOTO 4
30
460 AD$=AD$+IN$
470 PRINT@CP,AD$;
480 NEXT T
490 A$(PX,PY)=" "+AD$:A$
(PX,PY)=RIGHT$(A$(PX,PY),9)
500 A(PX,PY)=VAL(AD$)
510 GOSUB380
520 RETURN
530 TK$=LEFT$(A$(PX,PY),1):IF A$
(PX,PY)="" THEN BL$=CC$ ELSE BL$
=A$(PX,PY)
540 IF TK$="" THEN QX=PX:Y=PY:G
OSUB1760:GOSUB2020:BL$=TT$
550 RETURN
560 GOSUB1940:PRINT@8,IN$;"sum "
;
570 GOTO1290
580 A$(A7,A8)=""+"A1$+A2$+A3$+A4
$+A5$+A6$+A7$+A8$+A9$:A9$="":GOS
UB380:GOSUB1940:RETURN
590 GOSUB90
600 IF IN$="R" THEN PRINT"ROW:";
:GOTO630
610 IF IN$="C" THEN PRINT"COL:";
:GOTO630
620 GOTO590
630 I$=INKEY$:IF I$="" THEN 630
ELSE PRINTI$;
640 I1$=INKEY$:IF I1$="" THEN640
ELSE IF I1$=CHR$(13) THEN 660
650 I$=I$+I1$:PRINTI1$;
660 RETURN
670 IF IN$="R" THEN PRINT"row:";
I$;
680 IF IN$="C" THEN PRINT"col:";
I$;
690 RETURN
700 IF IN$="C" THEN R=ASC(I$)-64
:GOTO720
710 R=VAL(I$):FOR Y=R TO 31:FOR
X=1 TO 25:A$(X,Y)=A$(X,Y+1):A(X,
Y)=A(X,Y+1):GOSUB770:NEXT X,Y:FO
R X=1TO 25:A$(X,31)=""A(X,31)=0
:NEXT X:GOSUB380:RETURN
720 FOR X=R TO 24:FOR Y=1 TO 31:
A$(X,Y)=A$(X+1,Y):A(X,Y)=A(X+1,Y
):GOSUB730:NEXT Y,X:FOR Y=1 TO 3
1:A$(25,Y)=""A(25,Y)=0:NEXT Y:G
OSUB380:RETURN
730 IF LEFT$(A$(X,Y),1)<>"@" THE
N RETURN
740 QX=X:GOSUB1770:GOSUB1020:T1$
=STR$(T1):T3$=STR$(T3):IF LEN(T1
$)=3 THEN T1$=RIGHT$(T1$,2)
750 IF LEN(T3$)=3 THEN T3$=RIGHT
$(T3$,2)
760 A$(X,Y)=""+"T1$+T2$+T5$+T3$+
T4$+TX$:RETURN
770 IF LEFT$(A$(X,Y),1)<>"@" THE
N RETURN
780 QX=X:GOSUB1770:GOSUB1050:T2$
=STR$(T2):T4$=STR$(T4):IF LEN(T2
$)=3 THEN T2$=RIGHT$(T2$,2)
790 IF LEN(T4$)=3 THEN T4$=RIGHT
$(T4$,2)
800 A$(X,Y)=""+"T1$+T2$+T5$+T3$+
T4$+TX$:RETURN
810 PRINT@0,"delete:(R/C) ";:GOS
UB590:GOSUB1940:PRINT@0,"deletin
g ";:GOSUB670:GOTO700
820 PRINT@0,"insert:(R/C) ";:GOS
UB590:GOSUB1940:PRINT@0,"inserti
ng ";:GOSUB670:GOTO830
830 IF IN$="C" THEN R=ASC(I$)-64
:GOTO850
840 R=VAL(I$):FOR Y=31 TO R STEP
-1:FOR X=25 TO 1 STEP-1:A$(X,Y+1
)=A$(X,Y):A(X,Y+1)=A(X,Y):GOSUB8
60:NEXT X,Y:FOR X=1 TO 25:A$(X,R
)=""A(X,R)=0:NEXT X:GOSUB380:RE
TURN
850 FOR X=24 TO R STEP-1:FOR Y=3
1 TO 1 STEP-1:A$(X+1,Y)=A$(X,Y):
A(X+1,Y)=A(X,Y):GOSUB900:NEXT Y,
X:FOR Y=1 TO 31:A$(R,Y)=""A(R,Y
)=0:NEXT Y:GOSUB380:RETURN
860 IF LEFT$(A$(X,Y),1)<>"@" THE
N RETURN
870 QX=X+1:GOSUB1770:GOSUB1080:T
2$=STR$(T2):T4$=STR$(T4):IF LEN(
T2$)=3 THEN T2$=RIGHT$(T2$,2)
880 IF LEN(T4$)=3 THEN T4$=RIGHT
$(T4$,2)
890 A$(X+1,Y)=""+"T1$+T2$+T5$+T3
$+T4$+TX$:RETURN
900 IF LEFT$(A$(X,Y),1)<>"@" THE
N RETURN
910 QX=X+1:GOSUB1770:GOSUB1110:T
1$=STR$(T1):T3$=STR$(T3):IF LEN(
T1$)=3 THEN T1$=RIGHT$(T1$,2)
920 IF LEN(T3$)=3 THEN T3$=RIGHT
$(T3$,2)
930 A$(X+1,Y)=""+"T1$+T2$+T5$+T3
$+T4$+TX$:RETURN
940 V1$=LEFT$(A$(SX,Y),1):V2$=LE
FT$(A$(SX+1,Y),1):V3$=LEFT$(A$(S
X+2,Y),1)
950 IF V1$<>"@" THEN O1=A(SX,Y):
GOSUB1950:GOTO 970
960 QX=SX:GOSUB1760:GOSUB2000
970 IF V2$<>"@" THEN O2=A(SX+1,Y
):GOSUB1960:GOTO990
980 QX=SX+1:GOSUB1760:GOSUB2000
990 IF V3$<>"@" THEN O3=A(SX+2,Y
):GOSUB1970:GOTO1010
1000 QX=SX+2:GOSUB1760:GOSUB2000
1010 RETURN
1020 IF T1>R THEN T1=T1-1
1030 IF T3>R THEN T3=T3-1
1040 RETURN
1050 IF T2>R THEN T2=T2-1
1060 IF T4>R THEN T4=T4-1
1070 RETURN
1080 IF T2>R THEN T2=T2+1
1090 IF T4>R THEN T4=T4+1
1100 RETURN
1110 IF T1>R THEN T1=T1+1
1120 IF T3>R THEN T3=T3+1
1130 RETURN
1140 PRINT@0,"goto:(c:r) ";:GOSU
B1150:GOTO1220
1150 GOSUB90
1160 SX=ASC(IN$)-64:IF SX<1 THEN
1150 ELSE IF SX>24 THEN SX=23
1170 PRINTIN$;"";
1180 GOSUB90
1190 PRINTIN$;
1200 N$=INKEY$:IF N$="" THEN 120
0 ELSE IF N$=CHR$(13) THEN 1210
ELSE IN$=IN$+N$:PRINTN$;
1210 RETURN
1220 SY=VAL(IN$):IF SY=0 THEN SY
=1 ELSE IF SY=26 THEN SY=25
1230 SY=SY+N1
1240 PX=SX+(K-1):PY=SY+(K2-1):G
OSUB380:PRINT@CP,C$;:BL$=A$(PX,P
Y):GOTO160
1250 PRINT@0,"fix (Y/N) ";
1260 GOSUB90:PRINTIN$;:GOSUB1940
:IF IN$="Y" THEN 1270 ELSE IF IN
$="N" THEN 1280 ELSE 1250
1270 N1=1:RETURN

```

```

1280 N1=0:RETURN
1290 AD$="":FOR T=1 TO 13
1300 GOSUB90:IF IN$=CHR$(13) THE
N1350
1310 IF IN$=CHR$(8) THEN AD$=LEF
T$(AD$, (LEN(AD$)-1)):T=T-2:PRINT
@8,"@sum ";AD$;GOTO1300
1320 IF IN$="." OR IN$="+" OR IN
$="-" OR IN$="*" OR IN$="/" THEN
PRINTIN$;:IN$=" "+IN$:AD$=AD$+I
N$:GOTO1340
1330 PRINTIN$;:AD$=AD$+IN$
1340 NEXT T
1350 L=LEN(AD$):A1$=LEFT$(AD$,1)
:AD$=RIGHT$(AD$,L-1):L=L-1
1360 KP=INSTR(AD$,"."):IF KP=0 T
HEN GOTO1560
1370 A2$=MID$(AD$,1,KP-1):AD$=RI
GHT$(AD$,L-KP)
1380 L=LEN(AD$)
1390 A3$=LEFT$(AD$,1):AD$=RIGHT$(
AD$,L-1):L=L-1
1400 IF A3$="." OR A3$="+" OR A3
$="-" OR A3$="*" OR A3$="/" THEN
1410 ELSE GOTO 1560
1410 A4$=LEFT$(AD$,1):AD$=RIGHT$(
AD$,L-1):L=L-1
1420 KP=INSTR(AD$,"="):IF KP=0 T
HEN GOTO 1560
1430 A5$=MID$(AD$,1,KP-1):AD$=RI
GHT$(AD$,L-(KP-1))
1440 L=LEN(AD$)
1450 A6$=LEFT$(AD$,1):AD$=RIGHT$(
AD$,L-1):L=L-1:IF A6$("<>")="" THEN
GOTO 1560
1460 A7$=LEFT$(AD$,1):AD$=RIGHT$(
AD$,L-1):L=L-1
1470 A8$=AD$:A8=VAL(A8$)
1480 A1=ASC(A1$)-64:A1$=STR$(A1)
:IF LEN(A1$)=3 THEN A1$=RIGHT$(A
1$,2)
1490 IF LEN(A2$)=1 THEN A2$=" "+
A2$
1500 A4=ASC(A4$)-64:A4$=STR$(A4)
:IF LEN(A4$)=3 THEN A4$=RIGHT$(A
4$,2)
1510 IF LEN(A5$)=1 THEN A5$=" "+
A5$
1520 A7=ASC(A7$)-64:A7$=STR$(A7)
:IF LEN(A7$)=3 THEN A7$=RIGHT$(A
7$,2)
1530 IF LEN(A8$)=1 THEN A8$=" "+
A8$
1540 IF INSTR(AD$,"#")(>0) THEN A
9$="#"
1550 GOTO580
1560 GOSUB1940:PRINT@0,"format e
rror";:SOUND5,5:RETURN
1570 PRINT@0,"replicate: (c:r) "
;
1580 GOSUB90:PRINTIN$;:IF IN$="C
" THEN 1590 ELSE IF IN$="R" THEN
1650 ELSE 1570
1590 GOSUB1940:PRINT@0,"col:";
1600 GOSUB90: SX=ASC(IN$)-64:IF S
X<1 THEN SX=1 ELSE IF SX>25 THEN
SX=25
1610 PRINTIN$;:TO:";
1620 GOSUB90:S2=ASC(IN$)-64:IF S
2<1 THEN S2=1 ELSE IF S2>25 THEN
S2=25
1630 PRINTIN$;
1640 FOR Y=1TO 31:A$(S2,Y)=A$(SX
,Y):A(S2,Y)=A(SX,Y):NEXT Y:GOSUB

```

```

380:RETURN
1650 GOSUB1940:PRINT@0,"row:";
1660 GOSUB90:PRINTIN$;:S3$=IN$:G
OSUB90:GOSUB2140:PRINTIN$;:S3$=S
3$+IN$:S3=VAL(S3$):IF S3<1 THEN
S3=1 ELSE IF S3>31 THEN S3=31
1670 PRINT":TO:";
1680 GOSUB90:S4$=IN$:PRINTIN$;:G
OSUB90:GOSUB2140:S4$=S4$+IN$:PRI
NTIN$;:S4=VAL(S4$):IF S4<1 THEN
S4=1 ELSE IF S4>31 THEN S4=31
1690 FOR X=1 TO 25:A$(X,S4)=A$(X
,S3):A(X,S4)=A(X,S3):NEXT X:GOSU
B380:RETURN
1700 IF SX=24 THEN SX=23
1710 IF PX=26 THEN PX=25
1720 RETURN
1730 IF SY>=26 THEN SY=25
1740 IF PY=32 THEN PY=31
1750 RETURN
1760 TT=0:GOSUB1770:GOTO1850
1770 T1$=MID$(A$(QX,Y),2,2)
1780 T2$=MID$(A$(QX,Y),4,2)
1790 T3$=MID$(A$(QX,Y),7,2)
1800 T4$=MID$(A$(QX,Y),9,2)
1810 T5$=MID$(A$(QX,Y),6,1)
1820 TX=INSTR(A$(QX,Y),"#"):IF T
X(>0) THEN TX$="#"
1830 T1=VAL(T1$):T2=VAL(T2$):T3=
VAL(T3$):T4=VAL(T4$)
1840 RETURN
1850 IF T5$="+" THEN TT=A(T1,T2)
+A(T3,T4)
1860 IF T5$="-" THEN TT=A(T1,T2)
-A(T3,T4)
1870 IF T5$="*" THEN TT=A(T1,T2)
*A(T3,T4)
1880 IF T5$="/" THEN TT=A(T1,T2)
/A(T3,T4)
1890 IF T5$="." THEN 1910
1900 GOTO1930
1910 IF T1=T3 THEN 1920 ELSE IF
T1<T3 THEN FOR W=T1 TO T3:TT=TT+
A(W,T2):NEXT W ELSE IF T1>T3 THE
N FOR W= T1 TO T3 STEP-1:TT=TT+A
(W,T2):NEXT W
1920 IF T2<T4 THEN FOR W=T2 TO T
4:TT=TT+A(T1,W):NEXT W ELSE IF T
2>T4 THEN FOR W= T2 TO T4 STEP-1
:TT=TT+A(T1,W):NEXT W
1930 DP=0:GOSUB350:TT$=RIGHT$(TT
$,9):A(QX,Y)=TT:RETURN
1940 PRINT@0,"
";:RETURN
1950 U1= SX:U2=01:U3$=V1$:GOTO198
0
1960 U1= SX+1:U2=02:U3$=V2$:GOTO1
980
1970 U1= SX+2:U2=03:U3$=V3$:GOTO1
980
1980 IF MID$(A$(U1,Y),7,1)="." T
HEN PRINT#-DV, USING F$;U2; ELSE
IF U3$("<>")="" THEN IF A$(U1,Y)("<>")
" THEN PRINT#-DV, A$(U1,Y); ELSE
PRINT#-DV, " ";
1990 RETURN
2000 IF MID$(TT$,7,1)="." THEN P
RINT#-DV,USING F$;TT; ELSE PRINT
#-DV, TT$;
2010 RETURN
2020 T1$=" "+CHR$(T1+64)
2030 T3$=" "+CHR$(T3+64)
2040 PRINT@8,"@sum ";T1$;T2$;T5$
;T3$;T4$;

```

```

2050 RETURN
2060 PRINT"C/D/F/G/I/P/R/S/X ";
2070 GOSUB90
2080 GOSUB1940
2090 IF IN$="D" THEN 810 ELSE IF
IN$="I" THEN 820 ELSE IF IN$="G
" THEN 1140 ELSE IF IN$="F" THEN
1250 ELSE IF IN$="X" THEN 2100
ELSE IF IN$="C" THEN 2110 ELSE I
F IN$="R" THEN 1570 ELSE IF IN$=
"S" THEN 2150 ELSE IF IN$="P" TH
EN 2120 ELSE 2130
2100 POKE 65494,0:CLS:PRINT"term
inated";:NEW:END
2110 FOR X=1TO 25:FOR Y=1TO 31:A
$(X,Y)="" :A(X,Y)=0:NEXT Y,X:PRIN
T@0,"cleared
";:GOSUB380:GOTO160
2120 PRINT@0,"printing
";:GOTO2300
2130 GOSUB1940:PRINT@0,"invalid"
;:SOUND5,5:GOTO160
2140 IF IN$=CHR$(13) THEN IN$=""
:RETURN ELSE RETURN
2150 POKE65494,0:PRINT@0,"comman
d:STORAGE (L/S) ";
2160 GOSUB90:IF IN$="L" THEN 217
0 ELSE IF IN$="S" THEN 2180 ELSE
2160
2170 GOSUB1940:PRINT@0,"load :FI
LENAME ";:GOSUB2190:GOTO2280
2180 GOSUB1940:PRINT@0,"save:FIL
ENAME ";:GOSUB2190:GOTO2260
2190 AD$="":FOR I=1 TO 8
2200 GOSUB90
2210 IF IN$=CHR$(13) THEN 2250
2220 IF IN$=CHR$(8) THEN AD$=LEF
T$(AD$, (LEN(AD$)-1)):I=I-2:PRINT
@15,AD$;" ";:GOTO 2200
2230 AD$=AD$+IN$:PRINT@15,AD$;
2240 NEXT I
2250 AD$=AD$+"/DAT":RETURN
2260 POKE359,126:SCREEN0,0:OPEN"
O",#1,AD$
2270 FOR X=1 TO 25:FOR Y=1 TO 31
:WRITE #1,A$(X,Y),A(X,Y):NEXT Y,
X:CLOSE1:POKE359,57:SCREEN0,1:PR
INT@0,"saved
";:POKE65495,0:GOTO160
2280 OPEN"I",#1,AD$
2290 FOR X=1 TO 25:FOR Y=1 TO 31
:INPUT#1,A$(X,Y),A(X,Y):NEXT Y,X
:CLOSE1:PRINT@0,"loaded
";:POKE65495,0:GO
SUB380:GOTO160
2300 POKE65494,0:PRINT#-2,CHR$(1
5);:FOR Y=1 TO PY:FOR X=1 TO PX
2310 IF MID$(A$(X,Y),7,1)="." TH
EN PRINT#-2,USING F$;A(X,Y);:GOT
O2360
2320 IF LEFT$(A$(X,Y),1)="" THE
N K$=STR$(A(X,Y)):IF INSTR(K$,".
")(>0) THEN PRINT#-2, USING F$;A(
X,Y);:GOTO2360
2330 IF LEFT$(A$(X,Y),1)="" THE
N PRINT#-2,USING"#####";A(X,
Y);:GOTO2360
2340 IF A$(X,Y)="" THEN PRINT#-2
," ";:GOTO2360
2350 PRINT#-2,A$(X,Y);:GOTO2360
2360 NEXTX:PRINT#-2," ":NEXT Y:P
OKE65495,0:GOSUB1940:RETURN
2370 IN$=INKEY$:IF IN$="" THEN 2
370 ELSE PRINTASC(IN$):GOTO2370

```

Converting ASCII text from MS-DOS  
disks to CoCo disks

# The Great Transformation

By Marty Goodman

**M**uch as we love the CoCo, the fact is the IBM PC (and its clones) running Microsoft Disk Operating System (MS-DOS) is by far the most commonly used personal microcomputer for business. Many of you may use one at work, or have friends who use them.

This article is intended to provide you with a means of converting ASCII text files on a disk created using an MS-DOS computer into ASCII text files on a Color Computer-type disk. Next month, a companion article will provide you with the means of creating an MS-DOS-type disk on your Color Computer and the means to write CoCo text files to such a disk.

## System Requirements

A 64K Color Computer with Disk Extended BASIC (versions 1.0 or 1.1) and two disk drives are required for these programs. Two drives are needed to allow the file conversions to occur at a reasonable rate of speed. That is why no attempt was made to write this utility for single-drive systems.

The disk drives must be capable of fully accessing 40 tracks. There is no way around this; MS-DOS uses all 40 tracks. Most disk drives sold by third-party suppliers will access 40 tracks, as do most Tandy disk drives manufactured over the last couple of years. Most of the full-height drives Tandy sold in white cabinets will access 40 tracks. All of the half-height drives Tandy has been selling are capable of 40-track operation.

## The Problem of File Conversion

The Color Computer differs widely from the IBM PC. The CoCo uses disks formatted with 35 tracks and 18 256-byte sectors per track; the PC uses 40 tracks with nine 512-byte sectors per track. The CoCo uses single-sided disk drives; the PC uses double-sided disk drives. The CoCo uses a Western Digital

or Fujitsu brand disk controller chip; the IBM PC uses a NEC disk controller chip. The CoCo uses the Radio Shack Disk BASIC operating system; the PC uses MS-DOS. All of these differences cause problems.

The fact that many IBM PC word processors store files in a form that is not exactly ASCII text causes further complications. You simply have to make sure the files you wish to read on the IBM PC-type disk are pure ASCII text. Most programs that do not normally use pure ASCII for text storage provide, as an option or separate conversion program, the means of turning their file format into pure ASCII.

Working in our favor is the fact that both computers use the same size disk, that Microsoft wrote the code for both MS-DOS and Disk BASIC, and that the Western Digital (or Fujitsu) disk controller chip can read or write anything written by the NEC controller chip. (Oddly, the NEC chip cannot read everything written by the Western Digital chip. In converting CoCo material to an MS-DOS disk, one has to take the special limitations of the NEC chip into account when formatting a disk for it using the Western Digital chip.)

I have added one feature to my MS-DOS-to-CoCo conversion program. It pokes a little routine into memory that sets the high order bit of all characters in the MS-DOS file to zero before converting them to CoCo disk format. It also strips out line feeds from the MS-DOS file.

Most MS-DOS ASCII and other text files end lines with a carriage return character (Hex \$0D), followed by a line feed character (Hex \$0A). But CoCo word processors are accustomed to seeing lines ended *only* by the carriage return. Some CoCo word processors automatically remove any line feeds they may find in a file. *Telewriter-64* is one example. However, some CoCo editors (i.e. the editor for Macro 80C)

choke on a file if it contains line feeds. In general, when converting text from an MS-DOS system to a CoCo Disk BASIC system, it is useful, sometimes mandatory, to strip off line feeds. The resetting of the high-order bit to zero should help convert *WordStar* and some other type files into ASCII for compatibility with CoCo word processors and editors.

The "filter" routine that does the line feed and high-order bit stripping is executed in Line 5050. If you want to have the file converted without this filter, just delete Line 5050: load the converter program, then before running the program, type 5050 and press ENTER. This deletes Line 5050 and disables the filter. You may want to save the program with that line deleted.

## Program Limitations and Idiosyncrasies

There are several limitations with this file conversion program. First, it can only read single-sided disks (because so few CoCo owners use double-sided disk drives). Virtually all MS-DOS users use double-sided disks. However, originally MS-DOS used single-sided disks, and to maintain backwards compatibility, the current versions of MS-DOS have the ability to read and write single-sided versions of MS-DOS disks. In order to create an MS-DOS disk that can be read by a CoCo with single-sided drives, the user must first format a single-sided disk on the MS-DOS machine. Under MS-DOS, do this with Drive B and give the command `FORMAT B: /1`.

After prompting you to put a blank disk in Drive B, the computer formats a single-sided MS-DOS disk. You must transfer any files you want converted to that disk. Only MS-DOS disks prepared in this manner can be read by this MS-DOS-to-CoCo conversion program. (It is possible to write a conversion utility to read double-sided [normal] MS-DOS disks on the CoCo, but



double-sided drives are required.)

MS-DOS supports volume labels and subdirectories. To keep the conversion program simple, I elected not to write code that took either into account. Therefore, the files you wish to transfer *must* be put in the Root directory of the single-sided MS-DOS disk. This program ignores volume labels, subdirectories and killed files, but I do suggest not having any of these things on the disk you are preparing for file conversion.

Files can be of any length, but the computer reads and writes data a sector at a time. To keep the code simple, I made it convert files a sector at a time and let the last sector in each file be fully converted, even if the file is supposed to include only part of that last sector. The result is that it adds some trash at the end of files it converts, although all of the file does get converted. This extra trash often shows up as part of the original file itself. Indeed, that trash will very often be material from just before the end of the file. But if you look a little farther back, you'll see the true end of the file, followed by up to 255 characters of text, which is a repeat of stuff near the end of the file. The extra trash can be edited out with a word processor.

One note for *Teletwriter* users: *Teletwriter* does weird things when it encounters a caret sign in incoming text. The caret itself is not displayed, and the character following it is lost or altered. I suggest you make sure there are no caret signs in the text you convert. If need be, first edit the text using an MS-DOS-based word processor on your MS-DOS machine.

Users of *VIP Writer* will need to do a little extra work to prepare their files for loading into the word processor. Unfortunately, *VIP* is set up to interpret a null as an end-of-file marker. This MS-DOS-to-CoCo conversion routine doesn't actually remove the line feeds; it converts them into nulls. Because of this, *VIP* loads only the first line of the converted text. The rest is ignored as the null at the end of the line tells *VIP* it has reached the end-of-the-file.

To correct this, *VIP* users need to delete Line 5050 in the main program. This tells the converter not to strip the line feeds. Then, after the conversion is complete, you can do one of two things: 1) Leave the file as is and edit out the line feeds (the hard way), or 2) Run the file through the program in Listing 2 and answer the prompts. This listing strips the line feeds right out of the file.

### Using the Program

First, make up your MS-DOS single-sided disk with MS-DOS ASCII text

files you want to convert. Be sure to put all files in the Root directory. Be sure the disk does not have a volume label, subdirectories or a killed file. Put that disk in your CoCo's Drive 1. Put a disk with this conversion program (*MS2COCO.BAS*) in Drive 0. Type `LOAD "MS2COCO"` and `RUN`. Upon seeing the title page, make sure your specially prepared MS-DOS disk is in Drive 1, then press `ENTER`. Follow the prompts to view all entries on the root directory of the MS-DOS disk. Select the entry you want to convert by typing its number, then pressing `ENTER` when asked to confirm that selection.

Be sure you have adequate blank space on the Disk BASIC disk in Drive 0. That disk must be formatted in ordinary format — just follow the prompts. After selecting the file on the MS-DOS disk you wish to convert, you are asked to choose a filename for it as it will appear on the CoCo disk. Note that you are only to enter an eight-letter filename; the program automatically assigns the extension `"/TXT."`

The speed of file conversion is roughly 2400 Baud. This is accomplished by a sneaky programming trick, the "VARPTR trick." You might wish to examine the code between 5000 and 5200 to see how the `VARPTR` statement is used to help create a 256-character long string in one fell swoop.

During file conversion some curious numbers appear on the bottom of the screen. These numbers were put there mostly to help debug the program. Going from left to right, they represent the cluster number, track and sector number on the MS-DOS disk that is currently being acted on.

On the right bottom part of the screen you will see a display of the number of bytes that remain to be converted. As the file is converted, that number decreases to zero. The number gives an idea how far along the program is in the process of file conversion. When the program is done converting a file, it prompts with a beep and asks if you are done or if you want to convert another file.

Special thanks go to Don Hutchison (user name `DONHUTCHISON` on Delphi) for his beta testing of this utility and his help in dealing with line feeds, including the program in Listing 2. Also to Art Flexser (`ARTFLEXSER` on Delphi), author of `ADOS`, for his help in suggesting the `VARPTR` trick and for providing a routine from which Don derived the line feed-stripper program.

Thanks to Cray Augsburg (`RAINBOWMAG` on Delphi) for his beta testing that revealed the problems to be

encountered by users of *VIP Writer*. Extra special thanks go to Fred Cisin, author of *Xenocopy*, who spent hours patiently teaching me about MS-DOS disk file structure.

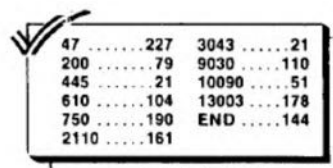
Next month we'll present the other half of this package: a group of programs that allow you to format an MS-DOS-type disk on the CoCo and to write Disk BASIC text files onto that MS-DOS disk using the CoCo. A short simple BASIC program will be included that inserts line feeds into CoCo ASCII files after the carriage return, making such files more palatable to MS-DOS-type text handling programs.

For those with other file conversion needs, please take note of the following:

Mark Data Products makes a program called *CoCo Util* that converts to and from Disk BASIC on an MS-DOS-type machine. (Note that this is an MS-DOS program and runs only on MS-DOS machines.) It does, of course, support double-sided MS-DOS disks. Mark Data Products, 24001 Alicia Parkway, No. 207, Mission Viejo, CA 92691, (714) 768-1551.

D.P. Johnson makes conversion utilities to handle file conversions between OS-9 and MS-DOS disks. These utilities run on the Color Computer under OS-9 and support double-sided disk drives. D.P. Johnson, 7655 S.W. Cedarcrest Street, Portland, OR 97223, (503) 244-8152.

For those with an IBM PC or other MS-DOS machine who wish the ultimate in file conversion utilities, let me recommend *XENOCOPY II*. It runs on nearly all MS-DOS machines, and reads from, writes to and formats over 250 different disk formats. This includes OS-9, Color Computer and hundreds of CP/M variant formats. If you obtain special hardware, this conversion program also supports a number of eight-inch and 3½-inch disk formats. *Xenocopy II* is available from Xenosoft, 1454 6th Street, Berkeley, CA 94710, (415) 525-3113. □



47	.....227	3043	.....21
200	.....79	9030	.....110
445	.....21	10090	.....51
610	.....104	13003	.....178
750	.....190	END	.....144
2110	.....161		

Listing 1: MS2COCO

```
1 CLEAR 512,&H5DFF
2 PCLEAR 4
3 DIM LKS(8)
5 DIM NTRYLC(8)
20 IS=&H60:ID=&H6000 'MS DOS DAT
A SECTOR BUFFER
25 FS=&H62:FD=&H6200 'FAT BUFFER
```

# Tandy ELECTRONICS

## Exclusive Offer To Rainbow/COCO Readers.



Color LOGO

**Save \$55** Reg 89.95 **34<sup>95</sup>**

Program Pak for the Color Computer. Helps children understand graphic relationships and develop problem solving skills. Manipulate a "turtle" around the screen to create simple or advanced graphics. With easy-to-understand instruction manual. 26-2722

Disk Version of Color LOGO

**Save \$70** Reg 169.95 **99<sup>95</sup>**

Same as above but in diskette form. Ideal for children under reading age. Pre-defined single key commands moves "turtle/" on the screen. 26-2721

Expand Your Child's Vocabulary

**Save \$7** Reg 14.95 **7<sup>95</sup>**

Vocabulary Tutor 1. An exciting word and description matching game that your children will love to play and learn with. 26-2568

Vocabulary Tutor 2 features matching words and definitions, placing words in the right sentences.

26-2569

## Educational Software for Your Tandy Computer

Educational Sourcebook  
**Save \$6** Reg 10.95 **4<sup>95</sup>**

A complete reference source to all educational software programs and where to find them. Includes all aspects of management and tutoring. With full index to make finding that selection so much easier. At this price, no computer user can afford not to have a copy. 26-2756

Australian Educational Sourcebook  
**Save \$7** Reg 14.95 **7<sup>95</sup>**  
EACH

The ideal supplement to the above. Don't limit your software package to just your local distributor. See the full range of what's available in the educational area. With all the Australian software as well as what's produced overseas. 26-7991

Learn All About Number Magic

**Save \$1<sup>96</sup>** Reg 2.95 **99<sup>c</sup>**

The best and easiest way to come to terms with numbers. Create and detect number patterns from the book and use them in your computer. 26-2752

How To Make Your Computer Like You

**Save \$1<sup>96</sup>** Reg 2.95 **99<sup>c</sup>**

Yes, they've got feelings too! And with "My TRS-80 Likes Me", you can get on your computer's best side. Makes general programming simple to understand. Learn how to write that program in as few steps as necessary. For all computer users. 26-2751

### Language, Arts and Reading Programs



**Save  
\$60**

Reg 99.95

**39<sup>95</sup>**

That's right! \$60 off a great range of software for the TRS-80 Model III. HMRS (High Motivation Reading Series) has 4 copies of the student reader and a read-along tape with each program. Computer measures general comprehension in areas such as sequence of events, details and separating fact from opinion. Spelling and vocabulary drills also involved. Charles Lindbergh/Amelia Earhart. 26-2513  
Hounds of the Baskervilles. Suspense. 26-2514  
Dracula. Separate the facts from fiction. 26-2515  
Moby Dick. A whale of a story! 26-2516  
The Beatles. The lives of the famous four. 26-2517  
20,000 Leagues Under the Sea. Adventure. 26-2518  
Student Record System. Monitor performance. 26-2521

Limited stock only of these exclusive specials  
Ask your Store Manager to help locate these for you

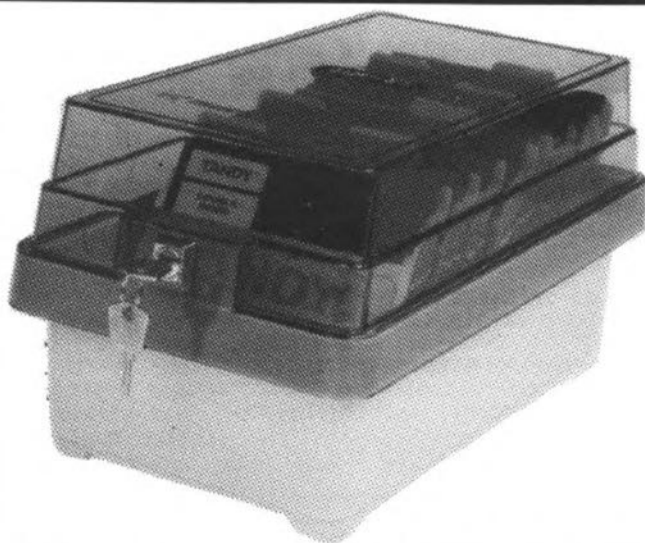
# Save on Diskettes and Accessories



**\$10 Off  
EACH  
PACK**

What great value! Ten single-sided, double density 13.3cm diskettes for under thirty dollars! All are unformatted and soft sectored. And unlike most, our disks are double coated with an advanced ferric-oxide blend to provide maximum lubrication and resistance to diffusion. With forty tracks on each disk for all your programs. 26-406 ... Reg 39.95 .... **29.95**

Double-sided diskettes double your memory facilities without halving your wallet! These double-sided, unformatted diskettes are less than four dollars per disk. Manufactured to the highest standards of excellence, you can be confident that your programs and data are safe. 13.3cm diskettes with 40 tracks available. 26-412 ... Reg 49.95 ..... **39.95**



**Save  
\$15**      **29<sup>95</sup>**      Reg 44.95

Protect your personal disks and keep them in an easy-to-find system. This locking diskette box will store up to 100 13.3cm disks for your quick reference. Strong, durable construction means that when locked, the only way in is with a key. Folding "see-through" top locks shut. An essential for any computer user in the home, office or school. 26-9406

## Expansions and Upgrades for the Tandy 1000

**256K Memory Expansion Board.** User-installable and with full instructions. Get the most out of your machine. Add additional memory if desired for 384K. Was 499.95 25-1004 ..... **Now 299.95**

**Tandy 1000 Disk Drive Kit.** Provides an additional 360K of disk storage for your computer. Mounts internally and installation is recommended (not included) Was 399.95 25-1005 ..... **Now 349.95**

**RS-232C Option Board.** Expand your computer's uses. Add software and external modem to talk with other computers by phone. Also great with serial plotters, printers and more. Reg 219.95 25-1006 ..... **Now 179.95**

**Hard Disk Controller Board.** Allows you to add hard disks drives for up to 35 million characters of storage. Includes driver software and cable for our 15, 35 megabyte systems. Was 549.95 25-1007 .. **Now 399.95**

**512K Memory Expansion Board.** Lets you upgrade your machine from 384K up to 512K. So easy to install with and a full instruction manual. Add additional memory for 640K. Reg 399.95 25-1009 ..... **Now 199.95**

**DigiMouse Controller/Calendar Board.** Dual purpose controller for DigiMouse and perpetual time-date clock. Includes demonstration program and documentation. Was 199.95 25-1010 ..... **Now 149.95**

**SALE ENDS: 30/7/86**

**WE SERVICE WHAT WE SELL!**

# Tandy ELECTRONICS

A DIVISION OF TANDY  
AUSTRALIA LIMITED  
INC. IN N.S.W.

Nearly  
350 Stores  
Australia-  
Wide

**Available from 350 Stores  
Australia-wide including  
Tandy Computer Centres**

*Independent Tandy Dealers may not be participating  
in this ad or have every item advertised.  
Prices may also vary at individual Dealer Stores*

```

30 DS=&H66:DD=&H6600 'MS DOS DIR
ECTORY BUFFER
35 DEND=&H71FF 'END OF DIRECTORY
40 POKE &H7E00,&H8E:POKE &H7E01,
&H60:POKE &H7E02,0 'LDX #56000
41 POKE &H7E03,&HA6:POKE &H7E04,
&H84 'LDA 0,X
42 POKE &H7E05,&H81:POKE &H7E06,
&H0A 'CMPA #50A
43 POKE &H7E07,&H26:POKE &H7E08,
&H03:POKE &H7E09,&H4F 'BNE $7E0C
CLRA
44 POKE &H7E0A,&H20:POKE &H7E0B,
&H02:POKE &H7E0C,&H84:POKE &H7E0D,
&H7F 'BRA $7E0E ANDA #57F
45 POKE &H7E0E,&HA7:POKE &H7E0F,
&H80 'STA ,X+
46 POKE &H7E10,&H8C:POKE &H7E11,
&H62:POKE &H7E12,&H00 'CMPX #56
200
47 POKE &H7E13,&H25:POKE &H7E14,
&HEE:POKE &H7E15,&H39 'BCS $7E03
RTS
49 REM DELETE STEP 5050 TO KILL
THE FILTER.
60 H=PEEK(&HC004):L=PEEK(&HC005)
:DKON=256*H+L
100 CLS:PRINT@32," MS DOS TO COC
O TEXT FILE XFER"
105 PRINT" FOR SINGLE SIDED MS D
OS DISKS"
110 PRINT:PRINT" (C) MARTY GOODM
AN JAN 1, 1986":PRINT
115 PRINT"FOR EITHER 8 OR 9 SEC
/ TRK."
120 PRINT"ONLY ROOT DIRECTORY FI
LES CAN"
125 PRINT"BE CONVERTED. FORMAT
THE DISK"
130 PRINT"YOU WILL PUT THE MS DO
S FIES"
135 PRINT"ON USING THE COMMAND:
":PRINT
140 PRINT" FORMAT B: /1"
145 PRINT:PRINT" PLACE SINGLE SI
DED MS DOS DISK"
150 PRINT" IN DRIVE 1 AND TAP
ENTER";
190 IF INKEY$ <> CHR$(13) THEN G
OTO 190
200 REM READ IN FIRST SECTOR OF
FAT
210 POKE &HEA,2:POKE &HEB,1:POKE
&HEC,0:POKE &HED,2:POKE &HEE,FS
:POKE &HEF,0
220 EXEC DKON
230 IF PEEK(&HF0) <> 0 THEN GOTO 9
000
300 REM CHECK FOR 8 VS 9 SECTOR
PER TRACK
310 GN=0:GOSUB 15000:T=CV AND 15
320 TS=0
330 IF T=&HE THEN TS=8
340 IF T=&HC THEN TS=9
350 IF TS=8 THEN GOTO 400
360 IF TS=9 THEN GOTO 450
370 CLS:PRINT@256-29,"WRONG KIND
OF MS DOS DISK"
380 PRINT" TAP ENTER TO RESTART
PROGRAM"
390 IF INKEY$ <> CHR$(13) GOTO 390
395 GOTO 100
400 REM INPUT 8 SEC /TRK DIRECTO
RY
410 FOR N=4 TO 7
420 POKE &HED,N
430 POKE &HEE,DS+(2*N)-8
435 EXEC DKON
440 IF PEEK(&HF0) <> 0 THEN GOTO 9
000
445 NEXT N
447 GOTO 500
450 REM INPUT SECOND FAT SECTOR
AND ALL OF DIRECTORY
455 FOR N= 5 TO 9
460 POKE &HED,N
465 POKE &HEE,DS+(2*N)-12
470 EXEC DKON
475 IF PEEK(&HF0) <> 0 THEN GOTO 9
000
480 NEXT N
490 GOTO 500
500 REM PUT DIR ON SCREEN
510 K=0:LKS(0)=0:Z=0
515 REM LOOP
520 CLS:PRINT@8,"DIRECTORY LISTI
NG"
530 PRINT@64," FILENAME.EXT
SIZE"
540 GOSUB 13000
550 IF FE=0 THEN GOTO 700
555 IF Z=0 THEN GOTO 750
560 F$="M" 'MIDDLE OF DIR
570 PRINT@512-96," ENTER CHOIC
E NUMBER OR"
580 PRINT" UP OR DOWN ARROW TO
SEE"
590 PRINT" PREVIOUS OR NEXT CH
OICES";
595 F$="M"
600 Z=Z+1:LKS(Z)=K
610 A$=INKEY$
615 IF A$="" THEN GOTO 610
620 IF A$=CHR$(10) THEN GOTO 680
625 IF A$=CHR$(94) THEN GOTO 660
627 IF VAL(A$)=0 THEN GOTO 610
630 IF VAL(A$)>Q THEN GOTO 610
635 CLS:VV=VAL(A$)-1:GOSUB 16000
:GOSUB 17000
640 PRINT@32,"YOU HAVE CHOSEN:"
645 PRINT:PRINT NAM$,FZ
650 PRINT:PRINT"HIT ENTER TO PRO
CEED, OR"
652 PRINT"ANY OTHER KEY TO RETUR
N TO MENU."
656 A$=INKEY$:IF A$="" GOTO 656
657 IF A$=CHR$(13) THEN GOTO 200
0
659 Z=Z-1:K=LKS(Z):GOTO 515
660 IF FE=0 THEN GOTO 610
662 K=LKS(Z):GOTO 515
668 IF Z=1 THEN GOTO 610
682 Z=Z-2:K=LKS(Z):GOTO 515
700 PRINT@512-96," END OF DIRE
CTORY"
710 PRINT" ENTER SELECTION OR
DOWN ARROW"
720 PRINT" TO SEE PREVIOUS PAG
E";
730 GOTO 600
750 PRINT@512-96," TOP OF DIRE
CTORY"
760 PRINT" ENTER SELECTION OR
UP ARROW";
780 PRINT" TO SEE MORE ENTRIES
";
790 GOTO 600
2000 REM FILE TRANSFER SECTION
2020 HCLU=40*TS
2030 CLS:PRINT@32,"PUT COCO DISK
IN DRIVE 0"
2040 PRINT"AND INPUT A FILE NAME
"
2045 PRINT"USE UP TO 8 LETTERS A
ND"
2050 PRINT"DO NOT USE AN EXTENTI
ON!";
2060 PRINT:INPUT CFN$
2070 IF CFN$="" THEN GOTO 2030
2075 IF LEN(CFN$)>8 THEN GOTO 20
30
2080 OPEN "O",1,CFN$+ "/TXT:0"
2090 CURCLU=BC
2092 IF TS=8 THEN QQ=5
2093 IF TS=9 THEN QQ=7
2095 ZCLU=INT (FZ/512)+1
2097 CLS:PRINT@256-32," NOW TR
ANSFERRING THE FILE"
2098 PRINT@512-96,"CLUSTER TRACK
SECTOR #BYTES"
2100 FOR M=1 TO ZCLU
2110 IF CURCLU>HCLU THEN M=M+1:G
OTO 2260
2120 X=CURCLU+QQ:T=INT(X/TS):S=X
-TS*T+1
2130 PRINT@480+2,CURCLU:PRINT@4
80+20,FZ-(512*(M-1));
2140 GN=CURCLU:GOSUB 15000:CURCL
U=CV
2200 GOSUB 5000
2210 IF CURCLU=0 THEN GOTO 190
2220 IF CURCLU>HCLU THEN GOTO 22
50
2230 NEXT M
2240 GOTO 3000
2250 IF CURCLU=&HFFF THEN M=M+1
2260 NEXT
2270 GOTO 3000
3000 PRINT #1,CHR$(&H0D);
3005 PRINT#1,CHR$(&H1A);
3010 CLOSE
3015 PRINT@480+20," ";:PRINT
@480+20,LB;
3020 SOUND 100,10
3030 PRINT@256-32," TRAN
SFER DONE "
3040 PRINT" TAP ENTER TO TRANSFE
R MORE "
3043 PRINT" TAP ANY OTHER KEY TO
EXIT"
3045 A$=INKEY$
3050 IF A$="" THEN GOTO 3045
3060 IF A$=CHR$(13) THEN GOTO 10
0
3070 CLS:END
5000 REM INPUT SECTOR TO BUFFER
5010 PRINT@480+9,T:PRINT@480+15
,S;
5020 POKE &HEA,2:POKE &HEB,1:POK
E &HEC,T:POKE &HED,S:POKE &HEE,I
S:POKE &HEF,0
5030 EXEC DKON
5040 IF PEEK(&HF0) <> 0 THEN GOTO
9000
5050 EXEC &H7E00
5060 A$=""
5065 P=VARPTR(A$)
5070 POKE P,128
5075 FOR Y=0 TO 3
5080 Z=ID+Y*128
5085 GOSUB 5200
5090 POKE P+2,MSB:POKE P+3,LSB
5100 PRINT#1,A$:
5110 NEXT Y
5130 RETURN
5200 MSB=INT(Z/256)
5210 LSB=Z-MSB*256
5220 RETURN
9000 REM PRINT DISK I/O ERROR
9020 CLS:PRINT@256-32," DIS
K ERROR ... SORRY!"
9030 PRINT" TAP ENTER TO RESTART
PROGRAM"
9040 IF INKEY$ <> CHR$(13) GOTO 90
40
9050 GOTO 100
10000 REM GET INFO FROM DIR
10001 REM NT IS ENTRY NUMBER
10002 REM NAM$=FILENAME ON EXIT
10003 REM F1=7 IF INVALID ENTRY
10004 REM F1=0 IF VALID ENTRY
10005 REM F1=1 IF SUBDIR
10006 REM F1=2 IF KILLED FILE
10007 REM F1=3 IF FREE (END OF
DIR)
10008 REM F1=4 IF ENTRY POINTS
BEYOND THE BUFFER SPACE.
10009 REM BUFFER SPACE IS FROM
$6600 THRU $71FF (6 SECTORS)
10010 REM OR 96 TOTAL ENTRIES.
10011 RE A=ATTRIBUTE BYTE
10015 ZZZ$=CHR$(&HE5)+STRING$(7,
CHR$(&HF6))
10020 DLOC=DD+32*NT
10030 IF DLOC>&H71FF THEN F1=4:R
ETURN

```

continued on Page 06

# COMPARE

by E. Pozzi and K. Paterson

The aim is to compare two programs to check for correctness of bytes.

I use it mainly in conjunction with my EPROM burner. One can double check the original program with the burnt one on the chip.

The program is self prompting. Two compare modes are available FAST and SLOW. In the slow mode the spacebar will stop and start the display.

COMPARE/BAS is the driver program, COMPARE/BIN will be executed by the selection of the FAST mode. Input the source code and assemble it to Disk as Compare/Bin.

To sum up you will have COMPARE/BAS plus COMPARE/BIN on your disk, the ML program being loaded in automatically when you run the Basic program.

## The Listing: COMPARE

```

0 GOTO40
10 '*** COMPARE ***
20 '** BY E.POZZI @ K.PATERSON**
';
25 'FOR QUERIES TEL 07 2774414
30 '***** 1-8-85 *****
31 SAVE"COMPARE:3":END
40 GOTO760
50 CLEAR200,&H1FFF
60 LOADM"COMPARE
70 CLS:PRINT:PRINT"loadm YOUR PR
OGRAMS NOW TO BE COMPARED ABOVE
E HEX$ 2000 THEN <RUN> THIS PR
OGRAM AGAIN
80 DEL-80
90 CLS7:POKE360,1:POKE361,121:
SCREEN0,1
100 PRINT@35,"** E.POZZI & K.PAT
ERSON **";
110 PRINT@108," PROGRAMS ";
120 PRINT@169," TO BE COMPARED "
;
130 PRINT@229," MUST HAVE BEEN L
OADED ";
140 PRINT@290," ABOVE HEX 2
000 ";
150 PRINT@485," SPACEBAR TO CONT
INUE ";
160 S$=INKEY$:IF S$=CHR$(32)THEN
180
170 GOTO 160
180 CLS7:PRINT@40," RAM COMPARAT
OR ";
190 PRINT@237," ENTER ";
200 PRINT@322," DEFAULT=&H20
00 ";:PRINT@290," ";:LINE IN
PUT "FIRST START ADDRESS &H?
";A$:PRINT@317,CHR$(239):PRINT@3
18,CHR$(239):PRINT@319,CHR$(239)
;
210 IF A$="" THEN A$="2000"
220 A$(1)=LEFT$(A$,2)
230 A$(2)=RIGHT$(A$,2)

```

```

240 A=VAL("&H"+A$(1))
250 B=VAL("&H"+A$(2))
260 POKE&HE19,A:POKE&HE1A,B
270 PRINT@290,STRING$(28,32);
280 PRINT@322," DEFAULT=&H40
00 ";:PRINT@290," ";:LINE I
NPUT "FIRST END ADDRESS &H?
";B$:PRINT@317,CHR$(239):PRINT@3
18,CHR$(239):PRINT@319,CHR$(239)
;
290 IF B$="" THEN B$="4000"
300 B$(1)=LEFT$(B$,2)
310 B$(2)=RIGHT$(B$,2)
320 C=VAL("&H"+B$(1))
330 D=VAL("&H"+B$(2))
340 POKE&HE56,C:POKE&HE57,D
350 PRINT@290,STRING$(28,32);
360 PRINT@322," DEFAULT=&HCO
00 ";:PRINT@290,"";:LINE INP
UT "SECOND START ADDRESS &H
?";C$:PRINT@318,CHR$(239):PRINT@
319,CHR$(239);
370 IFC$="" THEN C$="C000"
380 C$(1)=LEFT$(C$,2)
390 C$(2)=RIGHT$(C$,2)
400 E=VAL("&H"+C$(1))
410 F=VAL("&H"+C$(2))
420 POKE&HE16,E:POKE&HE17,F
430 PRINT@290,STRING$(28,32);
440 G=VAL("&H"+A$):H=VAL("&H"+B$
):I=VAL("&H"+C$)
450 PRINT@321,STRING$(30,42);:PR
INT@289," (F) FOR FAST OR (S) FO
R SLOW ";:PRINT@319,CHR$(239);
460 X$=INKEY$:IF X$=CHR$(70)
THEN 490
470 IF X$=CHR$(83) THEN 510
480 GOTO 460
490 EXEC&HE00
500 END
510 POKE65495,0:CLS7:PRINT@40,"
RAM COMPARATOR ";
520 P=100:Q=106:R=115:S=121
530 L=100:Z=115:FOR K=1TO10:PRIN
T@L,STRING$(9,32);:PRINT@Z,STRIN

```

```

G$(9,32);:L=L+32:Z=Z+32:NEXT K
540 FOR Z=1 TO 10
550 PRINT@P," "HEX$(G)";";
560 PRINT@Q,HEX$(PEEK(G))" ";
570 PRINT@R," "HEX$(I)";";
580 PRINT@S,HEX$(PEEK(I))" ";
590 IFHEX$(PEEK(G))=HEX$(PEEK(I)
) THEN650
600 SOUND 100,3
610 PRINT@485," SPACEBAR TO CONT
INUE ";
620 X$=INKEY$:IF X$=CHR$(32) THE
N 640
630 GOTO 620
640 PRINT@485,STRING$(22,32);
650 J$=INKEY$:IF J$=CHR$(32)
THEN 600
660 P=P+32:Q=Q+32:R=R+32:S=S+32
670 G=G+1:I=I+1:NEXT Z
680 IF G>H THEN 750
690 FOR E=1 TO 10
700 PRINT@P,STRING$(9,32);
710 PRINT@R,STRING$(9,32);
720 Q=Q-32:S=S-32
730 P=P-32:R=R-32:NEXT E
740 GOTO 540
750 GOTO 750
760 PCLEAR1:GOTO50

```

## The Listing: COMPARE2

```

00010 **COMPARE PROGRAM PART2
00020 **BY E.POZZI
00030 ** 1/08/1985
00040 BREAK EQU $03
00050 SPCBAR EQU $20
00060 ORG $E00
00070 BEGIN LDD #FFFFF
00080 LDY #0400
00090 MORE STD ,Y++
00100 CMPY #0600
00110 BLO MORE
00120 LDX #MESS1
00130 LBSR SHOW

```

```

00150 LDU #C000
00160 LDX #S4000
00170 AGAIN LDY #S04C5
00180 TFR X,D
00190 JSR BYTBIT
00200 TFR B,A
00210 JSR BYTBIT
00220 LDB #S60
00230 STB ,Y+
00240 LDA ,X+
00250 STA <0000
00260 JSR BYTBIT
00270 LDY #S04D1
00280 TFR U,D
00290 JSR BYTBIT
00300 TFR B,A
00310 JSR BYTBIT
00320 LDB #S60
00330 STB ,Y+
00340 LDA ,U+
00350 STA <0001
00360 JSR BYTBIT
00370 LDA $0000
00380 SUBA $0001
00390 STA $0002
00400 BNE WAIT
00410 CONT CMPX #S6000
00420 BNE AGAIN
00430 RTS
00440 BYTBIT PSHS A
00450 LSRA
00460 LSRA
00470 LSRA
00480 LSRA
00490 JSR CONVRT
00500 JSR DISPLY
00510 PULSA
00520 ANDA #S0F

00530 JSR CONVRT
00540 JSR DISPLY
00550 RTS
00560 CONVRT CMPA #S0A
00570 BCC LETTER
00580 ADDA #S70
00590 RTS
00600 LETTER ADDA #S37
00610 RTS
00620 DISPLY STA ,Y+
00630 RTS
00640 SHOW LDY #S044A
00650 LDB #S0A
00660 LOOP1 LDA ,X+
00670 STA ,Y+
00680 DECB
00690 BNE LOOP1
00700 RTS
00710 MESS1 FCC /COMPARATOR/
00720 MESS2 FCC /PRESS/
00730 FCB $60
00740 FCC /SPACEBAR/
00750 FCB $60
00760 FCC /TO/
00770 FCB $60
00780 FCC /CONTINUE/
00790 WAIT PSHS X,DP,B,A
00800 BSR MUSIC
00810 LDX #MESS2
00820 LDY #S522
00830 LDB #S1A
00840 ROUND LDA ,X+
00850 STA ,Y+
00860 DECB
00870 BNE ROUND
00880 WAIT1 JSR [S000]
00890 CMPA #BREAK
00900 BNE CONT1

00910 JMP S0027
00920 CONT1 CMPA #SPCBAR
00930 BNE WAIT1
00940 LDY #S522
00950 LDB #S1A
00960 LDA #S60
00970 DO STA ,Y+
00980 DECB
00990 BNE DO
01000 PULS A,B,DP,X
01010 LBRA CONT
01020 *
01030 MUSIC ORCC #S50
01040 *
01050 LDA #S32
01060 STA $FF23
01070 LDA #SFA
01080 STA $FF22
01090 LDA #S36
01100 STA $FF23
01110 BEEP LDB #100
01120 OUTLP LDA #40
01130 INLP1 DECA
01140 BNE INLP1
01150 LDA #S02
01160 ORA $FF22
01170 STA $FF22
01180 LDA #40
01190 INLP2 DECA
01200 BNE INLP2
01210 LDA #SFD
01220 ANDA $FF22
01230 STA $FF22
01240 DECB
01250 BNE OUTLP
01260 ANDCC #SAF
01270 RTS
01280 END BEGIN

```

# The Great Transformation

continued from Page 34

```

10040 FB=PEEK(DLOC)
10060 IF FB=0 THEN F1=3:RETURN
10070 NAM$=""
10080 FOR N=0 TO 7
10090 NAM$=NAM$+CHR$(PEEK(DLOC+N))
10100 NEXT N
10104 IF NAM$=ZZZ$ THEN F1=3:RET
URN
10105 IF FB=&HE5 THEN F1=2 :RETU
RN
10110 NAM$=NAM$+"."
10120 FOR N=8 TO 10
10130 NAM$=NAM$ +CHR$(PEEK(DLOC+
N))
10140 NEXT N
10145 F1=0
10150 A=PEEK(DLOC+11)
10155 T=A AND &H10:IF T<>0 THEN
F1=1
10160 T=A AND &H08:IF T<>0 THEN
F1=7
10200 RETURN
11000 REM FOR GIVEN ENTRY NUMBER
11001 REM GET FILE SIZE (FZ)
11002 REM AND BEGIN CLSTR (BC)
11010 DLOC=DD+NT*32
11020 FZ=PEEK(DLOC+28)+PEEK(DLOC
+29)*256+PEEK(DLOC+30)*65536+PEE
K(DLOC+31)*65536*256
11030 BC =PEEK(DLOC+26)+PEEK(DLO
C+27)*256
11040 RETURN
13000 REM GET 8 DIR ENTRIES
13002 REM Q=VALID ENTRY COUNT
13003 REM K=KOUNT OF ALL ENTRIES

```

```

13004 REM FE=255 IF MORE NTRIES
13010 Q=0:FE=255
13013 SCST=128
13015 REM LOOP
13030 NT=K
13040 GOSUB 10000:GOSUB 11000
13050 IF F1<>0 GOTO 13200
13100 NTRYLC(Q)=DLOC
13107 PRINT@SCST+32*Q,Q+1
13110 PRINT@SCST+3+Q*32,NAM$
13115 PRINT@SCST+19+Q*32,FZ
13120 Q=Q+1
13200 K=K+1
13210 IF F1=3 THEN FE=0:RETURN
13220 IF DB+32*K>DEND THEN FE=0:
RETURN
13230 IF Q>7 THEN RETURN
13240 GOTO 13015
15000 REM READ FAT
15001 REM GN =CLUSTER ENTRY#
15002 REM GN=0 TO 400
15003 REM CV = CONTENTS OF THE
CLUSTER NUMBER REQUESTED
15010 GIN=INT(GN/2)
15020 GCN=3*GIN
15030 GF=GN-2*GIN
15040 B1=PEEK(FS*256+GCN)
15050 B2=PEEK(FS*256+GCN+1)
15055 B3=PEEK(FS*256+GCN+2)
15060 N1=(B1 AND &HF0)/16
15070 N3=(B2 AND &HF0)/16
15080 N5=(B3 AND &HF0)/16
15090 N2=B1 AND &HF
15100 N4=B2 AND &HF
15110 N6=B3 AND &HF
15120 IF GF=0 GOTO 15200

```

```

15150 CV=N3+N6*16+N5*256:RETURN
15200 CV=N2+N1*16+N4*256:RETURN
16000 DLOC=NTRYLC(VV):GOTO 10030
17000 DLOC=NTRYLC(VV):GOTO 11020

```

## Listing 2: STRIPFL

```

10 'LINEFEED STRIPPER
20 'BY DON HUTCHISON [70425,1225
]
30 'ADAPTED FROM A PROGRAM BY AR
T FLEXSER, MARCH 1986
40 '
50 'MODIFIES ASCII FILES BY DELE
TING THE LINEFEEDS
60 '
70 CLEAR 200,&H7DFE
80 FOR I=&H7E00 TO &H7E29: READ
P$:POKE I,VAL("&H"+P$): NEXT
90 DATA 8D,A,8D,1D,81,A,27,F8,8D
,9,20,F4,C6,1,D7,6F,7E,C5,97
100 DATA C6,2,D7,6F,AD,9F,A0,2,F
,6F,6E,9F,A0,2,D,70,27,4,F,6F
110 DATA 32,62,39
120 IF PEEK(&HC004)<>&HD6 THEN P
OKE &H7E12,&HC4 'For 1.1 ROM
130 CLS: PRINT: PRINT: PRINTTAB(
8)"LINEFEED STRIPPER": PRINT
140 LINEINPUT "NAME OF INPUT FIL
E: ";I$
150 LINEINPUT "NAME OF OUTPUT FI
LE: ";O$
160 OPEN "I",#1,I$
170 OPEN "O",#2,O$
180 EXEC &H7E00
190 CLOSE #2: CLOSE #1
200 END

```

## BARDEN'S BUFFER

# The Meaning of Life

By William Barden, Jr.

The game I'm about to describe is more than a frivolous pastime. For some, it's a challenge in assembly language. For others, the key to unlocking the secrets of evolution. And for yet others, it's a way of generating interesting patterns. It's called "Life," and is an ancient computer game dating back at least 15 years.

Actually, it's not a game in the sense of a person versus computer confrontation. It's more a challenge of finding out how the game works and what the limits are, if any.

### The Rules of Life

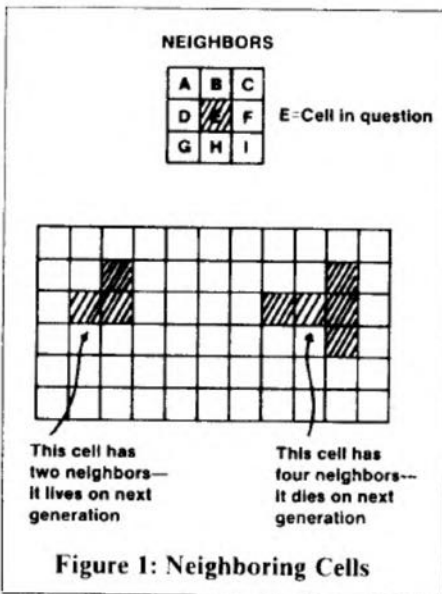
The rules were laid out by the game's inventor, mathematician John Horton Conway. Martin Gardner, of *Scientific American's* "Mathematical Games" fame, introduced it in his October 1970 column and provided periodic updates for several years thereafter. For the most part, the descriptions I'm providing come from the *Scientific American* columns. There's a recently published book on "Life" and other topics, called *The Recursive Universe* (William Poundstone, William Morrow and Company, 1985) that makes interesting reading.

The rules are deceptively simple. Start with a square matrix like a checkerboard. Each square of the checkerboard is called a cell. The checkerboard can be any width and any height, although something on the order of a CoCo screen (128 by 96) is a good size to start with. Put a pattern on the checkerboard by generating random points or entering points via a BASIC PSET command. The pattern defined is the starting generation.

Now consider each cell in the 128 by 96 matrix defined by the CoCo's screen. If a cell is on, it's considered to be living. Whether or not a cell survives until the next generation is dependent on its

immediate neighbors, the eight cells adjacent to the living cell, as shown in Figure 1. We'll call the neighbors A, B, C, D, F, G, H and I.

If the living cell has no neighbors or one neighbor, it dies from loneliness and disappears in the next generation. If the cell has two or three neighbors, it survives until the next generation. If the cell has more than three neighbors, it dies from overcrowding.



Not only can cells die, but new cells can be born. If an empty cell has three neighbors, the neighbors produce a new cell in the next generation. This only happens if an empty cell exists and there are three of the possible neighbors A, B, C, D, F, G, H and I.

Each generation is produced using these simple rules and there are an unlimited number of generations.

Here's an example. Start with a simple pattern such as the three cells in a straight line, as shown in Figure 2. The

result in generation one is a straight line of three cells at right angles to the first line. This pattern flips back and forth, oscillating in a style reminiscent of a blinker. To "Life" devotees, the pattern is called a blinker.

### The Appeal of Life

The interesting thing about "Life" is the unpredictability of the patterns produced. Start with a completely random pattern generated by:

```

100 PMODE 4,1
110 SCREEN 1,0
120 FOR I=1 TO 1400
130 PSET(RND(256),RND(192))
140 NEXT I
    
```

and apply the rules of "Life." You'll wind up with a situation analogous to life oozing up out of the primordial slime — a random pattern that produces some organisms that stay around forever and others that produce beautiful designs but die off after a dozen generations or so. Figure 3 shows the tenth generation of a "Life" game generated from 1,400 initial random points, grouped towards the screen center.

The rules of "Life" pose some interesting questions and they had programmers, computer scientists and mathematicians spending millions of dollars of computer time investigating "Life's" patterns. Are there patterns that move? Are there patterns that reproduce without limit? The answer to both of these questions is yes, but it's not immediately obvious to those watching the game for the first time.

### A High-Speed Life Generator

To study "Life," programmers and computer scientists use large mainframe computers and displays. One display allows 4,096 by 4,096 cells to be displayed at one time. As a matter of fact,

a dedicated system has been built to run "Life" at high-speed so the patterns can be observed and cataloged. Bear in mind this is not a project that is a profound breakthrough in artificial intelligence — it is primarily a fun thing that has some interesting implications. Still, "Life" has a large following.

#### A BASIC Life

Listing 1 shows "Life" implemented in BASIC. This BASIC program uses only a 14 by 20 element portion of the screen since BASIC is decidedly slow in producing the next generation. To use this program, enter a number for 'X' from zero to 19 and a number for 'Y' from zero to 13 to define the pattern, and watch the computation. It takes about 52 seconds to produce the next generation. The blinker pattern is defined by entering:

```
10,6
10,7
```

```
10,8
-1,-1
```

The two "-1" values terminate the entry and start "Life" processing.

#### An Assembly Language Life

A 14 by 20 matrix is really not big enough to see the interaction of the different patterns. The point of this column is a full-blown CoCo assembly language program to generate a 128 by 96 single-color version of "Life" on the CoCo (see Listing 2). Using the assembly language program, each generation of "Life" takes about 6.7 seconds. Still slow, but fast enough so you can easily watch the progress from generation to generation. I decided to use the lower resolution PMODE 0, instead of the maximum resolution PMODE 4, which requires four times the computation because there are four times the number of pixels.

#### The Program Algorithm

Having been through several versions of "Life" on different systems, I knew the program could never be fast enough. For that reason, I gave a lot of thought as to how the program should be implemented.

The first design consideration was the graphics screen. The graphics screen in Disk BASIC starts at location \$E00, as shown in Figure 4. If PMODE 0 is used, the resolution of the screen is 128 pixels wide by 96 pixels high. In PMODE 0 and in every single-color graphics mode, one bit is used to store the color for each pixel. A '1' in the bit means the foreground color is used; a '0' bit means the background color is used. Therefore, in PMODE 0 there's a total of 128 pixels/row x 96 rows = 12,288 pixels = 12,288 bits.

Of course, there are eight bits in a byte, so 12,288 bits/(8 bits/byte) = 1,536 bytes used to store each graphics screen.

The plan I use is to keep the PMODE 0,1 screen as the current screen and update a second screen, the PMODE 0,2 screen. (In the PMODE command, the second parameter specifies the page number, in this case '1' or '2'.)

To do this, I have it scan the current screen one cell (pixel) at a time. For each cell, a count of the neighbors is made, with the corresponding cell in the new screen set or reset according to the rules of "Life." The scan proceeds from right to left across each row, starting at the last row and ending on the first row, as shown in Figure 5.

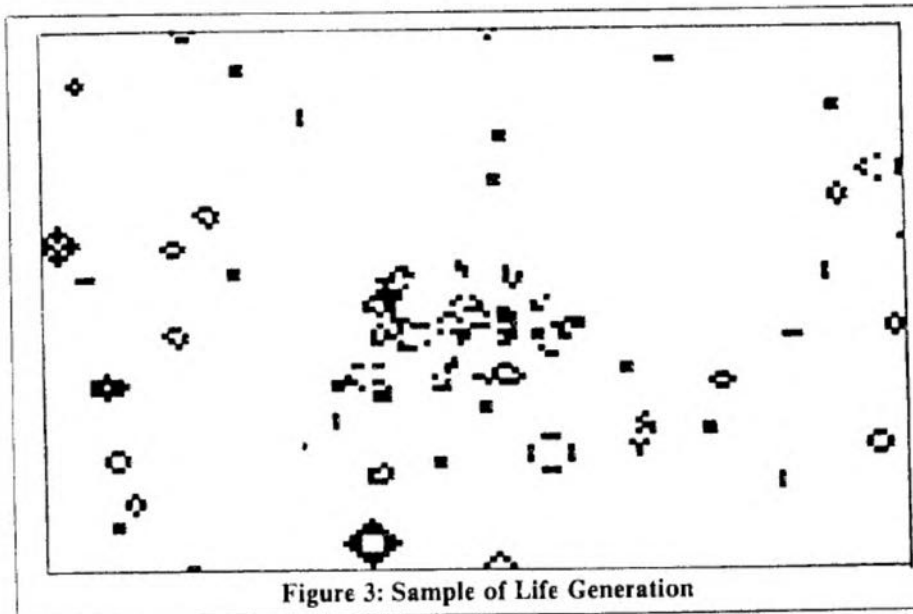
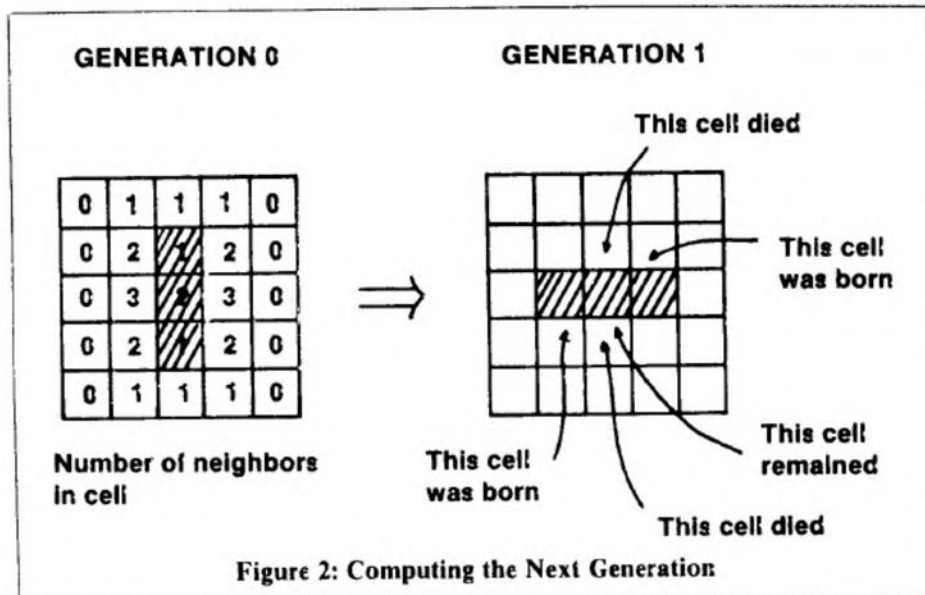
One other screen design consideration was what is to be done on the screen boundaries. There are two approaches to handling boundary conditions. One approach lets the living cells disappear beyond the screen edges. A second approach treats the whole screen as a "toroid" — a closed universe — as shown in Figure 6. The second approach is followed here. Patterns going off the right reappear on the left, patterns going off the top reappear on the bottom, and so forth.

After each screen scan of 12,288 cells, the second screen is written to the first screen by a quick assembly language subroutine. This makes the update appear almost immediately, avoiding a slow, partial screen update.

The program is divided into three parts: a main loop, a Count Neighbors subroutine and a Get Address subroutine.

#### Main Loop

The main loop scans through the first screen as shown in Figure 5, starting





with the last cell in Row 95 of Column 127. Rows are called 'Y' and are numbered from zero through 95. Columns are called 'X' and are numbered from zero through 127.

The current location is held in variable XY. Note that this is a two-byte variable. The first byte holds 'X' and the second byte holds 'Y'. This variable is initialized with X=127 and Y=95. The loop from MAN005 through the fifth instruction after MAN080 is the main loop of the program, done 128\*96 times to process each of the 12,288 cells. Each time through the loop, subroutine GETADD is called to calculate the addresses for the current XY. This is followed by a call to subroutine COUNTN to count the neighbors of the current cell.

After the call to COUNTN, a check is made of the current cell's on/off status. The byte location of the current

cell is held in ELOC, a 16-bit pointer set by GETADD. The bit position of the cell within the byte is held by EBIT, an eight-bit variable set by GETADD. A branch is made to MAN040 if the cell is empty.

If the cell is empty, a check of variable COUNT (set by COUNTN) is made. If COUNT=three, the code at MAN010 is called for the "birth." If COUNT< three, nothing is done.

If the current cell is not empty, a check is made of the number of neighbors. If COUNT=two or three, the corresponding cell in the second screen is set, otherwise the cell is reset. Again, pointer ELOC holds the address of the current cell, while EBIT holds the bit position of the cell within the byte retrieved. Variable EBITI is the inverted bit of the cell, set by GETADD. For example, if the current bit is represented by 0010000, EBITI holds 1101111. This

makes it easy to reset the bit.

At the end of the birth/death checks, the code at MAN080 decrements 'X' by one. If 'X' does not equal 1111111 (off the left edge of the screen), the next cell is considered. If 'X' is 1111111, 127 is stored in the first byte of XY for the 'X' value, and 'Y' is decremented by one. If 'Y' is not equal to 1111111, the next row above is considered.

When the 'Y' value is decremented down to 1111111, the last row has been processed and the new cells are in the second screen. The data in screen two is moved to screen one by the short move code starting near MAN085. This code is so fast the entire screen appears to change, even though the movement is done from top to bottom, a row at a time.

### Count Neighbors Subroutine

This subroutine counts the eight neighbors of the current cell. The result is COUNT, which holds a value of zero through eight and is used in the main loop to determine whether the cell lives or dies. The subroutine uses two tables. One of the tables starts at ALOC and is the byte location table. This table is established by the GETADD subroutine and holds the byte address of each of the neighbors of the current cell, the neighbors being the cell up and to the left, directly above, up and to the right, the cell directly left, and so forth.

The second table starts at ABIT and holds the bit configuration that defines the neighbor bit within the byte pointed to by the ALOC table. For example, if the current cell is defined by 00010000, the neighbor to the left is defined by 0010000 and the neighbor to the right by 00001000.

The Count Neighbors subroutine goes down through both tables, using the ALOC table entry to point to the byte containing the cell and the ABIT table entry to strip off the proper bit, which is counted if it is a one. The current cell is defined by ELOC and EBIT, and is beyond the end of both tables so that the subroutine only counts neighbors.

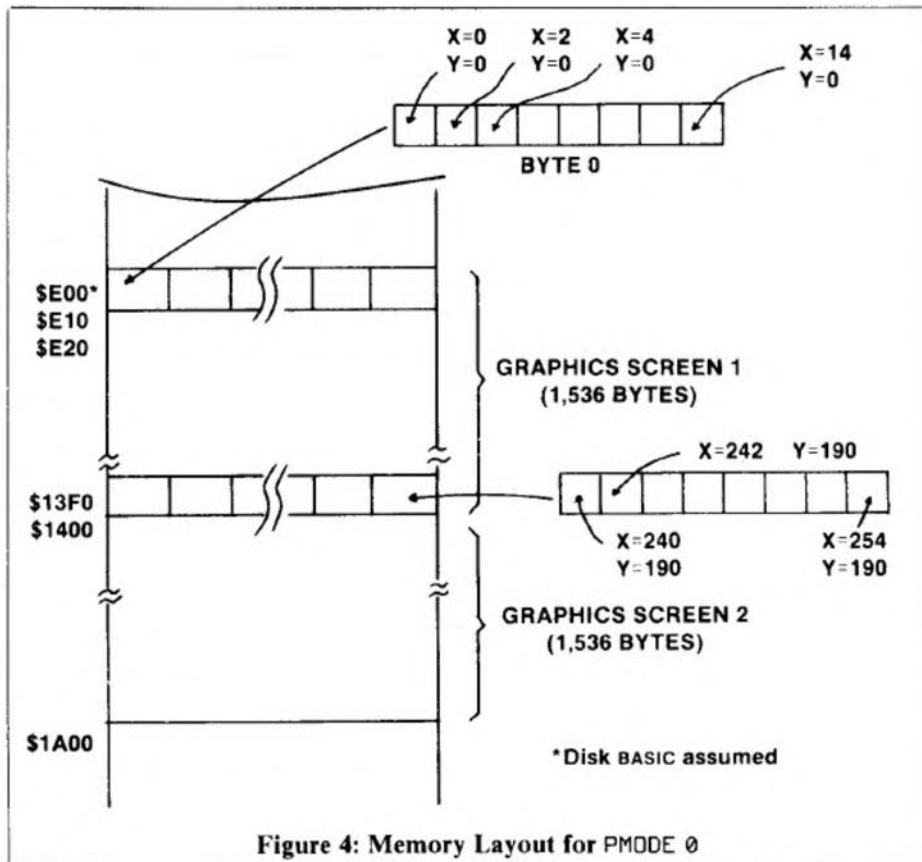


Figure 4: Memory Layout for PMODE 0

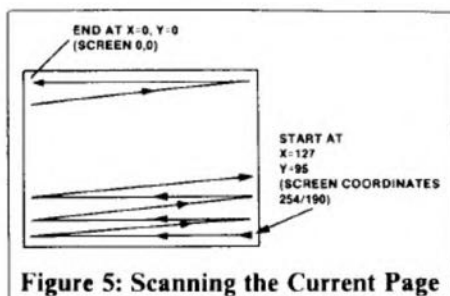


Figure 5: Scanning the Current Page

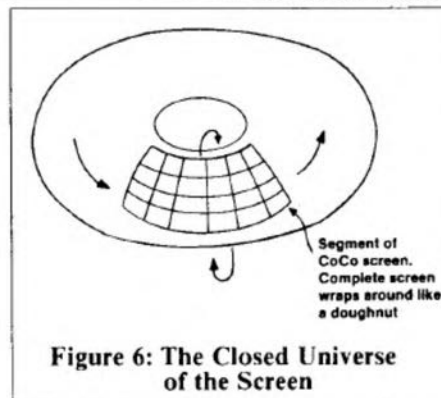


Figure 6: The Closed Universe of the Screen

### Get Address Subroutine

Most of the work in the program is done by the Get Address subroutine. It establishes the addresses in the ALOC table and the bit positions in the ABIT table. The graphics data for all 128 by 96 bits, remember, is represented by one bit somewhere within the 1,536 bytes of screen one. The first byte represents Y=0 and X's of 0, 1, 2, 3, 4, 5, 6 and 7; the next byte represents Y=0 and X's of 8, 9, 10, 11, 12, 13, 14 and so

on.

The subroutine locates the byte containing the current X,Y by multiplying the 'Y' value by 16, as there are 16 bytes per row. The 'X' value is then divided by eight and added to the Y\*16 value.

Eight is used as a divisor because there are eight cells per byte. The division is done by three consecutive shifts — it's equivalent to a BASIC INT function. The actual address in screen one is then computed by adding \$E00 to Y\*16+INT(X/8). This byte address is stored in DLOC, ELOC and FLOC the table locations for the current X,Y and its two neighbors on the same row.

The ALOC, BLOC, and CLOC locations in the preceding row can be found by subtracting 16 from the locations for the current row. The result is put into the three table entries. Similarly, the GLOC, HLOC and ILOC locations are found by adding 16 to the current row locations.

These table entries are valid providing one of the three rows isn't off the top or bottom (or another condition, which we'll discuss shortly). Checks are made for this, and 1,536 is either added or subtracted from the row to point to the wrap-around row from the other side of the screen, which gives the toroidal effect.

The code from location GET022 is used to compute and store the bit position within the byte to be accessed by the ALOC entry. This location is determined by the three least significant bytes of 'X'. If X=XXXXX000, for example, the bit position is 10000000; if X=XXXXX001, the bit position is 01000000, and so forth, up to X=XXXXX111, where the bit position is 00000001.

The cell mask values are contained within a cell mask table at location MASK. The entries in ABIT are initialized such that the current 'Y' values (B, E and H) get the mask table value, the 'Y' locations to the left get the bit position values with the bit shifted left one bit, and the Y locations to the right get the bit position values with the bit shifted right one bit. If the bit position for the current cell is 00010000, for example, ABIT, DBIT and GBIT get 00100000, and CBIT, FBIT and IBIT get 00001000.

The last part of the GETADD subroutine adjusts the ALOC table for the boundary conditions in cases where either the current 'Y' involves two bytes or the edge of the screen (left or right) has been encountered. If the current 'Y' is at the left bit of a byte, for example, the bit mask is 10000000. The left neighbor's bit mask is 00000001 in this

case (the right neighbor's bit mask is 01000000).

This means the left neighbor's byte location should be one less than the location stored. This check is made and the byte location adjusted in the two cases where the current 'Y' is at either end of the byte. A check is also made for the left and right edges, and 15 is added or subtracted to get the proper wrap-around byte in this case.

To avoid computing addresses for every X,Y, the byte address calculation is done only for X's that represent a bit at either the left or right end of the byte. These are the only cases where two bytes are involved. If the bit is in bit position one (01000000) through five (00000101), the prior ALOC addresses apply, and the address portion of GETADD is skipped at the beginning of the subroutine. The effect is to speed up the

the &H3E00 area by a PDKE loop before the program is executed.

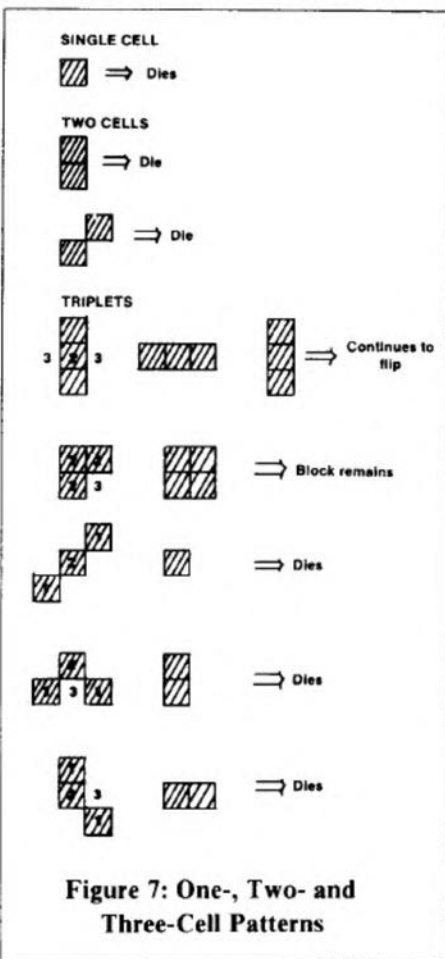


Figure 7: One-, Two- and Three-Cell Patterns

subroutine for two-thirds of the cells, knocking about 14 percent off the screen update rate.

#### A BASIC Driver

Listing 3 shows the BASIC driver program that implements the assembly language "Life" program. It has the assembly language machine language code embedded in it as a series of DATA values. The DATA values are relocated to

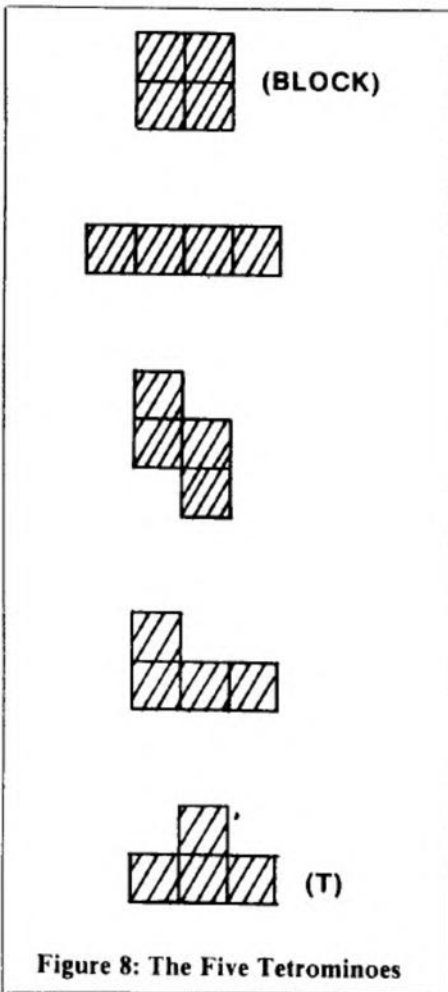


Figure 8: The Five Tetrominoes

The BASIC driver has provision for either entering a set of points for the initial "Life" pattern, or for generating a number of random points at the screen center. To enter a set of points, enter 'S' after the USER PNTS (S) OR RANDOM (R)? prompt message.

The program then asks for the X,Y position of the point: X,Y? Enter as many X,Y values as you want, and enter -1,-1 to terminate the entry. The X,Y points *must* be even numbers due to the half resolution of PMODE 0. For example, adjacent points are 100/100, 100/102 and 100/104.

To use a set of random points, enter R after the Set/Random message. The program asks for the number of points

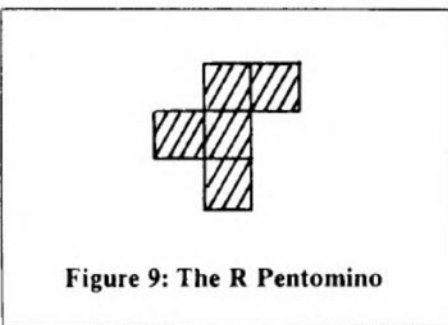


Figure 9: The R Pentomino

to use: NUMBER OF POINTS?

Too few points here, and the second generation virtually disappears, leaving only a few points that disappear on the next generation. If the entire screen is filled with points, the next generation disappears entirely.

#### Sample Patterns

You could systematically investigate all patterns and their succeeding generations in "Life," — much work has already been done in this area. The simplest pattern is a single cell (Figure 7), which dies in the next generation from loneliness. The next simplest patterns are two adjacent cells, either horizontally, vertically or diagonally. These also die in the next generation.

The next set of patterns are triplets. There are five ways in which three cells can be combined, as shown in Figure 7. Since the L-shape cells have two neighbors, they endure until the next generation. In addition, the cell in the nook of the 'L' is born. The resulting block is a stable life form — what Conway calls a still life — it never changes. We've seen the three cells in a straight line before; they change to a line at right angles on the next generation, a so-called blinker. The blinker flips back and forth from generation to generation. The other configurations die after one generation.

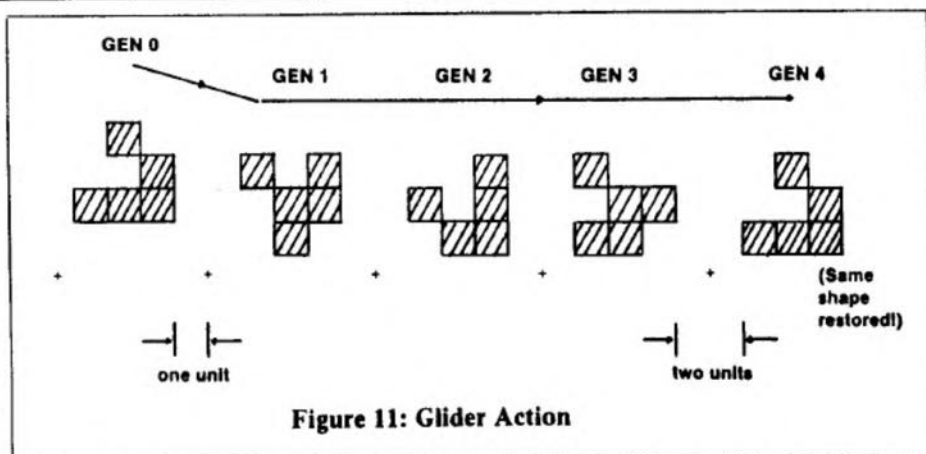


Figure 11: Glider Action

The next set of cells are made of four cells connected together. The game's jargon for these patterns is "tetromino," tetra meaning four. There are five ways that four cells can be connected, as shown in Figure 8. The block configuration is stable, as we've seen. The 'T' tetromino looks as if it will grow larger and larger, but stabilizes after nine generations into a "traffic light," a series of four blinkers.

The straight line tetromino turns into a block of six cells on the next generation, which in turn creates a "beehive," another unchanging pattern. The two remaining tetrominoes also produce beehives.

So far, "Life" isn't too exciting, but we've only considered four sets of

patterns. There is an infinite number to go!

The next set of patterns is formed by connecting five cells, "pentomino" shapes. One of the most interesting of these is the 'R' pentomino, shown in Figure 9. The 'R' pentomino seems to grow without bound, scattering debris all over the screen. However, after dozens of generations in our toroidal universe, the life forms settle down to simple patterns that are either still life or blinkers. An intermediate screen is shown in Figure 10.

#### Ah, Sweet Mystery of Life

Is there any configuration of cells that grows without bound, forever? Experimentation in this area produces a shape known as a glider. It glides across the screen (Figure 11). The glider can be generated by a glider gun, a complex arrangement of patterns that goes on producing gliders forever. (It's also seen in the 'R' pentomino patterns.) There are other patterns that replicate themselves as well.

It's a lot of fun to start with a pattern of your own design (you can do this by slightly altering the BASIC program) and watch what happens. At the very least, you'll see generations of interesting "Life" forms.

For more reading on this, get Poundstone's book or try to get the original *Scientific American* articles. For help, contact me at P.O. Box 3568, Mission Viejo, CA 92692.

Next month I'll be back with more assembly language topics. In the meantime, keep assembling! □

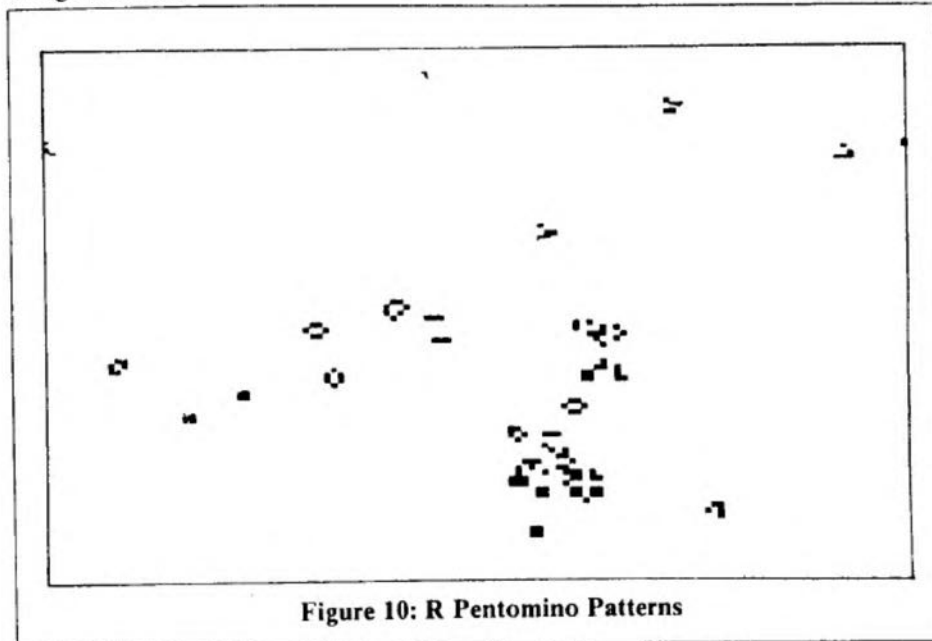


Figure 10: R Pentomino Patterns

# RAINBOW ON DISK \$ 15

**Listing 1: SLOWLIFE**

```

100 'RUDIMENTARY LIFE 20 X 14
110 DIM A(13,19)
120 'CLEAR ARRAY
130 FOR Y=0 TO 13:FOR X=0 TO 19
140 A(Y,X)=32
150 NEXT X: NEXT Y
160 CLS
170 'READ IN INITIAL VALUES
180 PRINT@448,"";
190 INPUT "X,Y:";X,Y
200 IF X=-1 THEN 240 ELSE PRINT
    @Y*32+X,"0";: A(Y,X)=79
210 PRINT@452," ";
220 GOTO 180
    
```

```

230 'MAIN LOOP
240 FOR Y=0 TO 13: FOR X=0 TO 19
250 XL=X-1: IF XL=-1 THEN XL=19
260 XP=X+1: IF XP=20 THEN XP=0
270 YL=Y-1: IF YL=-1 THEN YL=13
280 YP=Y+1: IF YP=14 THEN YP=0
290 'FIND # OF NEIGHBORS
300 NO=0
310 IF PEEK(&H4000+YL*32+XL) <> 96
    THEN NO=NO+1
320 IF PEEK(&H4000+YL*32+X) <> 96
    THEN NO=NO+1
330 IF PEEK(&H4000+YL*32+XP) <> 96
    THEN NO=NO+1
340 IF PEEK(&H4000+Y*32+XL) <> 96
    THEN NO=NO+1
350 IF PEEK(&H4000+Y*32+XP) <> 96
    
```

```

THEN NO=NO+1
360 IF PEEK(&H4000+YP*32+XL) <> 96
    THEN NO=NO+1
370 IF PEEK(&H4000+YP*32+X) <> 96
    THEN NO=NO+1
380 IF PEEK(&H4000+YP*32+XP) <> 96
    THEN NO=NO+1
390 IF PEEK(&H4000+Y*32+X) <> 96
    THEN IF NO=2 OR NO=3
        THEN A(Y,X)=79 ELSE A(Y,X)=
        32: GOTO 410
400 IF NO=3 THEN A(Y,X)=79
410 NEXT X: NEXT Y
420 'PRINT NEXT GENERATION
430 FOR Y=0 TO 13: FOR X=0 TO 19
440 PRINT@Y*32+X,CHR$(A(Y,X));
450 NEXT X: NEXT Y
460 GOTO 240
    
```

**Listing 2: FASTLIFE**

```

00100 *****
00110 * HIGH-SPEED LIFE IN 128 BY 96 PIXELS *
00120 *****
00130 *
00140 * MAIN LOOP
00150 *
00160 ORG $3E00
00170 LDD #127*256+95
00180 STD XY INITIALIZE X,Y
00190 MANG03 LDD XY GET X,Y
00200 LBSR GETADD FIND ALL ADDRESSES
00210 BSR COUNTN COUNT NEIGHBORS
00220 LDA [ELOC] GET BYTE
00230 ANDA EBIT GET CELL
00240 BEQ MANG40 GO IF EMPTY
00250 * LIVING CELL HERE
00260 LDA COUNT GET COUNT
00270 ANDA #0E THIS TRICK
00280 EORA #2 TESTS FOR 2 OR 3
00290 BEQ MANG10 GO IF 2 OR 3
00300 * ON AND NOT 2 OR 3
00310 MANG07 LDX ELOC POINT TO BYTE
00320 LDA 1536,X GET BYTE FROM NEXT
00330 ANDA EBITI RESET BIT - DEATH!
00340 STA 1536,X STORE BYTE
00350 BRA MANG08 GO FOR NEXT CELL
00360 * ON AND 2 OR 3
00370 MANG10 LDX ELOC POINT TO BYTE
00380 LDA 1536,X GET BYTE
00390 ORA EBIT BIRTH OR STAY ALIVE
00400 STA 1536,X STORE BYTE
00410 BRA MANG08 GO TO NEXT GEN
00420 * EMPTY HERE
00430 MANG40 LDA COUNT GET COUNT
00440 CMPA #3 TAKES 3 TO TANGO
00450 BEQ MANG10 GO IF BIRTH!
00460 * PREPARE FOR NEXT CELL
00470 MANG08 DEC XY DECREMENT X
00480 BPL MANG05 GO IF 0 - 7F
00490 LDA #127 RESET X
00500 STA XY STORE IN X
00510 DEC XY+1 DECREMENT Y
00520 BPL MANG05 GO IF 0 - 6F
00530 LDX #0E00 POINT TO PAGE 1
00540 LDY #0E00+1536 POINT TO PAGE 2
00550 MANG85 LDD ,Y++ GET WORD
00560 STD ,X++ STORE IN PAGE 1
00570 CMPX #0E00+1536 AT END?
00580 BNE MANG85 GO OF NO
00590 RTS RETURN TO BASIC
00600 *****
00610 * COUNT NEIGHBORS SUBROUTINE *
00620 *****
00630 COUNTN CLRBS SET COUNT TO 0
00640 LDX #ALOC POINT TO TABLE START
00650 LDY #ABIT POINT TO TABLE START-1
00660 CNT05 LDA [,X++] GET BYTE WITH CELL
00670 ANDA ,Y+ TEST BIT
00700 BEQ CNT10 GO IF NO CELL
00710 INCB BUMP COUNT
00720 CNT10 CMPX #ILOC+2 AT END OF TABLES?
00730 BNE CNT05 GO IF NO
00740 STB COUNT STORE COUNT
00750 RTS RETURN
    
```

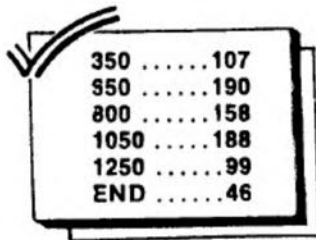
```

00760 *****
00770 * GET ADDRESS SUBROUTINE. GETS NINE ADDRESSES*
00780 *****
00790 GETADD PSHS A SAVE X
00800 ANDA #7 TEST BITS
00810 BEQ GET015 GO IF 000
00820 ANDA #6 IGNORE LSB
00830 EORA #6 TEST FOR 110 OR 111
00840 BNE GET022 BYPASS LOC COMP IF 001-101
00850 GET015 LDA #16 16 BYTES PER ROW
00860 MUL 16*X
00870 TFR D,X FOR NEXT ADD
00880 LDB ,S GET X
00890 LSRB X/2
00900 LSRB X/4
00910 LSRB X/8
00920 ABX 16*Y+INT(X/8)-DISP
00930 LEAX $000,X $000+16*Y+INT(X/8)
00940 STX DLOC CURRENT ROW
00950 STX ELOC
00960 STX FLOC
00970 TFR X,D FOR COMPUTATIONS
00980 SUBD #16 FOR PREV ROW - MAY BE MODS
00990 CMPD #0E00 ABOVE TOP?
01000 BHS GET011 GO IF NO
01010 ADDD #1536 WRAP AROUND FROM BOTTOM
01020 GET011 STD ALOC PREVIOUS ROW
01030 STD BLOC
01040 STD CLOC
01050 ADDD #32 FOR NEXT ROW
01060 CMPD #0E00+1535
01070 BLS GET020 GO IF NOT OFF BOTTOM
01080 SUBD #1536 WRAP AROUND FROM TOP
01090 GET020 STD GLOC NEXT ROW
01100 STD HLOC
01110 STD ILOC
01120 GET022 PULS B GET X
01130 ANDB #7 0-7
01140 LDX #MASK MASK TABLE ADDR
01150 ABX POINT TO MASK
01160 LDA ,X GET MASK
01170 STA BBIT CURRENT COLUMN
01180 STA EBIT
01190 STA HBIT
01200 COMA GET INVERTED BITS
01210 STA EBITI SAVE FOR RESET BIT
01220 LDA -1,X GET LEFT MASK
01230 STA ABIT PREVIOUS COLUMN
01240 STA DBIT
01250 STA GBIT
01260 LDA +1,X GET RIGHT MASK
01270 STA CBIT NEXT COLUMN
01280 STA FBIT
01290 STA IBIT
01300 * POSSIBLE ADJUST OF LOC'NS ON BOUNDARIES
01310 LDA ABIT GET LEFT MASK
01320 CMPA #1 IS IT LSB?
01330 BNE GET030 GO IF NO
01340 * TWO-BYTE CASE HERE
01350 LDX #0
01360 LDB ALOC+1 FINAGLE FACTOR
01370 ANDB #0F GET BYTE LOC'N LSB
01380 BNE GET025 GO IF NOT OFF LEFT
01390 LEAX +16,X OFF EDGE
01400 GET025 LEAX -1,X ADJUST IN EITHER CASE
01410 PSHS X STORE FOR COMPUTATION
01420 LDD ALOC ADJUST LEFT LOC'NS
    
```

```

#143# ADDD ,S
#144# STD ALOC
#145# LDD DLOC
#146# ADDD ,S
#147# STD DLOC
#148# LDD GLOC
#149# ADDD ,S++
#150# STD GLOC
#151# GET#3# LDA CBIT GET RIGHT MASK
#152# BPL GET#4# GO IF NOT $# CASE
#153# * TWO-BYTE CASE HERE
#154# LDX ## FINAGLE FACTOR
#155# LDB CLOC+1 GET BYTE LOC'N LSB
#156# ANDB #F# 16-BYTE BOUNDARY
#157# CMPB #F# TEST FOR OFF RIGHT
#158# BNE GET#3# GO IF NOT OFF
#159# LEAX -16,X OFF EDGE
#160# GET#3# LEAX +1,X ADJUST IN EITHER CASE
#161# PSHS X STORE FOR CALCULATIONS
#162# LDD CLOC ADJUST RIGHT EDGE LOC'NS
#163# ADDD ,S
#164# STD CLOC
#165# LDD FLOC
#166# ADDD ,S
#167# STD FLOC
#168# LDD ILOC
#169# ADDD ,S++
#170# STD ILOC
#171# GET#4# RTS RETURN
#172# * CELL MASK TABLE
#173# FCB 1
#174# MASK FCB 128
#175# FCB 64
#176# FCB 32
#177# FCB 16
#178# FCB 8
#179# FCB 4
#180# FCB 2
#181# FCB 1
#182# FCB 128
#183# * BYTE LOCATION TABLE
#184# * A*B*C
#185# * D*E*F
#186# * G*H*I
#187# ALOC FDB #
#188# BLOC FDB #
#189# CLOC FDB #
#190# DLOC FDB #
#191# FLOC FDB #
#192# GLOC FDB #
#193# HLOC FDB #
#194# ILOC FDB #
#195# ELOC FDB #
#196# * MASK TABLE FOR NEIGHBORS
#197# ABIT FCB #
#198# BBIT FCB #
#199# CBIT FCB #
#200# DBIT FCB #
#201# FBIT FCB #
#202# GBIT FCB #
#203# HBIT FCB #
#204# IBIT FCB #
#205# EBIT FCB #
#206# EBITI FCB #
#207# * WORKING VARIABLES
#208# COUNT FCB #
#209# XY FCB #
#210# END
INVERTED BITS
COUNT OF NEIGHBORS
CURRENT X, Y

```



**Listing 3: DRIVER**

```

100# ' HIGH-SPEED LIFE BASIC DVR
110# CLEAR 200, &H3DFF
120# FOR I=&H3E00 TO &H3F65
130# READ A: POKE I, A
140# NEXT I
150# DEFUSR = &H3E00
160# DIM PT(400)
170# CLS
180# PRINT @15, "LIFE"
190# INPUT "USER PNTS (S) OR
RANDOM (R)"; AS
200# IF AS="S" THEN 270
210# INPUT "NUMBER OF POINTS"; NP
220# FOR I=1 TO NP
230# PT(I)=(100+RND(56))*256+85+
RND(36)
240# NEXT I
250# I=I-1
260# GOTO 360
270# I=#
280# PRINT "ENTER X, Y (-1=END) "
290# PRINT @448, "X, Y";
300# INPUT X, Y
310# IF X=-1 THEN 360
320# PT(I+1)=X*256+Y
330# I=I+1
340# PRINT @448, " ";
350# GOTO 290
360# FOR J=1 TO 2
370# PMODE #, J
380# PCLS
390# NEXT J
400# PMODE #, 1
410# SCREEN 1, #

```

```

420# FOR J=1 TO I
430# PSET (INT (PT(J)/256), PT(J)-
INT (PT(J)/256)*256)
440# NEXT J
450# GN=1
460# A=USR#(0)
470# GN=GN+1
480# GOTO 460
490# DATA &HCC, &H75, &H5F, &HFD
500# DATA &H3F, &H83, &HFC, &H3F
510# DATA &H83, &H17, &H00, &H73
520# DATA &H8D, &H59, &HA6, &H9F
530# DATA &H3F, &H76
540# DATA &HB4, &H3F, &H80, &H27
550# DATA &H29, &HB6, &H3F, &H82
560# DATA &H84, &H0E, &H88, &H02
570# DATA &H27, &H10, &HBE, &H3F
580# DATA &H76, &HA6, &H89, &H06
590# DATA &H00, &HBA, &H3F, &H81
600# DATA &HA7, &H89, &H06, &H00
610# DATA &H20, &H17, &HBE, &H3F
620# DATA &H76, &HA6, &H89, &H06
630# DATA &H00, &HBA, &H3F, &H80
640# DATA &HA7, &H89, &H06, &H00
650# DATA &H20, &H07, &HB6, &H3F
660# DATA &H82, &H81, &H03, &H27
670# DATA &HE9, &H7A, &H3F, &H83
680# DATA &H2A, &HBA, &H86, &H7F
690# DATA &HB7, &H3F, &H83, &H7A
700# DATA &H3F, &H84, &H2A, &HB0
710# DATA &H8E, &H0E, &H00, &H10
720# DATA &H8E, &H14, &H00, &HEC
730# DATA &HA1, &HED, &H81, &H8C
740# DATA &H14, &H00, &H26, &HF7
750# DATA &H39, &H5F, &H8E, &H3F
760# DATA &H66, &H10, &H8E, &H3F
770# DATA &H78, &HA6, &H91, &HA4
780# DATA &HA0, &H27, &H01, &H5C
790# DATA &H8C, &H3F, &H76, &H26
800# DATA &HF4, &HF7, &H3F, &H82
810# DATA &H39, &H34, &H02, &H84
820# DATA &H07, &H27, &H06, &H84
830# DATA &H06, &H88, &H06, &H26
840# DATA &H44, &H86, &H10, &H3D
850# DATA &H1F, &H01, &HE6, &HE4
860# DATA &H54, &H54, &H54, &H3A
870# DATA &H30, &H89, &H0E, &H00
880# DATA &HBF, &H3F, &H6C, &HBF
890# DATA &H3F, &H76, &HBF, &H3F
900# DATA &H6E, &H1F, &H10, &H83
910# DATA &H00, &H10, &H10, &H83
920# DATA &H0E, &H00, &H24, &H03
930# DATA &HC3, &H06, &H00, &HFD
940# DATA &H3F, &H66, &HFD, &H3F
950# DATA &H68, &HFD, &H3F, &H6A
960# DATA &HC3, &H00, &H20, &H10
970# DATA &H83, &H13, &HFF, &H23
980# DATA &H03, &H83, &H06, &H00
990# DATA &HFD, &H3F, &H70, &HFD
1000# DATA &H3F, &H72, &HFD, &H3F
1010# DATA &H74, &H35, &H04, &HC4
1020# DATA &H07, &H8E, &H3F, &H5D
1030# DATA &H3A, &HA6, &H84, &HB7
1040# DATA &H3F, &H79, &HB7, &H3F
1050# DATA &H80, &HB7, &H3F, &H7E
1060# DATA &H43, &HB7, &H3F, &H81
1070# DATA &HA6, &H1F, &HB7, &H3F
1080# DATA &H78, &HB7, &H3F, &H7B
1090# DATA &HB7, &H3F, &H7D, &HA6
1100# DATA &H01, &HB7, &H3F, &H7A
1110# DATA &HB7, &H3F, &H7C, &HB7
1120# DATA &H3F, &H7F, &HB6, &H3F
1130# DATA &H78, &H81, &H01, &H26
1140# DATA &H29, &H8E, &H00, &H00
1150# DATA &HF6, &H3F, &H67, &HC4
1160# DATA &H0F, &H26, &H03, &H30
1170# DATA &H88, &H10, &H30, &H1F
1180# DATA &H34, &H10, &HFC, &H3F
1190# DATA &H66, &HE3, &HE4, &HFD
1200# DATA &H3F, &H66, &HFC, &H3F
1210# DATA &H6C, &HE3, &HE4, &HFD
1220# DATA &H3F, &H6C, &HFC, &H3F
1230# DATA &H70, &HE3, &HE1, &HFD
1240# DATA &H3F, &H70, &HB6, &H3F
1250# DATA &H7A, &H2A, &H2A, &H8E
1260# DATA &H00, &H00, &HF6, &H3F
1270# DATA &H6B, &HC4, &H0F, &HC1
1280# DATA &H0F, &H26, &H02, &H30
1290# DATA &H10, &H30, &H01, &H34
1300# DATA &H10, &HFC, &H3F, &H6A
1310# DATA &HE3, &HE4, &HFD, &H3F
1320# DATA &H6A, &HFC, &H3F, &H6E
1330# DATA &HE3, &HE4, &HFD, &H3F
1340# DATA &H6E, &HFC, &H3F, &H74
1350# DATA &HE3, &HE1, &HFD, &H3F
1360# DATA &H74, &H39, &H01, &H80
1370# DATA &H40, &H20, &H10, &H08
1380# DATA &H04, &H02, &H01, &H80

```

# FORUM

by John Poxon

This month I'll talk about LOOPS.

In basic it is possible to have FOR TO NEXT loops. Doubtless you've heard of them. I won't waste your time repeating BASIC theory, but will proceed to show you a FORTH equivalent, the DO LOOP. Actually the equivalent is better than the original, as you'll see. There are several types of loop. We'll have a go at the others later.

For a start, create a do loop using a colon definition. I'll call mine COUNTDOWN. You call yours what you will. I suggest a meaningful name is better than one that merely gives vent to your frustrations! Thus:

```
: COUNTDOWN 10 10 0 DO DUP .
1- LOOP ;
```

Try it. You will find the integers from 10 to 0 inclusive printed at lightning speed on the screen.

It would have been easy to put a PAUSE inside the DO LOOP so that the counts occurred at rate discernible to human eye, instead of as a blur; but more of that later. Take a moment to let your mind run riot on the possibilities that the DO LOOP could bring.

Lets identify each term in COUNTDOWN and then discuss its purpose in the FORTH scheme of things. The colon definition has been discussed previously and accounts for the colon and semi-colon. The first 10 is merely starting data placed on the stack as fodder for the contents of the DO LOOP. The second 10 and the zero are the limit and index respectively. Everything between DO and LOOP is the set of operations done each time that the LOOP er . . loops.

When COUNTDOWN is called, 10 is put on the stack; then DO puts the limit and the index on the return stack. (The return stack is a different place from the stack where ordinary number operations occur. It has certain special uses, as you'll see). Each word inside the loop is progressively executed until LOOP is reached. LOOP compares the limit and the index. If the index is less than the limit DO is re-executed and the index is incremented by one. When LOOP finds that the index and the limit have the same value execution moves to the next word in the colon definition. I mentioned PAUSE before; we could define PAUSE as:

```
: PAUSE 1000 0 DO LOOP ;
```

i.e. just like a simple delay routine in BASIC.

The return stack contains the present value of the index. No doubt you'll agree that such a number could be useful. Here's a routine from Starting FORTH, p 130, that uses the present value of the index obtained by using the FORTH word I. (I copies the top value on the return stack and puts the value on the calculation stack).

```
: MULTIPLICATIONS CR 11 1 DO
DUP I * . LOOP DROP
```

Typing (say) 5 MULTIPLICATIONS results in the following:

```
5 MULTIPLICATION
5 10 15 20 25 30 35 40 45 50
```

The CR is merely a carriage return.

It is possible to create a loop which increments by a value other than one, that is, just like step in BASIC. The format is like this:

```
: NAME (limit) (index) DO (operation(s)) (step
value) +LOOP ;
```

```
i.e. : INTENS 100 0 DO I . 10
+LOOP ;
```

will count up in tens to 90.

Doubtless you feel like racing off and creating DO LOOPS willy-nilly. Before you do there are a few things you need to know.

- 1) A DO LOOP executes at least once.
- 2) DO LOOPS must be executed inside a colon definition.
- 3) Execution ceases when the limit is equalled or passed.
- 4) If you have a loop which leaves a "remainder" on the stack then ultimately the stack will crash if it happens too often.

There are two ways of ridding the stack of garbage: you can keep on typing . which will progressively (and very tediously) print and remove the garbage; or you can type in and attempt to execute a nonsense word, e.g. QQQ, which will promptly empty the stack. I suggest the latter course of action!

There is more: you'll find it in Starting FORTH. Lets look briefly at two other types of (indefinite) loop.

In order to understand the following types of loop we must understand flags. A flag in this case is a number placed on the stack and used as the criteria for continued looping (or not) by UNTIL or WHILE. The flag is often created by a conditional test which leaves a one or a zero on the stack. A zero value means false and a one means true. Some conditional tests are = - < > 0= 0< 0> . They can be associated with the logical operators AND OR and NOT.

Lets see what a couple of the conditional tests do. 0< tests to see if the top value on the stack is negative. If so then a one (true) is written onto the stack else a zero (false) is placed there. A - used as a conditional test would return a true if one number of a pair of non equal numbers was subtracted from the other. The following example will demonstrate the use of a test.

The BEGIN UNTIL loop.

This loop continues to loop while a false (zero) appears on the stack for UNTIL to "see". Thus, in the following line:

```
: ENDAT30 30 100 BEGIN
." GREATER THAN " 1- = UNTIL
." EQUAL TO " ;
```

the loop will loop until the 100 is decremented to

# VIRTUALLY DONE

by John Redmond

With some misgivings, I interrupt our discussion of the graphics language, XGAL, to meet a need that appears to be even greater - that of a disk operating system for A\*FORTH. I apologize to those who have been waiting for the Turtle Graphics routines but I think that, in the long run, you will be happier.

One of the criticisms sometimes expressed about Forth is that it is incompatible with other operating systems. This is true - but it is also true for just about every operating system. So why bother about yet another system? A few reasons spring to mind:

1. It costs nothing (in the sense that it is an integral part of the Forth interpreter/compiler).
2. It is very fast.
3. It can be altered to meet the needs of the user.
4. It divides the disk into multiples of 1 kbytes (screens) and is less wasteful than the granule system.
5. It wastes nothing of the disk space (a 40-track disk has a full 180 kbytes per side).
6. Because the 1 kbyte screens are standard in Forth, provided that the disk format is compatible, you can take your disk to any other computer (e.g., an IBM PC) and compile your files.
7. Finally, the Forth approach using virtual memory was established well before the current fashion in virtual memory and virtual disks. It is, quite simply, much more powerful and flexible than anything else.

So now the time has come for A\*FORTH to go virtual and this month we discuss its operating system. It's written, of course, in standard Forth and, as usual, it requires a 32k system but not Extended Color Basic. The only ROM routines it uses are the official Tandy ones, so that it is revision-independent. I use it on BDOS.

To make the best use of space, the memory map has been altered from that used in the tape version (this is easily done as described in the A\*FORTH manual). The top 6k is reserved for the high-res screen and directly below that come the four disk buffers (we could have decided on more or less than four, but this is a good compromise). The stacks come directly below the buffers and leave about 10-11k of space for programs. (This is an enormous amount of room for a Forth program).

---

30. While doing so the message "GREATER THAN" will be printed until the loop is exited, when the message "EQUAL TO" will be printed.

In this case the conditional = returns false until the two uppermost values on the stack are equal.

The BEGIN WHILE REPEAT loop.

This loop is slightly more complex than the last one in that any words between the FORTH words BEGIN and WHILE are always executed, while any FORTH words between WHILE and REPEAT are only executed while a conditional (at WHILE) is true. As soon as the conditional returns a false the execution terminates at WHILE. For example:

The idea of the virtual memory handling is to specify one of the 1k screens on disk (typically numbered 0 to 180 for a SSDD system), e.g.,

```
9 BLOCK
```

This command will return the address of the first location where the contents of screen 9 (on disk) now resides in memory. The address is that of the start of one of the disk buffers. A good trick, but how is it done? Look at the definition of BLOCK. It is a high-level word, i.e., it invokes lower-level words to do much of the hard work. BLOCK expects the screen number on the stack and the first thing that it does is check whether it is the same screen as last asked for by the interpreter/compiler. (The number of the last screen is held in the variable PREV and the address last returned is held in the variable NOW.) If it is the same, the address in NOW is simply returned. Otherwise PRESENT is asked to determine whether is in fact present in any of the buffers. If so, the corresponding address is fetched and returned.

If the screen is not in memory, it has to be loaded from disk but, before this can be done, a buffer must be allocated. This is done, appropriately, by BUFFER. Look at its definition. It looks for a vacant buffer and, if it finds one, reserves it for the screen. If all buffers are in use, BUFFER finds the oldest (i.e., the one used least recently). If the contents of this buffer have been UPDATED (e.g., altered by editing), ?PUT writes the buffer back to the correct place on the disk. Only then can the desired screen be read in from the disk and the final address be returned by BLOCK.

Exhausting, isn't it? The marvellous thing is, though, that it all happens so fast and that the gentle programmer doesn't need to know or care about all the housekeeping that is involved.

I'm reasonably pleased with the code this month - and I'm fairly difficult to please. For those who want to come to grips with Forth, it's not a bad idea to work through the code - but remember that you should work backwards, starting with the high-level words. Furthermore, this is a fairly standard implementation in terms of variable names and memory usage, so it will give you some insight into the workings of a typical Forth operating system.

For the uninitiated, some comments on the user-available words: MTB (= EMPTY-BUFFERS) kills the contents of all buffers and should always be used on startup to kill garbage. FLUSH does just the opposite. It writes the contents of all updated

---

```
: TRU-FALSE 30 100 BEGIN ." IS"  
 1- - WHILE ." TRUE " REPEAT  
  ." FALSE " ;
```

will continue to print IS TRUE (executing both parts of the loop) until the conditional - shows false, when TRUE is not printed and the loop is exited, resulting in the printing of the words IS FALSE.

That's all for now. I hope as usual that you've enjoyed the article. If you would like to talk about FORTH please feel free to ring me on 07 208 7820.

buffers and should always be used immediately at the end of an editing session. LOAD interprets/compiles instructions or a source program from a source text in virtual memory.

LOAD has the remarkable property of being recursive. It will allow one screen to load another screen, which can then load another screen or range of screens, and so on. This means that you can use a 'load screen' to load different groups of screens that make up the final applications program. It's like a considerable enhancement of the #include option in C but, again, much more flexible. The extraordinary thing about such a series of nested loads is that the interpreter/compiler always returns to where it started. It manages this by keeping a record of the current position each time it enters LOAD (look at the definition of LOAD).

To continue, THRU loads a range of screens from virtual memory, e.g., 43 46 THRU loads screens 43 to 46, inclusive. COPY copies screens from one area of the disk to another, e.g., 43 71 6 COPY copies 6 screens, starting at 43, to the area starting at 71. Watch out for overlapping ranges! Last of all, TAPE and DISK toggle the vector location, as described in the A\*FORTH manual.

To finish up: next month we will describe an editor for creation and alteration of screens in virtual memory. Again, it is pretty standard - it's an enhancement of the editor described in Brodies 'Starting Forth'. Registered A\*FORTH users can update by sending to me \$7 plus either two single-sided or one double-sided disk. (Make sure that they are new and formatted, please! The quality of the disks is your concern.). In return, you will get a fully-configured disk version of A\*FORTH plus an extensive library of Forth programs and utilities (including the editor and graphics packages). I consider this a very good deal, but it is the sort of support that legitimate software users are entitled to expect. Thieves need not bother contacting me.

```
ISCREEN: 40
00: \ \ EDITOR 1.
01: : SSEDIT ;
02: HEX
03: 10 CONSTANT L/SCREEN
04: 200 /LINE / 2 - CONSTANT L/PAGE
05: /LINE L/SCREEN * CONSTANT BUFFLENGTH
06: 88 CONSTANT CURSOR
07: 20 CONSTANT BL
08: 0E07 CONSTANT CURCHAR
09: 0BF CURCHAR C1
10: VARIABLE BUFFSTART
11: VARIABLE CPTR
12: VARIABLE ACTIVE
13: VARIABLE LIMITS 2 ALLOT
14:
15: DECIMAL
JWR 14APR86
```

```
SCREEN: 41
00: \ \ EDITOR 2
01: HEX
02: : IBUFF PAD /LINE + ;
03: : FBUFF IBUFF /LINE + 1+ ;
04: : CLEAR BL FILL ;
05: : +CURSOR CPTR +1 ;
06: : +CHECK + L/SCREEN MIN ;
07: : -CHECK - 0 MAX ;
08: : SHOWN ( U-F ) LIMITS 20 >R OVER > SWAP R ) 1- > AND ;
09: : HOME 400 CURSOR 1 ;
10: : SCREENBASE CURSOR 0 600 OVER - 60 FILL ;
11: : TOBASE 560 CURSOR 1 SCREENBASE ;
12: : NOTFOUND TOBASE ." not found" ;
13:
14:
15: DECIMAL
JWR 19APR86
```

```
SCREEN: 42
00: \ \ EDITOR 3.
01: : SHIFT SWAP COUNT ROT SWAP CMOVE ;
02: : MATCH FBUFF COUNT ROT -TEXT 0= ;
03: : INSTRING 0 WORD C0 IF DUP /LINE 1+ CLEAR
04: : HERE SWAP OVER C0 /LINE MIN 1+ CMOVE ELSE DROP THEN ;
05: : TOFIND FBUFF INSTRING ;
06: : TOADD IBUFF INSTRING ;
07:
08: CREATE (STAMP) 13 ALLOT
09: : #CHARS ( ADD-U ) 0 BEGIN OVER C0
10: : WHILE 1+ SWAP 1+ SWAP REPEAT SWAP DROP ;
11: : GETSTAMP PAGE ." DATE STAMP (12 CHARS MAX): "
12: : CR (STAMP) 12 EXPECT ;
13: : DOSTAMP (STAMP) BUFFSTART 0 50 + DUP 12 CLEAR
14: : (STAMP) #CHARS CMOVE ;
15:
JWR 15APR86
```

```
SCREEN: 43
00: \ \ EDITOR 4.
01: : 'BUFFEND BUFFSTART 0 BUFFLENGTH + ;
02: : THISLINE ACTIVE 0 ;
03: : START ( # ) /LINE * BUFFSTART 0 + ;
04: : LSTART ACTIVE 0 START ;
05: : LEND LSTART /LINE + 'BUFFEND MIN ;
06:
07: : ACTIVATE ( # ) DUP ACTIVE ! START CPTR ! ;
08: : CHANGELINE CPTR 0 BUFFSTART 0 - /LINE / ACTIVE ! ;
09:
10:
11:
12:
13:
14:
15:
JWR 14APR86
```

```
SCREEN: 44
00: \ \ EDITOR 5.
01: : FORWARD FBUFF C0 +CURSOR ;
02: : BACK FBUFF C0 NEGATE +CURSOR ;
03: : +SLIDE /LINE OVER + 'BUFFEND OVER - ?DUP
04: : IF CMOVE ELSE ZDROP THEN ;
05: : -SLIDE DUP /LINE + DUP ROT 'BUFFEND ROT - ?DUP
06: : IF CMOVE ELSE ZDROP THEN 'BUFFEND /LINE - /LINE CLEAR ;
07: : +SHUFFLE ( 'curs,ten ) DUP +CURSOR
08: : OVER + LEND OVER - CMOVE ;
09: : xSHUFFLE ( -'curs,ten ) OVER LEND ( NOT IF ZDROP EXIT THEN
10: : >R DUP 1 - OVER LEND SWAP - CMOVE LEND 1 - R ) CLEAR ;
11: : KILL CPTR 0 DUP LSTART - -SHUFFLE LSTART CPTR ! ;
12:
13:
14:
15:
JWR 19APR86
```

```
SCREEN: 45
00: \ \ EDITOR 6.
01: : ? 2DUP ( IF OVER - TYPE ELSE ZDROP THEN ;
02: : .LEFT LSTART CPTR 0 ? ;
03: : .RIGHT CPTR 0 LEND 2- ? ;
04: : (.LINE) .LEFT CURCHAR C0 EMIT .RIGHT ;
05: : .LINE ( n ) DUP THISLINE =
06: : IF DROP (.LINE) ELSE START /LINE 2- TYPE THEN CR ;
07: : .HEADER HOME ." line " THISLINE ." screen " SCR 0 . CR ;
08:
09: : RANGE ( --hi,lo ) THISLINE 2 -CHECK L/PAGE OVER +CHECK SWAP
10: : 2DUP LIMITS 2! ;
11:
12:
13:
14:
15:
JWR 14APR86
```

```
SCREEN: 46
00: \ \ EDITOR 7
01: : (P) DUP /LINE CLEAR TOADD IBUFF SWAP SHIFT ;
02: : (F) 0 SWAP CPTR 0 1+ DO I MATCH
03: : IF 1 CPTR 1 CHANGELINE 1+ LEAVE THEN LOOP ;
04: : (A) IBUFF C0 LEND 2- CPTR 0 - ) NOT
05: : IF CPTR 0 DUP IBUFF C0 +SHUFFLE IBUFF SWAP SHIFT THEN ;
06: : (E) CPTR 0 LSTART - FBUFF C0 ( NOT
07: : IF CPTR 0 FBUFF C0 -SHUFFLE BACK THEN ;
08: : (B) THISLINE 1 -CHECK ACTIVATE ;
09: : (N) THISLINE 1 +CHECK ACTIVATE ;
10:
11:
12:
13:
14:
15:
JWR 19APR86
```

```
SCREEN: 47
00: \ \ EDITOR 8.
01: : (L) .HEADER DO I .LINE LOOP SCREENBASE ;
02: : LST THISLINE SHOWN IF LIMITS 20 ELSE RANGE THEN (L) ;
03: : L LIMITS 2+ 0 ACTIVATE LST ;
04: : ULIST UPDATE DOSTAMP LST ;
05: : T ( # ) 0 MAX L/SCREEN 1- MIN ACTIVATE LST ;
06: : ^ L/PAGE 0 LIMITS 2! 0 ACTIVATE LST ;
07: : C LSTART CPTR 1 LST ;
08: : N THISLINE 1+ L/SCREEN MIN ACTIVATE LST ;
09: : B THISLINE 1- 0 MAX ACTIVATE LST ;
10: : E (E) ULIST ;
11: : P THISLINE L/SCREEN ( IF
12: : LSTART (P) ULIST ELSE ABORT" off screen" THEN ;
13: : U (N) THISLINE L/SCREEN ( IF
14: : CPTR 0 DUP +SLIDE (P) ULIST ELSE ABORT" no room" THEN ;
15:
JWR 14APR86
```

```
SCREEN: 48
00: \ \ EDITOR 9.
01: : X LSTART DUP IBUFF 1+ /LINE CMOVE
02: : /LINE IBUFF C1 -SLIDE (B) ULIST ;
03: : F TOFIND 'BUFFEND (F) IF FORWARD LST ELSE NOTFOUND THEN ;
04: : D TOFIND 'BUFFEND (F) IF FORWARD E ELSE NOTFOUND THEN ;
05: : A TOADD (A) ULIST ;
06: : R TOFIND 'BUFFEND (F) IF FORWARD (E) (A) ULIST
07: : ELSE NOTFOUND THEN ;
08: : K TOFIND LEND (F)
09: : IF KILL ULIST ELSE NOTFOUND THEN ;
10: : Z CPTR 0 LEND OVER - CLEAR ULIST ;
11: : WIPE SCR 0 BLOCK B/BUFF CLEAR ^ ;
12: : LIST DUP SCR 1 BLOCK DUP CPTR 1 BUFFSTART 1 ^ ;
13:
14: GETSTAMP \ do it now!
15:
JWR 19APR86
```



# 16 BIT!

Life In The Fast Lane!

Jerome  
Siappy

Class 68008 CoCo Add-On Update.....

The 512K memory update for the Class Computer is complete and working great, and provides 2016 Sectors of FAST RAM disk. This is a secondary use of the Class 68008 Computer. Even using this expansion board in this fashion makes using CoCo OS-9 a dream. Multi-tasking is truly transparent.

Also available is an updated version of TRansL, a 6809 to 68000 code translator with full error checking. If you are thinking of transferring 6809 programs to the Class 68008, Atari or Amiga, this will accelerate the process.

Final software and design implementation is being done to support Hard Disk for the Class 68008. Plans are under way to support 5, 10, and 20 megabyte hard drives and also the hard drive system will be evaluated to support other computers.

The good news is that OS9 68K is being ported to the Class 68008 as I write these notes. This version will support hard and floppy disks, Wordpak and normal CoCo's or the 256 and 512K CoCo's.

I haven't any design details of the 'new CoCo' but if it's software and hardware compatible with the 'old CoCo', then it's highly conceivable that the Class Computer will work with the new CoCo; great

isn't it!

Price configuration has not been finalized but be assured you will be capable of having a fully fledged add-on OS9 68K system at a competitive price and still maintain total CoCo compatibility.

Speaking of compatible, I'm investigating adaptables with the new CoCo, to run 6809 OS-9 and 68000 OS-9 simultaneously. Will keep you posted.

In the future months I will still keep you updated on the Class 68008, as I still have to review 'Kamelion', the interface operating system and next month I'll try to show you a sample translation of a 6809 OS9 to 68000 translator program.

It is my intent over the coming months to show the true advantage of multi processor concepts.

Well I must get back to programming OS-9 68K. Until next month.

Happy Computing,  
Jerome.

P.S. If any of you users out there would like to form a 68K Users' Group please let me know through Blaxland Computer Services (ph.047-39-3903). The 68K computers will set the standard for years to come in personal computers. Let's share and accelerate ...

continued from Page 10

There are still memory constraints inherent to OS-9 Level I, but a RAM disk seems to speed up things quite a bit.

Brian Lantz has licensed both the Disk BASIC and OS-9 RAM disk drivers to DISTO. The Disk BASIC version is on a separate disk.

After entering `LOADM"RAMPAK":EXEC`, you are prompted for the default RAM disk drive number and Multi-Pak slot number. Then, you are asked whether to clear the RAM disk.

After these prompts, the RAM disk is formatted and available for use as another disk drive. You are in the 64K RAM mode and the driver is located at \$FD00. This is why you are asked whether to clear the RAM disk. If you have to use the Reset button, the data on the RAM disk is not lost, but you have to re-initialize the driver before it can be recovered.

The DISTO Super RAMDisk OS-9 Driver by Brian Lantz is virtually the same driver software used for other memory expansions. The module is named RAMDisk and, after loading this module, you must link the driver "R0" to the system. You must then format the RAM disk. The default format is 40 tracks, single-sided, but the device

descriptor can be changed to take full advantage of the spare memory.

The only drawback to doing this is backing up the RAM disk. The use of a utility by Computerware named *Dircopy* makes it easier.

The DISTO unit is compatible with all CoCos. Previous units reviewed would not work with the CoCo 2. The cost of using it is the price of a Multi-Pak. The unit is well-constructed and functions as advertised.

I do see some shortcomings with the Disk BASIC software and documentation. I sometimes wonder how to take advantage of all of this extra memory. The only documentation for Disk BASIC is a typewritten page explaining how to boot the driver. No information is given about the hardware aspects of the unit. The user should have the page addresses and a description of the hardware for experimentation purposes.

The OS-9 software documentation is just adequate. It explains all of the initialization steps and procedure files, but again it lacks any information about the hardware. This is not so bad for OS-9, due to the nature of the system, but I'd like to see it.

In comparison to other units I've seen, this unit is adequate from a hardware standpoint, and doesn't require soldering, wiring or opening the computer. We will have to wait and see if any software other than the RAM disk applications develops.

# BASIC 09

by Jack Fricker

**T**his time we are going to look at errors. No, not the occasional one that I manage to put in this column (nobody's perfect). But the annoying ones generated by OS9 when something goes wrong. This is one of the things that critics of OS9 say is wrong with the operating system.

There are a couple of solutions to this problem, one of these is the PRINTERR command that comes with CoCo OS9. The problem with this is that once enabled it cannot be disabled without rebooting the system and it also uses up precious memory.

One other method is the one that I am presenting here (Listing 1). The advantages of this is that it returns the memory after it is finished. The reason that I originally wrote this program is that on my 68000 OS9 system (yes, 68000) there is no similar command, nor is there any similar command on OS9 Level 2 (not version 2).

Level 2 is the version of OS9 that will run on the mythical new CoCo if and when it is ever released.

Anyway enough of the soap box stuff. This program will work on any version or level of OS9. Because it is written in BASIC09 it works on either system with the changing of one line.

Now about the causes of some of them. The first one to look at is error # 216 (path name not 'found). This occurs because the name of the file or program is not found in the current "execution" or "data" directory; it doesn't mean that the file doesn't exist. It just means that you didn't tell it to look in the right place. The way to find out where you are in your disk is to use the "PWD" and "PXD" commands.

"PWD" will tell you your working or data directory and "PXD" will tell you your execution directory and DIR and DIR X will tell you what your files in those directories are.

The next common error to look at is 214 (no permission). What this means is that the attributes of the file have been set so that only the creator has access to that file. To change the attributes of that file use the "ATTR" command. Another cause of this error is if you try to list a directory or if you treat a file as a directory. This problem occurs mostly commonly if you don't follow the OS9 convention of keeping all files in lower case and

all directories in upper case.

Then there is the dreaded error #207 (out of memory). Unfortunately there is very little that can be done about it.

One of the things you can do is to build a boot disk with only the files that you need. We have covered this in earlier articles which brings me to another kind of error - the ones that I make!

In an earlier article about building boot disks I stated you should have OS9, OS9P2, "boot" and "init" in the temp file and then use them with "os9gen" to generate new boot disks. "os9gen" will put these files on the disk itself. Although it will work it wastes memory terribly!

```

PROCEDURE basic09 errnum
0000   ON ERROR GOTO 50
000A 10   DIM filename:STRING(32)
0024     REM make filename no longer than 32 chars
long
0056     DIM inpath:BYTE
0062     REM make inpath a 1 byte integer
0084     DIM char$:STRING(1000)
009A     REM make char long enough to handle 1000
bytes
00CA     PRINT CHR$(12)
00D6     REM clear screen
00E8     filename:="/d0/sys/errmsg"
0102     INPUT "number of error that occurred
",number
012E 30   OPEN #inpath,filename:READ
0146 40   WHILE NOT (EOF(#inpath)) DO
015C     INPUT #inpath,char$
016C     num=val(left$(char$,3))
1082     IF num=number THEN
1096       PRINT char$
109E       GOTO 60
01A4     ENDIF
01A8 50   ENDWHILE
01B2 60   CLOSE #inpath
01C0 70   END

```

# Featuring a New Text Formatter

By Dale L. Puckett

**O**S-9 Users Group member Frank Malaney of Pataskala, Ohio, takes the spotlight this month. Malaney contributed the source code for *PrintForm*, a public domain program he has been distributing as "shareware" for several months. He also passed along some useful C programming tips. Rounding out our May offering is an alternative, *SysGo*, from Robert A. Larson at USC; another tip from Steve Goldberg in Bethpage, New York; some short C programs for beginners from Dennis J. Duke in Bessemer, Alabama, and Eric Richards in Auburn, Alabama, plus a

look at a few new OS-9 products. Remember, if you have a question, a short to medium-sized program or an operating tip, we would love to share it with our readers. Send your thoughts to us at THE RAINBOW or EMAIL them to DALEP on RAINBOW's Delphi CoCo SIG or to my PPN, 70010,542 on CompuServe.

#### *PrintForm* is Modular

*PrintForm*, our feature offering from Frank Malaney, performs most of the functions of *DynaForm* and corrects many

of the printer problems that were present in early versions of this word processing software. The problems revolved around the printer setup standard used by Tandy. Most manufacturers set up their printers to only return the printhead to the left-hand margin after they receive a carriage return character, 0D Hex. Radio Shack printers, however, automatically add a line feed following every carriage return. This drives some software and most programmers crazy.

Ever since the first column, we have been preaching the virtues of OS-9's modularity, and Frank Malaney is a believer who broke

BLAXLAND  
COMPUTER  
SERVICES  
PTY. LTD.

(047) 39-3903

COCO & 1000  
SPECIALISTS

PRE COCOCONF SPECIALS

ON ALL COCO HARDWARE AND SOFTWARE

Ring for YOUR price.

Class Computer (68000) literature available.

Don't forget to get your entries in for the OS-9/68000 programming contest.

See us on Goldlink this month!

76A MURPHY ST. BLAXLAND 2774

the program into 18 different modules, small pieces "... to protect the sanity of the programmer," Malaney said. Breaking long programs into short segments also helps the computer, particularly a Color Computer with only 64K of memory. As you know, if you have ever tried to compile a long C or PASCAL program, most compilers generate a large number of error statements for each

actual error in the source code.

"The best way to handle this situation is to correct the first error, recompile the program and then fix the next error that shows up, etc.," Malaney said. He also noted that small modules that perform a single function are much easier to debug after you get the program compiled but it still does not work properly. "It is much easier to deter-

mine which code is not working correctly and to rethink the logic when that module only performs a single task," he said.

The two-line C program, *test.c*, can also make your initial compiles go faster.

```
#include "header.c"
#include "usage.c"
```

Use this OS-9 command line:

```
OS9: ccl test.c -oa >>/p
```

*Header.c* is the name of a file that defines all of the global variables in *PrintForm*. *Usage.c* is the name of the file Malaney is checking for syntax errors. When you compile *test.c* with the previous command line, you are greeted with a very fast pass through the compiler and a list of all the errors on your printer.

When compiling *PrintForm* use the following OS-9 command line:

```
OS9: ccl pf.c -m=4k
```

This line increases the data space allotted to the program by 4K during the compile. This prevents running out of memory while printing nested files.

If you do not want to type in the *PrintForm* source code listed here, Malaney will send it on a disk for \$15. He includes a copy of the manual on the disk, which can be printed out. Send check or money order to Frank Malaney, 8708 Mink Street SW, Pataskala, OH 43062. Enjoy!

*Vi*

This utility removes the *O.Pak* Hi-Res screen utility, returns to the standard OS-9 screen and executes *TSEDIT* with its file ID. After you are finished editing, it returns to *O.Pak's* Hi-Res screen. It uses the C "system()" function to do this. *O.Pak*, *Nores* and *TSEDIT* must be stored in your current execution directory before you run *Vi*.

```
#include <stdio.h>
#define CMD1 "NoRes"
#define CMD2 "TSEDIT"
#define CMD3 "o.pak"
```

```
main(argc, argv)
```

```
int argc;
char *argv[];
{
    char *Cmd_line, CMD1);
    system(Cmd_line);
    strcpy(cmd_line, CMD2);
    strcat(Cmd_line, argv[1]);
    System(Cmd_line);
    strcpy(Cmd_line, CMD3);
    System(Cmd_line);
}
```

#### Listing A:

```
#include <stdio.h>
#include <os9.h>
#define void int
#define clear 12 /* clear screen character */
#define home 1 /* home cursor character */

main()
{
    /* Routine checks both joysticks. Press fire button to end each test */
    /* Test uses OS-9 I$GETSTT system call */
    int x,y,fire,choice;

    putchar(clear);

    for (choice = 0; choice < 2; choice++)
        do
        {
            putchar (home);
            joystick(choice, &x, &y, &fire);
            printf( "%2d : x=%3d y=3d/n", choice, x, y);
        }
        while (fire==0);
}

void joystick(num,xval,yval,button)
int num,*xval,*yval,*button;
{
    struct registers reg;

    reg.rg_x=num; /* x= joystick # (0 or 1) */
    reg.rg_a=1; /* a = path #1 or standard output */
    reg.rg_b=SS_JOY; /* b= function code $13 */

    if (_os9(I_GETSTT, &reg)) /* system call */
    {
        printf(" ** ERROR in joystick read/n");
        exit(1);
    }

    *xval=reg.rg_x; /* x= horizontal value */
    *yval=reg.rg_y; /* vertical value */
    *button=reg.rg_a; /* a = fire button ($FF= on)($00= off) */
}
```

#### Listing B:

```
* PBUF -- copyright (c) S. B. GOLDBERG
*
* Initializes printer buffer to prevent memory
* fragmentation.
*
    ifpl
    use /d0/defs/os9defs
    endc
*
    mod len,name,prgrm+objct,reent+1,entry,dsiz
*
    rmb 200 for stack
```

```

dsiz equ .
*
name fcs /pbuf/
fcc /(c) 1985 S. B. Goldberg/
*
entry leax pntr,pcr name of printer
lda #write. write mode
os9 I$open open path
bcs out exit with error
os9 I$Close close printer path
bcs out exit with error
clrb clear error flag
out os9 f$exit quit
pntr fcc "/p" name
emod
len equ *
end

```

Listing 1: pf.c

```

#include "header.c"
#include "main.fast.c"
#include "linefeed.c"
#include "c_return.c"
#include "p_rint.c"
#include "space.c"
#include "putcont.c"
#include "left_m.c"
#include "contr.c"
#include "pr.header.c"
#include "end_page.c"
#include "sing_line.c"
#include "dot.c"
#include "cont_proc.c"
#include "usage.c"
#include "cput.c"

```

Listing 2: header.c

```

/* This is the header file file which contains all of the */
/* define's and global variables for a new text processing */
/* and formatting program that will do the most common */
/* functions of "dynaform". */
#include <stdio.h>
#include <ctype.h>
#define FALSE 0
#define TRUE 1

int spacing = 1; /* set by .SS or .MS */
int offset = 8; /* set by .PO */
int pg_no = 1; /* set by .BP or .PN */
int pg_len = 66; /* set by .PL */
int bot_mar = 8; /* set by .MB */
int foot_mar = 2; /* set by .FM */
int top_mar = 3; /* set by .MT */
int header_mar = 2; /* set by .HM */
int linefeed = FALSE;
int code[27][8];
int line_no = 1;
int first_char = TRUE; /* denotes first character on a line */
int underline = FALSE; /* controls "controlled underlining" */
int q_flag = FALSE; /* true after control Q */
int s_flag = FALSE; /* true after control S */
int w_flag = FALSE; /* true after control W */
int y_flag = FALSE; /* true after control Y */
int sheet_flag = FALSE; /* if false tractor paper, if true single
sheet */
int pr_flag = TRUE; /* flag for printing characters */

int spage = 0; /* number of page to start printing */
int epage = 30000; /* page number to stop printing */

```

### Joysticks in c

Another person experimenting with new frontiers is Eric Richards of Auburn, Alabama. He was so impressed with the new mouse-driven packages at Radio Shack stores nationwide that he wanted to try his hand at programming the joystick ports (Listing A). The value of the 'Y' coordinate returned by Eric's program is the opposite of that returned by the corresponding routine in Radio Shack Color BASIC. The 'X' value returns the same value as the equivalent BASIC routine

### Fixes for Kansas City BASIC

Steve Odneal, OS-9 Users Group treasurer and author of *Kansas City BASIC*, has submitted two fixes to that program. You can use EDIT to change the source code file supplied with the program and reassemble it. Or, you can send your original *Kansas City BASIC* disk with \$5 for postage and handling to Steve at 8609 East 73 Terrace, Kansas City, MO 64133 and he will do it for you.

Before you change the actual code, edit the line at the label REVS. This sets the revision level of the program module. At the label XP290, delete the following three lines:

```

XP290 CMPA #' -
      BNE XP291
      LEAY 1,Y

```

Replace them with:

```

XP290 CMPA #$FF Sub_function ?
      BNE XP291 ..No
      LDA 1,Y Get Sub-Function Code
      CMPA #$92 Minus Function?
      BNE XP291 ..No
      LEAY 2,Y Skip Codes

```

This change fixes a subtraction problem. Now, following the label TSTVE04, find this line:

```
CMPA #'@ Range Check
```

Replace it with:

```
CMPA #$2F Range Check
```

Two instructions later, just before the statement:

```
TST HCLDA+1,U
```

Insert:

```

TFR A,B
SUBB #'0 Subtract a zero
CMPB #9 Is is a number?
BLS TSTVE06 .. Yes
CMPB #16 Is is Alpha?
BLS TSTV15 ..No
TSTVE06 EQU *

```

The last set of changes allows variable names with numerics to be used following the initial required alphabetic character.

Odneal reports that he is getting excellent response to *Kansas City BASIC* and noted that several users have asked for string and

```

char head[133];
char foot[133];
char temp[133];

int contrl;

FILE *path, *fopen();

-----
*/

Listing 3: main.fast.c

main(argc, argv)
int argc;
char *argv[];

{
FILE *input_file;
int i, j, cnt, temp, count = 1, out_flag = FALSE;
char option;

static char hd[] = " ";
static char ft[] = " ";
if((input_file = fopen("prtr.contrl","rx")) == NULL)
{
printf("I couldn't open printer configuration file");
exit(1);
}

fread(&code[0][0], sizeof(int), 216, input_file);
fclose(input_file);

if(code[0][0] == 1) /* check if linefeed needed */
linefeed = TRUE;
contrl = code[0][1]; /* load character used as control flag */

/* Open a path for output and get number of copies */
if(argc > 2)
{
for(i=2; i < argc; ++i)
{
if((argv[i][0] == '-')
{
j = 1;
while((option = argv[i][j]) != NULL)
{
if(isalpha(option) == FALSE)
{
printf("Error in options\n");
usage();
exit(6);
}
cnt = 0;
++j;
while(isdigit(argv[i][j]) != FALSE)
{
cnt = cnt * 10 + (argv[i][j] - 48);
++j;
}
option = toupper(option);
switch (option)
{
case 'C':
count = cnt;
if(count < 1)
{
printf("Number of copies set to
zero\n");
exit(1);
}
break;
case 'S':
spage = cnt;
if(spage > 1)
pr_flag = FALSE;
break;
case 'E':
epage = cnt + 1;
break;
default :
printf("Unknown option\n");
usage();
exit(7);
break;
}
}
}
else
{
if((path = fopen(argv[i], "w")) == NULL)
{
printf("I can't open a path for
%s\n", argv[i]);
usage();
exit(2);
}
out_flag = TRUE;
}
}
}
if(out_flag == FALSE)
{

```

numeric arrays and graphics ability. He notes that graphics would be the easiest and asks that you let him know if you have strong interest in having graphics support in *Kansas City BASIC*.

A lot of people stop after one major project like *Kansas City BASIC*. Not Steve! He is working on a *Kansas City COBOL* compiler for OS-9 and researching a FORTH and C. All will be packaged with the source code provided.

"So much software today is overpriced, unchangeable and poorly documented," Odneal said. "I feel that if users have good BASIC software with proper documentation, most of them can modify it to meet their own needs. The entire OS-9 community will benefit. If you would like to join this effort, let me know."

#### Microware Shipping OS-9 FORTRAN

Phyllis Casel, the communications coordinator at Microware, reports shipping the 6809 FORTRAN Compiler in February. The new compiler is a subset of the FORTRAN 77 ANSI standard with a number of powerful extensions. Highlights include the ability to generate code for two- or four-byte integers, single and double precision floating point support, a full math library and an updated C compatible linker and assembler.

If you are looking forward to moving up to an OS-9 68K system, take note. The OS-9 Network file system, which features a user interface similar to the normal OS-9 file system, is also shipping as is a brand new version, 2.00, of the 68K C compiler.

The popularity of OS-9 is growing so fast that Microware is expanding to meet demands.

#### More Tricks

The new Iniz command in the 2.00.00 version of OS-9 is excellent. It lets you eliminate the memory fragmentation caused by opening a path to a printer or other device during operation. You simply put the command "Iniz P" in your *startup* file and go.

When I first tried to run Iniz, without reading the directions of course, I typed "Iniz /p" on the command line and wound up with a nasty error message on my Color Computer screen. I scratched my head and looked at the book only to learn that the programmer who wrote Iniz had dropped the slash, '/' — the same slash that always tells OS-9 to look for a device rather than a file — from the command line syntax. I wonder why? I thought the idea behind OS-9's unified I/O was to make everything consistent.

Now, the good news. If you don't have Version 2.00 and don't plan on getting it for a while, you can emulate the Iniz feature with a short program (Listing B) from Steve Goldberg in Bethpage, New York. When you run the program — usually from your *startup* file — it merely opens up a path to your printer and then closes it before you have had an opportunity to load any other programs in memory. This means the printer buffer is set up at the very top of RAM, leaving you with a continuous block of free memory.

```

        if((path = fopen("/p","w")) == NULL)
        {
            printf("I can't open a path to the printer\n");
            exit(3);
        }
    }
    /* Open the path for the input file */
    for(i=0; i < count; ++i)
    {
        strcpy(head,hd);
        strcpy(foot,ft);

        if(argc >= 2) /* check for path name */
        {
            if((input_file = fopen(argv[1],"r")) == NULL)
            {
                printf("I can't open %s for reading\n",argv[1]);
                usage();
                exit(4);
            }
        }
        else
        {
            printf("You must put a filename in the command
line\n");
            usage();
            exit(5);
        }

        /* We are now ready to begin the actual printing of the document */
        print(input_file);
        fclose(input_file);

        /* When we return to this point we must now finish the last page */
        end_page();
        /* Reset all variables for next pass if required */
        if(spacing > 1)
            _flag = FALSE;
        else
            pr_flag = TRUE;
        spacing = 1;
        offset = 8;
        pg_no = 1;
        pg_len = 66;
        bot_mar = 8;
        foot_mar = 2;
        top_mar = 3;
        header_mar = 2;
    }
    fclose(path); /* close our output path (to printer ?) */
}
-----
*/

```

#### Listing 4: linefeed.c

```

/* this function puts out either a cr-lf or a blank-cr pair depending
of the */
/* state of the linefeed flag. The blank is required by some printers
as they*/
/* will not respond to only a cr.
*/

int Linefeed()
{
    char lf = '\012'; /* linefeed code */
    char cr = '\015'; /* carriage return code */

    if(linefeed == TRUE)
    {
        cput(cr,path);
        cput(lf,path);
    }
    else
    {
        space();
        cput(cr,path);
    }
    ++line_no;
}
-----
*/

```

#### Listing 5: c\_return.c

```

/* this function processes each linefeed found in the text and
determines */
/* how many line spaces between lines are required
*/

int c_return()
{
    int i;
    for(i=1; i<= spacing; ++i)
        Linefeed();
}

```

#### Eliminating Hard Coding

It bugs me to see a programmer ruin an otherwise excellent piece of software by hard coding system device information into the program itself. Let's study an example.

The new 256K RamDisks available now for the Color Computer make OS-9 operation a dream — if the programmer hasn't hard-coded his program. When I first boot OS-9, I *format* the RamDisk and *backup* the disk that contains my current execution directory. Then, I change both the execution and data directories to the RamDisk.

But, what do you think happens when you hit the wrong key while typing a command line? You guessed it, OS-9 reports an error and if you have installed *PrintErr*, you hear drive /d0 start up while OS-9 looks for the proper message to print. I saw this happen a couple of times with dismay.

Here's the fix. At an offset of 0016 — in the 2.00 version of *PrintErr* — you will find the string /D0/SYS/ERRMSG. Use *Debug* to change the /D0 to "...". This tells OS-9 to look in the SYS directory on the parent of the parent of the current data directory. If your current data directory is /R0, the "..." will cause *PrintErr* to look on /R0. If it is /H0, it will cause it to look on /H0. After you have made the change and exited *Debug*, save the module to a disk file, *newPrintErr* perhaps. Rename the original to *PrintErr.Original* and then type:

```

OS9: verify </d0/cmds/newPrintE
rr >/d0/cmds/PrintErr U

```

*Desk Mate*, the mouse-driven masterpiece from Tandy, has the same problem — it hard codes four separate device names. Fortunately, the four pathlists are coded in only one file, *desk*. Here is a table with the old values and the new values.

Table 1: Offset Values

00E72F 44 30 (/D0) (old)	2F 52 30 (/R0) (new)	or 2E 2E 2E (...)
01102F 44 31 (/D1) (old)	2F 52 30 (/R0) (new)	or 2E 2E 2E (...)
01392F 44 30 2F 43 4D 44 53 (/D0/CMDS) (old)	2F 52 30 2F 43 4D 44 53 (/R0/CMDS) (new)	or 2E 2E 2E 2F 43 4D 44 53 (.../CMDS) (new)
01622F 44 31 2F 43 4D 44 53 (/D1/CMDS) (old)	2F 52 30 2F 43 4D 44 53 (/R0/CMDS) (new)	or 2E 2E 2E 2F 43 4D 44 53 (.../CMDS) (new)

After you make the changes above, update the *desk* module's CRC using *verify*'s u parameter. Then, run *Desk Mate* from your RamDisk and stare back in amazement. Click the button on the mouse a couple of times and *Desk Mate* will snap to

```

    first_char = TRUE;
)
-----
/
Listing 6: print.c
int print(fi)
FILE *fi; /* we passed a file pointer */
{
    int c, test;

    while((c=getc(fi)) != EOF)
    {
        test = TRUE;
        if(c=='.' && first_char == TRUE)
        {
            dot(fi); /* process dot commands */
            test = FALSE;
        }
        else if(c == ctrl)
        {
            contr(fi); /* process control characters */
            test = FALSE;
        }
        if(test == TRUE)
        {
            if(line_no == 1 && first_char == TRUE)
            {
                header(fi); /* printer header */
            }
            if(first_char == TRUE)
            {
                left_m(); /* print left margin */
                first_char = FALSE;
            }
            if(c != ' ' && c != '\015') /* test for blank and cr */
            {
                cput(c, path);
            }
            else
            {
                if(c == ' ')
                    space();
                if(c == '\015')
                {
                    c_return();
                    if((line_no + spacing) > (pg_len - bot_mar))
                        end_page();
                }
            }
        }
    } /* closes while */
} /* close print() */
-----
/

```

#### Listing 7: space.c

```

/* space() will toggle underline off if it is on and print */
/* a space and then toggle the underline back on if the */
/* underline flag is TRUE */
int space()
{
    int off=22; /* code to turn underline off */
    int on = 21; /* code to turn underline on */
    char c = ' ';

    if(underline == TRUE)
    {
        putcont(off);
        cput(c, path);
        putcont(on);
    }
    else
        cput(c, path);
}
/* -----
*/

```

#### Listing 8: putcont.c

```

/* putcont() is passed as int and will use that int to point to a */
/* row in code[row][col] matrix. It will put the int's in the row */
/* to the output until the element is >127. */
int putcont(row)
int row;
{
    int col, pcode;

    for(col=0 ; code[row][col] <= 127; ++col)
    {
        putc(code[row][col], path);
    }
}

```

your next application almost instantaneously. It's a lot like running similar software on a Macintosh with a hard disk but a whole lot cheaper. Despite the fact that applications are all relatively simple, *Desk Mate* has to be the best thing to hit the Color Computer, since OS-9. In fact, I have a friend here in Washington who uses *Desk Mate*'s text editor to enter almost everything he writes because of the large, easy-to-read characters it displays. If he has a complicated formatting job and needs a more powerful text processor, he simply runs the output file generated by *Desk Mate*'s text editor through a more comprehensive text processor such as *DynaStar*.

#### Yet Another Tip

How do you prompt yourself to change disks from a procedure file while doing a single disk copy? One quick way suggested by Brian Lantz, president of the OS-9 Users Group, is to use the OS-9 SLEEP utility command. You always knew there was a use for that command, didn't you? Try this in your procedure file:

```

-X
Echo
T
* Change Disk then
* Type 'Break' key to continue
-T
Sleep 0
X
(* Rest of procedure file follows the "x" *)

```

Notice how we used the four built-in Shell commands x, -x, t and -t. The -x command in the first line of the procedure above tells OS-9 to ignore any errors on the command line and go ahead with the rest of the procedure file. Without that command, OS-9 would abort the procedure file if it hit an error.

The t command tells the Shell to pass anything on the standard input path through to the standard output path. The -t tells it not to pass this information. Notice how we used the t command in conjunction with the echo command to send more than one line of text in our prompt. Do not forget the asterisk, '\*'. It tells the OS-9 Shell that everything else on the line that follows is a comment and should be ignored.

#### Auto RamDisk

We recently experimented for a long time trying to come up with a way to switch our current execution directory to /R0/CMDS and our current data directory to /R0 automatically from the *startup* file. Our first attempt looked something like this:

```

PRINTERR
XMODE /P1 LF
TMODE .1 -UPC -PAUSE
SETIME </TERM
INIZ P1 T2
FORMAT /R0
YDALE'S RAMDISK
BACKUP #100 /DO /R0
YY
TMODE .1 PAUSE
CHX /R0/CMDS
CHD /R0

```



```

)
/* -----*/
Listing 9: left_m.c
/* left_m() will provide for indentation from the left side of the
page */
/* before any line is printed. It will also toggle the underline off
and */
/* on if the underline flag is TRUE. The column that the text is to be
*/
/* printed in is changed with the .PO command.
*/

int left_m()
(
int off=22; /* code to turn underline off */
int on = 21; /* code to turn underline on */
int col;
char c = ' ';

if(underline == TRUE)
    putcont(off); /* turn off underline if flag is TRUE */

for(col = 1; col < offset; ++col)
    cput(c,path);

if(underline == TRUE)
    putcont(on); /* turn underline if flag is TRUE */
)
-----*/
/

```

#### Listing 10: contr.c

```

/* contr(fi) has the file pointer passed to it. it will get the next
*/
/* character, and pass it to cont_proc() for the actual processing.
*/
/* it checks to see if the next character is also a control char. If
*/
/* it is, it will call itself. if not, it will return the character
*/
/* to the file.
*/

contr(fi)
FILE *fi;
(
int col,c,i;

c = getc(fi);

if(c == 'P' || c == 'p')
(
i=0;
while((c=getc(fi)) != contrl)
(
temp[i]=c;
++i;
)
temp[i] = NULL;
printf("%s\n",temp);
c = readln(0,temp,132);
temp[c-1] = NULL;
for(i=0; temp[i] != NULL; ++i)
    cput(temp[i],path);
printf("\n");
c = getc(fi); /* throw away letter following control char
*/
)
else
    cont_proc(c);

if((c = getc(fi)) == contrl)
    contr(fi); /* call again */
else
    ungetc(c,fi); /* return character to file */
)
-----*/
/

```

#### Listing 11: pr.header.c

```

/* This function controls the spacing at the top of the document and
*/
/* the printing of the header.
*/

int header()
(

```

Unfortunately, it didn't work. As soon as this startup procedure file was executed, it sent an End of File signal to OS-9. This killed the Shell running it, and all the changes we made died with the Shell.

Next, we replaced the CHX and CHD command lines with:

```
EX LOGIN </TERM
```

We also edited the first line of the file, SYS/PASSWORD. When we were finished it read:

```
,,0,128,/r0/cmds,/r0,shell
```

This worked and we wound up in the proper execution and data directories. But when we did a procs command, we found that we had an extra Shell alive. It was the Shell that executed the startup procedure file and it was wasting 6K of memory. As an experiment, I tried to kill the extra Shell by typing:

```
OS9: kill 4
```

That didn't work because OS-9 will not let you kill the parent of a child process that is running. Finally, I left the login command line out of the startup file and typed it myself — interactively after the startup command was finished and OS-9 prompted me. This worked and I was left with only one Shell! Unfortunately the switch was still only semi-automatic.

Incidentally, we used the tmode -pause command in our procedure file so the backup command wouldn't stop and wait for us to press a key after it filled the screen with reports. Another alternative, if you have installed the new nil device that comes with Version 2.00, is to redirect the output of the backup command to that device. The following command line will do the job for you.

```
BACKUP #100 /DO /RO >/NIL.
```

#### SysGo: The Real Answer

Robert Larson at the University of Southern California at Los Angeles dropped us a note several months ago to promote the virtues of *Kermit* over *Xmodem*. We quote:

"*Kermit* has several advantages over *Xmodem*. It makes fewer assumptions about the system it is running on and the communications path it can use, so it will work on a wider variety of systems," Larson said. "*Xmodem* is probably still better for what it was designed for — CP/M to CP/M file transfer over eight-bit data links that can handle bursts of 132 characters. There are hundreds of *Kermit* implementations and dozens of *Xmodem* implementations. The central *Kermit* authority of Columbia University also helps make sure that all versions of *Kermit* work with each other and that improvements in the protocol are made in a compatible way."

Larson reported that there are at least three separate conversions of the "old" UNIX *Kermit* to OS-9. He said the latest version he has worked on is based on the Glen Seaton version with connect code from Bradley Bosch and some fixes by James Jones. It is available via the normal *Kermit*

```

int hd_line;

hd_line = top_mar - header_mar;

while(line_no < top_mar)
{
    if(line_no == hd_line)
        sing_line(head);
    Linefeed();
}
}
/* -----
*/

```

#### Listing 12: end\_page.c

```

/* This function will print blank lines at the bottom to the footer
line */
/* It will call for the footer line to be printed and print enough
*/
/* additional blank lines to get to the top of the next page.
*/

int end_page()
{

int foot_line;
char temp;

foot_line = pg_len - bot_mar + foot_mar;
while(line_no <= pg_len)
{
    if(line_no == foot_line)
    {
        sing_line(foot);
    }
    Linefeed();
}
++pg_no;
line_no = 1;

if(spage == pg_no)
    pr_flag = TRUE;
if(epage == pg_no)
    pr_flag = FALSE;

if(sheet_flag == TRUE && pr_flag == TRUE) /* single sheet flag */
{
    printf("Put in next sheet of paper.\nHit a key\n\n");
    temp = getchar();
}
}
/* -----
*/

```

#### Listing 13: sing\_line.c

```

/* sing_line() prints out both the header and the footer lines. It is
*/
/* passed a pointer to the proper line. It also contains procedures to
*/
/* print the page number and handle control characters in these lines.
*/

int sing_line(buffer)
char buffer[];
{

int i;
int temp_flag = FALSE;
char spc = '\007';
left_m();

if(underline == TRUE)
{
    temp_flag = TRUE;
    underline = FALSE; /* turn underline flag off */
    putcont(22); /* turn underlining off */
}

for(i = 0; buffer[i] != NULL; ++i)
{
    if(buffer[i] == '#' && pr_flag == TRUE)
        fprintf(path, "%d", pg_no);
    else if(buffer[i] == spc)
    {
        ++i;
        cont_proc(buffer[i]);
    }
    else if(buffer[i] == ' ')
        space();
    else
        cput(buffer[i], path);
}
}

```

distribution channels at Columbia University. He reported that he has also posted 35 copies, including four to Europe and one to Australia, via UUCP USENET mail. The Glen Seaton version is available in the OS-9 Users Group Library and on CompuServe.

We feature here a replacement *SysGo* module Larson contributed. It is smaller and faster than the original, but more importantly, it holds the clues to making the automatic change to alternate execution and data directories.

#### Alternate SysGo Listing

```

ifpl
use /d0/defs/os9defs
endc

c.cr equ $d
mod com,nams,$C1,$81.start,$00C8
namefcs /SysGo/
fcb 6

Cnds Fcc /Cnds/
Fcb c.cr

Shell Fcc /Shell/
Fcb c.cr

Startup Fcc /Startup -p/
fcb c.cr

Initdat Fcb $55,$00,$74,$12,$7F,$FF,$03,$B7
Fcb $FF,$DF,$7E,$F0,$0C
idatlen equ *-initdat

startleax <rti,PCR
OS9 f$icpt
leax <initdat,PCR
ldu #0071
ldb #idatlen
movidat lda, X+
sta,U+
decb
bne movidat
leax <Cnds,PCR
lda #4 execution directory
os9 i$chgdire
leax <Shell,PCR
leau <startup,PCR
ldd #0100
ldy #21
os9 f$fork
bcs infloop
os9 f$wait
restart leax <Shell, PCR
ldd #0100
ldy #0000
os9 f$fork
bcs infloop
os9 f$wait
bcc restart
infloop bra infloop

rtirtl

enod
acm equ *

```

*SysGo* is an OS-9 program that just happens to be the first process to run when booting the system. Essentially, it does three things: executes the procedure file, *startup*; starts your first process — read program and usually a Shell; then, it simply waits for all other processes to die.

If you look at the previous code, you will notice that Larson's version of *SysGo* goes into a wait state just before the label, "restart." If for some reason the original Shell that it has just started were to die, *SysGo* will automatically restart another Shell. This keeps you from crashing the system if you accidentally kill all the processes running.

When OS-9 runs the *SysGo* program it automatically sets the execution directory to

```

if(temp_flag == TRUE)
{
    underline = TRUE; /* turn underline flag back on */
    putcont(21);      /* turn underlining on */
}
}
/*
-----
**/

Listing 14: dot.c

/* dot() processes the dot commands
*/

int dot(fi)
FILE *fi;
{
    int i, temp_len, num, flag = TRUE, dot_c, c;
    char pause;
    char spc = '\007';
    FILE *new_file;

    /* form a code number from a two character string */
    c = getc(fi); /* get first character */
    dot_c = (toupper(c) - 64) * 30;
    c = getc(fi); /* get second character */
    dot_c = dot_c + (toupper(c) - 64);

    if((c = getc(fi)) == '\015')
    {
        temp[0] = NULL;
        num = 0;
        temp_len = 0;
        flag = FALSE;
    }
    else
    {
        i = 0;
        while((c = getc(fi)) != '\015')
        {
            if(c == contrl)
                temp[i] = spc;
            else
                temp[i] = c;
            ++i;
        }
        temp[i] = NULL;
        flag = TRUE;
    }
    if((num = strlen(temp)) > 0)
        num = atoi(temp);

    /* The switch cases now begin */

    switch(dot_c)
    {
        case 76: /* .BP page break */
            end_page();
            if(num > 0)
            {
                pg_no = num;
                if(spage > pg_no || pg_no >= epage)
                    pr_flag = FALSE;
                else
                    pr_flag = TRUE;
            }
            break;
        case 106: /* .CP conditional page break */
            if((line_no + spacing * num) > (pg_len - bot_mar))
                end_page();
            break;
        case 409: /* .MS multiple line spacing */
            if(num == 0)
                spacing = 2;
            else
                spacing = num;
            break;
        case 589: /* .SS single line spacing */
            spacing = 1;
            break;
        case 193: /* .FM set footer margin */
            foot_mar = num;
            break;
        case 253: /* .HM set header margin */
            header_mar = num;
            break;
        case 392: /* .MB set bottom margin */
            bot_mar = num;
            break;
        case 410: /* .MT set top margin */

```

/D0/CMDS. It knows that /D0 is the startup device because it looked in the Init module, which is simply a look-up table that holds the initial information needed to start the system. Information stored in Init includes the upper limit of RAM memory, the number of entries in the IRQ polling table, the number of entries allowed in the system device table, the name of the first program to run (most often *SysGo*, the name of the device that holds the default directory — usually, /D0, the device that becomes the standard input and output paths) and, finally, the name of the bootstrap file, *os9boot*, in the case of Color Computer OS-9.

The secret to changing data and execution directories to /R0, or even /H0, is to add some code to change those directories. However, since a RamDisk doesn't exist until the start-up procedure file runs format and backup to create it, you cannot add this code until after *SysGo* runs the start-up procedure. You will need to add two new labels just in front of the CMDS label in the *SysGo* listing. Something like this:

```

newdir fcc "/R0"
    fcb c.cr
newexe fcc "/R0"
Cmnds fcc /Cmnds/ resume old code here

```

Then, after the OS-9 *f\$wait* call, just in front of the existing "restart" label, add the following code:

```

leax <newdir,PCR point to new data directory
lda #3 files may be updated
os9 i$chgdtr
leax <newexe,PCR point to new execution directory
lda #4 files may be executed
os9 i$chgdtr do it
Restart leax <Shell, PCR and resume old code

```

The code creates a Shell and runs the programs that have been placed in the start-up procedure file. When the *startup* file ends, OS-9 receives an EOF signal and the Shell that ran the procedures dies. When this happens, execution continues with the new code that changes the current data directory to /R0 instead of /D0 and the current execution directory to /R0/CMDS instead of /D0/CMDS. After *SysGo* runs your code, it falls into the code at the label restart where it starts another Shell.

This *SysGo* module was written for Version 1.00 and 1.01. It should also work with Version 2.00. However, it does not start the clock module like the *SysGo* that comes with Version 2.00. If you use this version, you need to start the clock with the *setime* command in the *startup* file.

To install this *SysGo* module you go through several steps. First, assemble the code using the *asm* command that comes with OS-9. Then, replace the original *SysGo* with your version in a new OS9Boot file using *OS9Gen*. Hopefully, Larson's code and our short notes have removed some of the mystery surrounding *SysGo* and you feel free to experiment and customize your system to your heart's content. Let me know how it works out and if you really come up with a unique version be sure to share it with us.

**Users Group Sports First Online Recruit**  
 Congratulations to John M. Graf of

```

top_mar = num;
break;
case 492: /* .PL set page length */
pg_len = num;
break;
case 494: /* .PN set page number */
pg_no = num;
if(spage > pg_no || pg_no >= epage)
pr_flag = FALSE;
else
pr_flag = TRUE;
break;
case 495: /* .PO set page offset */
if(num == 0)
offset =1;
else
offset = num;
break;
case 586: /* .SP space lines on page */
if(line no == 1)
header(fi); /* print header before spacing */
if(num == 0)
Linefeed();
else
{
for(i = 1; i <= num; ++i)
Linefeed();
}
break;
case 593: /* single sheet flag set */
sheet_flag = TRUE;
break;
case 496: /* print text and wait for character */
printf("%s\n",temp);
printf("Push any key to continue\n\n");
getchar(pause);
break;
case 195: /* .FO text for footer line buffer */
strcpy(foot,temp);
break;
case 245: /* .HE text for header line buffer */
strcpy(head,temp);
break;
case 189: /* open and use text from a new file */
if((new_file = fopen(temp,"r")) == NULL)
printf("I can't open %s for reading\n\n",temp);
else
{
print(new_file);
fclose(new_file);
}
break;
case 500: /* .FT print text and wait for line from stdin
and print that line */
if(temp[0] == NULL)
printf("Enter line of TEXT\n");
else
printf("%s\n",temp);
c = readln(0,temp,132);
temp[c-1] = NULL;
sing_line(temp);
Linefeed();
printf("\n");
break;
case 93: /* .CC change control character */
ctrl = num;
break;
case 102: /* .CL comment line */
break;
default:
printf("Unknown operator- code of %d\n\n",dot_c);
break;
}
/*
=====*/

```

Riverside, California. John was the first person to join the group online using the new services available on THE RAINBOW's Delphi Color Computer SIG. His Username is JMFG if you want to say hello.

RAINBOWfest-Palo Alto was a big one for the OS-9 community. Paul Searby gave an inspiring keynote speech at the first OS-9 buffet breakfast attended by more than 60 people. Brian Lantz presented an excellent seminar for OS-9 users and was kind enough to fill in during the first half of my seminar when snow in Washington and rain in California delayed my arrival Saturday.

We saw an interesting approach to OS-9 in a new Winchester BASIC product from Owl Ware in Palo Alto. Interesting idea and we'll be telling you a lot about it with information direct from its author, Alan Reinhart, next month.

The OS-9 community certainly has its heroes and they strive to make your entry into the world of OS-9 Version 2.00 easier. Included in our list of good guys are Ed Bender at PBJ, Dan Johnson at D.P. Johnson and Paul Searby at Computerware. All had to dive for the disassemblers as they hustled to rewrite new drivers that would run on Version 2.00 of OS-9. Next month, we'll take an in-depth look at this new version of OS-9 and try to let you know what you can do with it.

During the evolution, our aforementioned heroes entered several new packages into the utility arena. We'll feature some of the more advanced products, especially Brian Lantz's *kShell*, here next month. Once you use the *kShell*, which is modeled after the Shell in OS-9 68K, you'll never go back. Until then, keep on hacking. □

```

/* cont_proc(c) is passed a character which is to processed as a
control*/
/* charactor, check to see that it is an alpha, convert it to upper
*/
/* case and subtract 64 form it converting it to a control code. This
*/
/* number is the row in the code[row][col] matrix. It is passed to
*/
/* putcont() which will send the code to the output. after returning,
*/
/* it checks to see if the next charactor is also a control char. If
*/
/* it is, it will call itself. if not, it will return the charactor
*/
/* to the file.
*/
int cont_proc(c)
int c;

{
int col;

    if(isalpha(c))
    {
        col= toupper(c) - 64;

    switch(col)
    {
        case 17: /* toggle for control Q */
            if(q_flag == FALSE)
            {
                putcont(17);
                q_flag=TRUE;
            }
            else
            {
                putcont(18);
                q_flag = FALSE;
            }
            break;
        case 19: /* toggle for control S */
            if(s_flag == FALSE)
            {
                putcont(19);
                s_flag = TRUE;
            }
            else
            {
                putcont(20);
                s_flag = FALSE;
            }
            break;
        case 21: /* toggle for controlled underlining */

            if(underline == FALSE)
            {
                putcont(21);
                underline = TRUE;
            }
            else
            {
                putcont(22);
                underline = FALSE;
            }
            break;
        case 23: /* toggle for control W */
            if(w_flag == FALSE)
            {
                putcont(23);
                w_flag = TRUE;
            }
            else
            {
                putcont(24);
                w_flag = FALSE;
            }
            break;
    }
}

```

```

        case 25: /* toggle for control Y */
            if(y_flag == FALSE)
            {
                putcont(25);
                y_flag = TRUE;
            }
            else
            {
                putcont(26);
                y_flag = FALSE;
            }
            break;
        default:
            putcont(col);
            break;
    } /* close out switch */
}
else
    return;
}
/*
=====*/

```

Listing 16: *usage.c*

```

/* usage.c prints out the proper syntax and */
/* available options for printf      */
int usage()
{
    printf("\npf filename [-options] [output path]\n");
    printf("    filename is file to be printed and is required\n");
    printf("    options must be preceded by '-'\n");
    printf("        c = number of copies\n");
    printf("        s = page to start printing\n");
    printf("        e = page to stop printing\n");
    printf("    follow option letter with desired number (no
spaces)\n");
    printf("Default output path is to the printer\n");
}

```

Listing 17: *cput.c*

```

/* cput.c prints the character to the output path only if */
/* the printing flag is true. This provides for partial */
/* printing of documents */
int cput(c)
int c;
{
    if(pr_flag == TRUE)
        putc(c,path);
}

```

Listing 18: *print.mod.c*

```

/* This program accepts a file from standard input */
/* and outputs a file called "prtr.contrl" for */
/* use by a printer formatting program */
#include <ctype.h>;
#include <stdio.h>;

main()
{
    int matrix [27][8];
    int i, j, test, flag;
    char input;
    FILE *input_file, *fopen();

    /* initialize all elements in matrix to 128 */
    for (i=0; i <= 26; ++i)
    {
        for (j=0; j <= 7; ++j)
        {
            matrix[i][j] = 128;
        }
    }
}

```

```

    )
    flag=0; /* flag =0 until a "*" is found */
    while((input = getchar()) != EOF)
    {
        if(input == '*')
            flag=1; /*allows comment line */
        if( input == '+' && flag == 0) /*check for "+" sign */
        {
            flag =1; /* an exception- stop search on this line */
            matrix[0][0] = 0;
        }
        if( input == '=' && flag == 0) /*check for "=" sign */
        {
            flag = 1;
            input = getchar(); /*move by first blank */
            matrix[0][1] = return_int();
        }
        if( isalpha(input) && flag ==0) /* start processing of */
            /* of control letters */
        {
            flag = 1;
            i = toupper(input) - 64; /* convert letter to ascii */
            /* control code */
            j=0;
            input = getchar(); /* skip first blank */
            while((input =getchar()) != '*')
            {
                ungetc(input,stdin); /* if not "*", put char back
                /* on file
                matrix[i][j] = return_int();
                ++j;
            }
            ungetc(input,stdin); /* put "*" back on file*/
        }
        if( input == '\015') /* test for cr */
            flag = 0; /* flag is reset to process next line */
    }
    printf("\nThe printer module file has been read \n");
    /* open and write the contents of matrix to the file */
    /* note-- "wx" will write the file in execution dir */
    if((input_file = fopen("prtr.contrl","wx")) == NULL)
    {
        printf("I can't open prtr.contrl\n");
        exit(1);
    }
    fwrite(&matrix[0][0],sizeof(int),216,input_file);
    fclose(input_file);

}

/* function to get a string and convert it to an integer */
int return_int()
{
    char num_str[5],in;
    int num, i;

    i = 0;

    while(isdigit(in = getchar()))
    {
        num_str[i]=in;
        ++i;
    }
    num_str[i] = NULL;
    num = atoi(num_str);
    return(num);
}

```





# GOLDSOFT

## Hardware & Software for your TANDY computer.

HARDWARE																							
<b>The CoCoConnection:</b>																							
Connect your CoCo to the real world and control robots, models, experiments, burglar alarms, water reticulation systems — most electrical things. Features two MC 6821 PIAs; provides four programmable ports; each port provides eight lines, which can be programmed as an input or output; comes complete with tutorial documentation and software; supplied with LED demonstration unit. Switchable memory addressing allows use with disk controller or other modules via a multipack interface; plugs into Cartridge Slot or Multipack, uses gold plate connectors; a MUST for the hardware designer and debugger!		<b>\$206.00</b>																					
<b>Video-Amp:</b> Connects simply to your CoCo to drive a Colour or Mono monitor.		<table style="width: 100%; border: none;"> <tr> <td style="border: none;">With instructions</td> <td style="text-align: right; border: none;">\$25.00</td> </tr> <tr> <td style="border: none;">With instructions and sound</td> <td style="text-align: right; border: none;">\$35.00</td> </tr> </table>	With instructions	\$25.00	With instructions and sound	\$35.00																	
With instructions	\$25.00																						
With instructions and sound	\$35.00																						
<b>The Probe:</b> A temperature measuring device which attaches to the joystick port of your CoCo or T1000, or to the joystick port of your CoCo Max. Comes with programs to start you thinking, and is supported monthly in Australian CoCo magazine.		<b>\$39.95</b>																					
SOFTWARE																							
<b>Magazines:</b>																							
Australian Rainbow Magazine — THE magazine for advanced CoCo users! Australian CoCo Magazine — THE magazine for the new user of a Tandy computer. Also suits owners of CoCos, MC 10s, Tandy 1000s, 100s, 200s & 2000s.																							
<b>Back Issues:</b>																							
Australian Rainbow Magazine. (Dec '81 to now.) Please Note: Some months out of stock. Australian CoCo Magazine. (Aug '84 to now.) Please Note: Some months out of stock. CoCoBug Magazine. For CoCo — usually 8 programs in each magazine. (Sep '84 to Oct '85) Australian MiCo Magazine. For Tandy MC 10 computers. (Dec '83 to Jul '84) Australian GoCo Magazine. For Tandy Model 100 users. (Jul '83 to Jul '84)		<table style="width: 100%; border: none;"> <tr> <td style="border: none;">Australian Rainbow</td> <td style="border: none;">1986</td> <td style="text-align: right; border: none;">\$4.50</td> </tr> <tr> <td style="border: none;"></td> <td style="border: none;">1982-1985</td> <td style="text-align: right; border: none;">\$2.50</td> </tr> <tr> <td style="border: none;">Australian CoCo</td> <td style="border: none;">1986</td> <td style="text-align: right; border: none;">\$3.75</td> </tr> <tr> <td style="border: none;"></td> <td style="border: none;">Sept 1984-1985</td> <td style="text-align: right; border: none;">\$3.00</td> </tr> <tr> <td style="border: none;"></td> <td style="border: none;">each</td> <td style="text-align: right; border: none;">\$1.00</td> </tr> <tr> <td style="border: none;"></td> <td style="border: none;">each</td> <td style="text-align: right; border: none;">\$2.00</td> </tr> <tr> <td style="border: none;"></td> <td style="border: none;">each</td> <td style="text-align: right; border: none;">\$1.50</td> </tr> </table>	Australian Rainbow	1986	\$4.50		1982-1985	\$2.50	Australian CoCo	1986	\$3.75		Sept 1984-1985	\$3.00		each	\$1.00		each	\$2.00		each	\$1.50
Australian Rainbow	1986	\$4.50																					
	1982-1985	\$2.50																					
Australian CoCo	1986	\$3.75																					
	Sept 1984-1985	\$3.00																					
	each	\$1.00																					
	each	\$2.00																					
	each	\$1.50																					
<b>CoCoOz, on Tape or Disk:</b> The programs you see listed in Australian CoCo Magazine are available on CoCoOz! No laborious typing — just (C)LOAD and Go!		<table style="width: 100%; border: none;"> <tr> <td style="border: none;">Each Tape</td> <td style="text-align: right; border: none;">\$9.50</td> </tr> <tr> <td style="border: none;">Subscription, 6 months</td> <td style="text-align: right; border: none;">\$42.00</td> </tr> <tr> <td style="border: none;">12 months</td> <td style="text-align: right; border: none;">\$75.00</td> </tr> <tr> <td style="border: none;">Each DISK</td> <td style="text-align: right; border: none;">\$10.95</td> </tr> <tr> <td style="border: none;">Subscription on disk, 12 months</td> <td style="text-align: right; border: none;">\$102.50</td> </tr> </table>	Each Tape	\$9.50	Subscription, 6 months	\$42.00	12 months	\$75.00	Each DISK	\$10.95	Subscription on disk, 12 months	\$102.50											
Each Tape	\$9.50																						
Subscription, 6 months	\$42.00																						
12 months	\$75.00																						
Each DISK	\$10.95																						
Subscription on disk, 12 months	\$102.50																						
Back issues of CoCoOz are always available																							
<b>Rainbow on Tape, or Disk:</b> Australian. The programs you see listed in Australian Rainbow Magazine are available on tape. A boon if you don't understand the language! American. We also supply the programs found in American Rainbow on tape. Please specify either Australian or American.		<table style="width: 100%; border: none;"> <tr> <td style="border: none;">Each Tape</td> <td style="text-align: right; border: none;">\$15.00</td> </tr> <tr> <td style="border: none;">Subscription, 12 months</td> <td style="text-align: right; border: none;">\$144.00</td> </tr> <tr> <td style="border: none;">NEW for 1986 ONLY Each DISK</td> <td style="text-align: right; border: none;">\$15.00</td> </tr> <tr> <td style="border: none;">Subscription on disk, 12 months</td> <td style="text-align: right; border: none;">\$172.00</td> </tr> </table>	Each Tape	\$15.00	Subscription, 12 months	\$144.00	NEW for 1986 ONLY Each DISK	\$15.00	Subscription on disk, 12 months	\$172.00													
Each Tape	\$15.00																						
Subscription, 12 months	\$144.00																						
NEW for 1986 ONLY Each DISK	\$15.00																						
Subscription on disk, 12 months	\$172.00																						
<b>MiCoOz:</b> The programs in the MiCo section of Australian CoCo Magazine. (For MC 10 computers only) Back issues of MiCoOz are always available.		<table style="width: 100%; border: none;"> <tr> <td style="border: none;">Each Tape</td> <td style="text-align: right; border: none;">\$9.50</td> </tr> <tr> <td style="border: none;">Subscription, 12 months</td> <td style="text-align: right; border: none;">\$75.00</td> </tr> </table>	Each Tape	\$9.50	Subscription, 12 months	\$75.00																	
Each Tape	\$9.50																						
Subscription, 12 months	\$75.00																						
<b>GOLDDISK 1000</b> — programs from 'softgold' for your Tandy 1000 on disk		<b>\$10.95</b>																					
<b>CoCoLink:</b> CoCoLink is our Bulletin Board which you can access with any computer if you have a 300 baud modem and a suitable terminal program. There is a free visitor's facility, alternatively membership entitles you to greater access of the many files available. We can also be contacted through Viatel (Telecom).		<table style="width: 100%; border: none;"> <tr> <td style="border: none;">Subscription to CoCoLink,</td> <td style="text-align: right; border: none;">\$29.00</td> </tr> <tr> <td style="border: none;">12 months</td> <td style="border: none;"></td> </tr> </table>	Subscription to CoCoLink,	\$29.00	12 months																		
Subscription to CoCoLink,	\$29.00																						
12 months																							
<b>Books:</b>																							
HELP: A quick reference guide for CoCo users.		\$9.95																					
BYTE: Guide for new CoCo users.		\$4.00																					
MiCo HELP: A quick reference for owners of MC 10 computers.		\$9.95																					
<b>Othello:</b> by Darryl Berry The board game for your CoCo.		<table style="width: 100%; border: none;"> <tr> <td style="border: none;">Tape 16K ECB</td> <td style="text-align: right; border: none;">\$15.95</td> </tr> </table>	Tape 16K ECB	\$15.95																			
Tape 16K ECB	\$15.95																						
<b>Say the Wordz:</b> by Oz Wiz & Pixel Software Two curriculum based speller programs for your Tandy Speech/Sound Pack.		<table style="width: 100%; border: none;"> <tr> <td style="border: none;">Tape 32K ECB</td> <td style="text-align: right; border: none;">\$29.95</td> </tr> </table>	Tape 32K ECB	\$29.95																			
Tape 32K ECB	\$29.95																						
<b>Bric a Brac:</b>																							
Blank tapes .....12 for \$18.00 or \$1.70 each																							
Cassette Cases ..... 12 for \$3.50																							
Disks .. (They work!) ..... \$3.50 each or \$29.50 per box of 10.																							

### HOW TO ORDER

- Option 1: Use the subscription form in this magazine.  
 Option 2: Phone and have ready your Bankcard, Mastercard or Visa number.  
 Option 3: Leave an order on Viatel or CoCoLink, but be sure to include your Name, Address, Phone Number, Credit Card Number and a clear indication of what you require, plus the amount of money you are authorising us to bill you.

## The Best of CoCoOz:

### What's on:

#### Best of CoCoOz #1. EDUCATION

ROADQUIZ ..... ROB WEBB  
 HANGMAN ..... ALEPH DELTA  
 AUSTGEOG ..... P. THOMAS  
 SPELL ..... IAN LOBLEY  
 FRACTUT ..... ROBBIE DALZELL  
 ICOSA ..... BOB WALTERS  
 TAXMAN ..... TONY PARFITT

#### Best of CoCoOz #2 part 2. 32K GAMES.

TREASURE ..... DAVISON & GANS  
 MASTERMIND ..... GRAHAM JORDAN  
 ANESTHESIA ..... MIKE MARTYN  
 OREGON TRAIL ..... DEAN HODGSON  
 ADVENTURE ..... STUART RAYNER

#### Best of CoCoOz #2 part 1. 16K GAMES.

LE-PAS ..... Wrongsoft  
 COCOMIND ..... STEVE COLEMAN  
 OILSLICK ..... JEREMY GANS  
 CCMETEOR ..... BOB THOMSON  
 BATTACK ..... JEREMY GANS  
 PROBDICE ..... BOB DELBOURGO  
 CHECKERS ..... J & J GANS

#### Best of CoCoOz #3. UTILITIES.

PAGER ..... ?  
 HI ..... ALEX. HARTMANN  
 SPOOL64K ..... WARREN WARNE  
 CREATITL ..... BRIAN FERGUSON  
 FASTEXT ..... OZ-WIZ  
 DATAGEN ..... ROBIN BROWN  
 SPEEDCTR ..... PAUL HUMPHREYS  
 PRINTSORT ..... PAUL HUMPHREYS  
 BIGREMS ..... BOB T  
 DIR ..... PAUL HUMPHREYS

#### Best of CoCoOz #4. BUSINESS

HI ..... ALEX. HARTMANN  
 (Disk Directory manager)  
 BANKSTAT ..... BARRY HATTAM  
 (Statement annal & store)  
 INSURE ..... ROY VANDERSTEEN  
 (Analyse home contents)  
 SPOOL64K ..... WARREN WARNE  
 (Printer spooler req 64K)  
 2BC ..... WARREN WARNE  
 (Hold 2 sep progs in mem)  
 DATABASE ..... PAUL HUMPHREYS  
 (THE tape database)  
 RESTACC ..... DUNG LY  
 (Tape restruant accounts)  
 PRSPDSHT ..... GRAHAM MORPHETT  
 (Disk print out SPDSHEET)

Best of CoCoOz #5. Adventure Games  
 Best of CoCoOz #6. Preschool Education

Coming in July - Best of CoCoOz #7, Grafix!

MARKET ..... ALEPH DELTA  
 TOWNQUIZ ..... ROB WEBB  
 ALFABETA ..... RON WEBB  
 TANK ADDITION ..... DEAN HODGSON  
 TABLES ..... BARRIE GERRAND  
 KIDSTUFF ..... JOHANNA VAGG  
 FLAGQUIZ ..... ROB WEBB

SHOOTING GALLERY ..... TOM DYKEMA  
 GARDEN ..... DAVE BLUHDORN  
 YAHTZEE ..... KEVIN GOWAN  
 BATTLESHIP ..... CHRIS SIMPSON  
 ANDROMIDA ..... MAX BETTRIDGE

PYTHON ..... ?  
 POKERMCH ..... GRAHAM & MATTHEWS  
 SPEEDMATH ..... DEAN HODGSON  
 LNDATTCK ..... ALDO DEBERNARDIS  
 INVADERS ..... DEAN HODGSON  
 RALLY ..... TONY PARFITT  
 FOURDRAW ..... JOHANNA VAGG

COPYDIR ..... THOMAS SZULCHA  
 LABELLER ..... J.D. RAY  
 SCRPT ..... TOM DYKEMA  
 MONITOR+ ..... BRIAN FERGUSON  
 BEAUTY ..... BOB T  
 PCOPY ..... B. DOUGAN  
 RAMTEST ..... TOM DYKLEMA  
 DISKFILE ..... B. DOUGAN  
 LABEL ..... F. BISSELING

PERSMAN ..... PAUL HUMPHREYS  
 (Personal finance management)  
 CC5 ..... GRAHAM MORPHETT  
 (Sales Invoicing-tape sys)  
 COCOFILE ..... BRIAN DOUGAN  
 (Tape data base)  
 DPMS ..... PAUL HUMPHREYS  
 (Disk Program Management Sys)  
 4OKGREY ..... RAY GAUVREAU  
 (40K Basic for grey 64K CoCo)  
 TAXATION ..... ?  
 (Calc tax payable)  
 SPDSHEET ..... GRAHAM MORPHETT  
 (Disk 22 coloum spreadsheet)  
 ACS3 ..... GREG WILSON  
 (Multi disk data base)

Tape \$10.00  
 Disk \$21.95

Tape \$10.00  
 Disk \$21.95

Tape \$10.00  
 Disk \$21.95

Tape \$10.00  
 Disk \$16.00

Tape \$10.00  
 Disk \$21.95

**Specials**  
 Any two tapes \$17.00  
 Any 3 tapes \$20.00

Two Disks or more each \$16.00

# LOOK AT THIS

C30 Cassettes .20c ea!!  
 (Minimum order \$6.00)

1981 & 1982 For Sale!!  
 Dec 81, Mar 82, Sep 82,  
 Oct 82, Dec 82.

Just a few issues left!  
 Quick!! Once they're gone,  
 they're gone! \$1.00ea

### "Best of" Series:

ALL disks \$16.00ea  
 or 2 @ \$14.00ea

### Best of Education (#1)

now \$6.50 tape only.

### Best of Games Pt 1, 16K Games

now \$6.50 tape only.

### July Only:

The CoCoConnection \$185.00

Available only from Goldsoft  
 PO BOX 1742  
 Southport, QLD. 4215.

# User Group Contacts

(Stop between numbers = b.h. else  
a.h.; but, hyphen between = both.)

ACT:  
CANNBERRA NTH JOHN BURGER 062 58 3924  
CANNBERRA STH LES THURBON 062 88 9226

NSW:  
SYDNEY:  
BANKSTOWN CARL STERN 02 646 3619  
BNKSTWN WEST ARTH PITTARD 02 72 2881  
BLACKTOWN KEITH GALLAGHER 02-627-4627  
CARLINGFORD ROSKO MCKAY 02 624 3353  
CHATSWOOD BILL O'DONNELL 02 419 6081  
CLOYTON HERMAN FREDRICKSON 02 6236379  
GLADESVILLE MARK ROTHWELL 02 817 4627  
HILLS DIST ARTHUR SLADE 02 622 8940  
HORNSBY ATHALIE SMART 02 848 8830  
KENTHURST TOM STUART 02 654 1610  
LEICHHARDT STEVEN CHICOS 02 560 6207  
or GORGE ECHEGARAY 02 560 9664  
LIVERPOOL LEONIE DUGGAN 02-607-3791  
MACQUARIE FIELDS

BARRY DARNTON 02 618 1909  
ROSEVILLE KEN UZZELL 02 467 1619  
SUTHERLAND IAN ANNABEL 02 528 3391  
SYDNEY EAST JACKY COCKINOS 02 344 9111  
ALBURY RON DUNCAN 060 43 1031  
ARMIDALE DOUG BARBER 067 72 7647  
BLAXLAND BRUCE SULLIVAN 047 39 3903  
BROKEN HILL TERRY NOOMAN 080 88 2382  
CAMDEN KEVIN WINTERS 046.66.8068  
COFFS HARBOUR BOB KENNY 066 51 2205  
COOMA ROSS PRATT 0648 23 065  
COORAMBONG GEORGE SAVAGE 049 77 1054  
COOTAMUNDRA CHERYL WILLIS 069 42 2264  
DENILQUIN WAYNE PATTERSON 058 81 3014  
DUBBO GRAEME CLARKE 068 89 2095

or MIKE MUNRO 068-82-5011  
JOHANNA VAGG 068 52 2943  
FORBES GARY BAILEY 065 54 5029  
FORSTIER PETER SEIFERT 043 32 7874  
GOSFORD PETER LINDSAY 066 42 2503  
GRAFTON MICHAEL J. HARTMANN 067 79 7547  
GUYRA PAUL MALONEY 069 24 1860

JUNEE RICK FULLER 065-62-7222  
KEMPSEY BRETT WALLACE 069-53-2081  
LEETON ROB HILLARD 066 24 3089  
LISMORE DAVID BERGER 063 52 2282  
LITHGOW BILL SNOW 049 66 2557  
MAITLAND ALF BATE 067 52 2465  
MORÉE BRIAN STONE 063-72-1958  
MUDGEÉ WENDY PETERSON 065 68 6723

NAMBUCCA HDS GRAEME CLARKE 068 89 2095  
NARROMINE LYN DAWSON 049 49 8144  
NEWCASTLE ROY LOPEZ 044 48 7031  
NOWRA JIM JAMES 063 62 8625  
ORANGE DAVID SMALL 068 62 2682  
PARKES RON LALOR 065 83 8223  
PORT MACQUARIE DAVID SEAMONS 047 51 2107

SPRINGWOOD ROBERT WEBB 067 65 7256  
TAMWORTH GARY SYLVESTER 046 81 9318  
TAHMOOR TERRY GRAVOLIN 065 45 1698  
UPPER HUNTER FRANK MUDFORD 067 78 4391  
URALLA CES JENKINSON 069 25 2263  
WAGGA WAGGA JOHN WALLACE 043 90 0312  
WYONG

NT:  
DARWIN BRENTON PRIOR 089.81.7766

QLD:  
BRISBANE:  
BIRKDALE COLIN NORTH 07 824 2128  
BRASSALL BOB UNSWORTH 07 201 8659  
EAST ROB THOMPSON 07 848 5512  
IPSWICH MILTON ROWE 07 281 4059  
NORTH JACK FRICKER 07 262 8869

WA:  
PERTH IAN MACLEOD 09 448 2136  
KALGOORLIE TERRY BURNETT 090.21.5212

PINE RIVERS BARRY CLARKE 07 204 2806  
SOUTH WEST BOB DEVRIES 07 375 3161  
SANDGATE MARK MIGHELL 07 269 5090  
SCARBOROUGH PETER MAY 07 203 6723  
BIGGENDEN ALAN MENHAM 071 27 1272  
BLACKWATER ANNIE MEIJER 079.82.6931  
BOWEN TERRY COTTON C/O 077 86 2220  
BUNDABERG RON SIMPKIN C/O TANDY  
CAIRNS GLEN HODGES 070 54 6583  
DALBY ANDREW B. SIMPSON 074.62.3228  
GLADSTONE CAROL CATHCART 079 78 3594  
GOLD COAST GRAHAM MORPHEIT 075 51 0015  
HERVEY BAY LESLEY HORWOOD 071 22 4989  
MACKAY LEN MALONEY 079511333x782  
MARYBOROUGH NORM WINN 071 21 6638  
MT ISA PAUL BOUCKLEY-SIMONS 077 43 6280  
MURGON PETER ANGEL 071 68 1628  
ROCKHAMPTON KEIRAN SIMPSON 079 28 6162  
TARA STEVEN YOUNGBERRY  
TOOVOOMBA GRAHAM BURGESS 076 30 4254  
TOWNSVILLE JOHN O'CALLAGHAN 077 73 2064  
WHITEROCK GLEN HODGES 070 54 6583

SA:  
ADELAIDE JOHN HAINES 08 278 3560  
NORTH STEVEN EISENBERG 08 250 6214  
GREENACRES BETTY LITTLE 08 261 4083  
MORPHEITVALE KEN RICHARDS 08 384 4503  
PORT NOARLUNGA ROB DALZELL 08 386 1647  
SEACOMBE HTS GLENN DAVIS 08 296 7477  
PORT LINCOLN BILL BOARDMAN 086 82 2385  
PORT PIRIE KEVIN GOWAN 086 32 1368  
WHYALLA MALCOLM PATRICK 086 45 7637

TAS:  
HOBART BOB DELBOURGO 002 25 3896  
KINGSTON- WIM DE PUIT 002 29 4950  
WYNYARD ANDREW WYLLIE 004 35 1839

VIC:  
MELBOURNE:  
MELBOURNE CCC JOY WALLACE 03 277 5182  
DANDENONG DAVID HORROCKS 03 793 5157  
DONCASTER JUSTIN LIPTON 03 857 5149  
FRANKSTON BOB HAYTER 03.783.9748  
NARRE WARREN LEIGH EAMES 03 704 6680  
NTH EASTERN KEVIN KAZAZES 03 437 1472  
MELTON MARIO GERADA 03 743 1323  
RINGWOOD IVOR DAVIES 03 758 4496  
SUNBURY JACK SMIT 03.744.1355

BAIRNSDALE COLIN LEHMANN 051 57 1545  
BALLARAT MARK BEVELANDER 053 32 6733  
CHURCHILL GEOFF SPOWART 051 22 1389  
EMERALD LEIGH EAMES 059 68 3392  
GEELONG DAVID COLLEN 052 43 2128  
HASTINGS MICHAEL MONCK 059 79 2879  
MAFFRA MAX HUCKERBY 051 45 4315  
MOE JIMMY WELSH 051 27 6984  
MORWELL GEORGE FRANCIS 051 34 5175  
SALE BRYAN McHUGH 051 44 4792  
SHEPPARTON ROSS FARRAR 058 25 1007  
SMYTHESDALE TONY PATTERSON 053 42 8815  
SWAN HILL BARRIE GERRAND 050.32.2838  
TONGALA TONY HILLIS 058 59 2251  
TRARALGON MORRIS GRADY 051 66 1331  
WONTHAGGI LOIS O'NEARA 056 72 1593  
YARRAWONGA KEN SPONG 057 44 1488

WA:  
PERTH IAN MACLEOD 09 448 2136  
KALGOORLIE TERRY BURNETT 090.21.5212

CANADA - CoCo:  
Ontario Richard Hobson 416 293 2346

## SPECIAL INTEREST GROUPS

BUSINESS:  
BRIZBIZ BRIAN BERE-STREETER 07 349 4696

OS9 GROUPS:  
NATIONAL OS9 USERS' GROUP  
GRAEME NICHOLS 02 451 2954

NSW  
SYDNEY  
BANKSTOWN CARL STERN 02 646 3619  
CARLINGFORD ROSKO MCKAY 02 624 3353  
GLADESVILLE MARK ROTHWELL 02 817 4627  
SYDNEY EAST JACKY COCKINOS 02.344.9111  
COOMA FRED BISSELING 0648 23263

QLD  
BRISBANE JACK FRICKER 07 262 8869

VIC  
LATROBE VLY GEORGE FRANCIS 051 34 5175

WA  
KALGOORLIE TERRY BURNETT 090.21.5212

MC-10 GROUPS:  
LITHGOW DAVID BERGER 063 52 2282  
ORANGE DAVID KEMP 063 62 2270  
PORT LINCOLN BILL BOARDMAN 086 82 2385  
ROCKHAMPTON TIM SHANK 079 28 1846  
SYDNEY RAJA VIJAY 02 519 4106  
WARRNAMBOOL GARY FURR 055 62 7440

TANDY 1000 / MS DOS:

QLD:  
BRISBANE  
NORTH BRIAN DOUGAN 07 30 2072  
SOUTH BARRY CAWLEY 07 390 7946  
GOLD COAST GRAHAM MORPHEIT 075 51 0015

VIC:  
MELBOURNE TONY LLOYD 03 500 0878

NSW:  
GLADESVILLE MARK ROTHWELL 02 817 4627  
SYDNEY WEST ROGER RUTHER 047.39.3903  
WYONG JOHN WALLACE 043 90 0312

FORTH:  
BRISBANE JOHN POXON 07 208 7820  
PORT LINCOLN JOHN BOARDMAN 086 82 2385  
SYDNEY JOHN REDMOND 02 85 3751

ROBOTICS:  
BOWEN TONY EVANS 077 86 2220  
GOLD COAST GRAHAM MORPHEIT 075 51 0015  
TAMWORTH ROBERT WEBB 067 65 7256  
WAGGA WAGGA CES JENKINSON 069 25 2263

CHRISTIAN USERS' GROUP:  
COLLIE RAYMOND L. ISAAC 097 34 1578

300 BAUD BULLETIN BOARDS  
SYDNEY:  
INFOCENTRE 02 344 9511  
TANDY ACCESS 02 625 8071  
THE COCO - CONNECTION 02 618 3591  
DAIL DUBBO (6pm - 8am) 068 82 5011

QLD:  
CoCoLink 075 32 6370

WA:  
COCO UG 09 307 1397

1200/75 BAUD TANDY INFORMATION  
VIATEL:  
GOLDLINK #642#

# GOLDDIINK

a Goldsoft Service

ON



\*642 #

AUSTRALIAN RAINBOW MAGAZINE

Registered by Australia Post -

Publication No. QBG 4009

AUSTRALIAN CoCo / softgold

Publication No. QBG 4007

P.O. BOX 1742

SOUTHPORT, QLD. Australia. 4215.

POSTAGE  
PAID  
AUSTRALIA