Subscription Fees

- ❑ 6 issues USA $35
- ❑ 6 issues Canada/Mexico $42.50
- ❑ 6 issues other countries surface mail
  - ___ Surface mail $40
  - ___ Air mail $52

Outside U.S., pay via postal or international money order or credit card; personal checks from non-U.S. banks will be returned.

ADDRESS CHANGES: Subscribers who move may have the delivery of their most recent issue(s) delayed unless MICROpendium is notified six weeks in advance of address changes. Please your old address as it appears on your mailing label when making an address change.

Check each item ordered (or list on separate page and enter total amount here

_____

Check/MO ❑ ❑ ❑

Credit Card Number

_____

Expiration Date_____
(Minimum credit card order is $9)

Signature_____
(Required on credit card orders.)
No sales tax on magazine subscriptions. Texas residents add 7.75% sales tax on other items, including back issues and disk subscriptions. Credit card orders add 5%.

Name_____

Address_____

City_____ State____ ZIP_____

The set of numbers at the top of your mailing label indicates the cover date of the last issue of your subscription.

PERIODICALS

Micropendium
P.O. Box 1343
Round Rock, TX 78680

A 1/99
CHARLES GOOD
P.O. BOX 647
VENEDOCIA OH 45894

Covering the TI99/4A and Geneve home computers

MICROpendium

Volume 15 Number 1 January/February 1998 $6

Protection schemes

No matter how good they are, someone always defeats them

Extended BASIC

3TO5COLCAT offers convenience of multi-column disk catalogs

The Art of Assembly

The ins and outs of Instances

Beginning c99

Dealing with memory requirements

Graphics

Transferring digitized graphics between a PC and a TI

Hardware projects

- Using BASIC and a computer to control 'Stamps'
- Connecting a third drive to your TI

MICRO Reviews

Sound Generator

Byte Magazine and Scott Foresman page scans

JM Base

CONTENTS

MICROpendium

MICROpendium (ISSN 10432299) is published bimonthly for $35 per year by Burns-Koloen Communications Inc., 502 Windsor Rd., Round Rock, TX 78664-7639. Periodical postage paid at Round Rock, Texas. POSTMASTER: Send address changes to MICROpendium, P.O. Box 1343, Round Rock, TX 78680-1343.

All correspondence should be mailed to MICROpendium at P.O. Box 1343, Round Rock, TX 78680.

Foreign subscriptions are $42.50 (Canada and Mexico); $40 surface mail to other countries; $52 airmail to other countries.

All editions of MICROpendium are mailed from the Round Rock (Texas) Post Office.

Mailing address: P.O. Box 1343, Round Rock, TX 78680.

Telephone & FAX: (512) 255-1512

Internet E-mail: jkoloen@earthlink.net

Home page: http://www.earthlink.net/~jkoloen/

John Koloen — Publisher

Laura Burns — Editor

COMMENTS

The troubles we've seen

You wouldn't believe the problems we had producing the November/December 1997 edition of MICROpendium. What you probably know is that the last issue of the year came out in mid-December instead of mid-November as we had planned.

If something could go wrong, it did. First, our main laser printer went out. Then our backup laser printer went out. Then a hard drive containing the MICROpendium layout failed.

Beyond the hardware problems, we published an Extended BASIC program called Load Maker that came out all messed up. It contained embedded code which did not come through the translation process from program to text file. I tested the program prior to converting it to text and it worked fine. But I didn't have a chance to closely read it on the layout since by then I was dealing exclusively with hardware problems. We're including a working version of the program on the January/February MICROpendium disk.

I think the hardware problems are behind us for now. We installed a new hard drive so I think we're covered on that score. The printers are both working but the real test comes when we print out the thousands of sheets that we collate into MICROpendium. I know I'll have my fingers crossed for a couple of weeks in January.

Along with keeping my fingers crossed, we're also printing only 48 pages this issue. If we have another hardware breakdown, it will be easier (and cheaper) to produce a smaller publication. If it works out, then we'll be back to 56 pages next time. We're sorry for the inconvenience caused by these problems, but there's not much we can do about them.

(Just as we started printing out this edition, we ran into problems. Our backup refuses to take data from our Mac and the primary printer is back in the shop to have what called a "PCU board" replaced. If this edition is late in getting to you, you'll know why.)

Fest West '98

Fest West '98 is shaping up to be one of the most promising TI events of the year. It certainly will be unique. Congratulations are due to Tom Wills for coming up with the idea of a TI "reunion" at the birthplace of the TI99/4A in Lubbock. Tom and the SouthWest 99ers user group deserve a great deal of credit for turning this ambitious dream into a reality. It's not likely to ever happen again, so this is probably the one opportunity TIers will have to congregate in the city where it all began, not to mention talk with other TIers and perhaps find a bargain or hard-to-find item from vendors. See you in Lubbock.

—JK

Stick around!

I don't recall writing you very often, but want to express my concern for the problems that are facing your publication. I know without you and those who help you that the TI would have long been gone. I am sorry that the advertisers are not supporting you and do want to see a publication as long as possible.

I have faced the problems of semi-retirement and am working, but am on deferred retirement and working two days a week. My letter is without a point so far, so the point I want to make is: One day in the future I am going to ask you to help me make a program that no one else has ever asked for and I hope you will still be there to make it for me.

EUGENE BARRETT
REDDING, CALIFORNIA

Clearing up some V9T9 questions

I read with some interest the article about V9T9 in the November/December issue. Although I've never used V9T9 myself, I can clear up some of the questions asked on pages 13 and 14.

First on the issue of names for memory image (program) files. If a program file has more than one part, it gets divided into files with sequential file names. Thus TISLAND, TISLANE and TISLANF are a set of three such program files. A loader for such files uses the first name as the base, then increments the last character to "look for" the next file in the series. This process works no matter what that last character in the first file name is. Thus TISLAN1, TISLAN2 and TISLAN3 would work just as well in the case mentioned.

Each such program file has a six-byte header at the very beginning of the file, which the loader uses. If the first two bytes are >FFFF, the loader "knows" that this is not the last file in the series, so it has to increment the file name's last character to load another in the series. The last file in a series always has >0000 in the first two bytes of its header, indicating that no more files need to be loaded. The next two bytes of this header are the length of this file's contents (plus six for the header), which tells the loader how much to read from the VDP buffer into main memory. The last two bytes of the header are the loading point for this file, which tells the loader where in main memory to load this section.

In a series of this kind, the load point of the first one is always the entry point, and that's where the loader branches to after the last file in the series has been loaded.

The file names for all but the first one are derived by the loader, and the lengths and load points are in the header of each file in the series. The loader thus bases its actions on the first file name and the header contents in each file.

Also on page 14 were questions concerning the meanings of some CALL LOAD operations. Here's what

those mean. CALL LOAD(8196,254,0) sets the address for where CALL LINK will search for the linkage names; 8196 decimal is the address >2004 in the low memory. In this case, it means that CALL LINK will start its search at >FE00 (254,0 = >FE00) in the high memory. Presumably the listing for the routine named LOAD and its starting point is at that location. LINK then passes the first file name of the series along to the code via a normal STRREF process. The three files TISLAND, TISLANE, TISLANFare most probably *not* GROM stuff, but machine code segments that load into and run in high memory, with TISLAND loading at >A000, TISLANE at >BFAA and TISLANF at >DFF4. This is not necessarily fact, just a very well-educated guess on my part.

The much more mysterious one is the 1 to 4 loop which loads to addresses starting at -6144, and going to that number times two, three and four as well. The number -6144 is equivalent to >E800 in the high memory. That number times two, three and four becomes >D000, >B800 and >A000, respectively. It's not immediately obvious why the loader would need those locations set to zero. Perhaps the code in the loader checks those for zero to confirm whether loading of the program files was accomplished correctly.

I hope this clarifies a few things for Randy and others of our readers. Not having tried V9T9 myself, I can't make heads or tails of the rest of that article, but I thought I should clarify those things I do understand.

BRUCE HARRISON
HYATTSVILLE, MARYLAND

'Texas tenacity' praised

I still read through every edition, even though my TI99s and Geneves are packed away, as they have been for the past three or four years. I retired from the Navy in '95, on the West Coast, and then spent a year in college (just to give me somewhere to go each day) studying electronics technology. I have also moved twice in the past four years.

Now I'm digging out my old toys, because I miss them, and I have more time to play with them now.

Any road, I want to say that it must be some kind of stubborn Texan tenacity that keeps you folks going, and I hope it continues as long as there is a TI99 left with a spark in it.

There are no active TI user groups that I know of in Canada anymore, but perhaps you could publish a note in your next issue about how many subscribers you have or users that you know about in various countries. It would be nice to know how many of us there are and, if possible, who might be

nearby. (e.g., for me, the Maritime Provinces & Maine).

Have a great holiday season; I look forward to getting my Jan/Feb 2000

Continued on page 6

FEEDBACK

Continued from page 5

issue.

GEOFF FRUSHER
DARTMOUTH, NOVA SCOTIA, CANADA

There's a fellow Nova Scotian and a man in New Brunswick, plus a couple of folks in Maine. We are somewhat reluctant to print addresses without permission. Canada has at least one active users group, but the one we know about is in Ontario. — Ed.

BUGS AND BYTES

Tesch modifies Geneve using 512K chips

In late November Tim Tesch revealed to subscribers of the TI list server (ti99@TheRiver.com) that he successfully modified a Geneve using a 512K memory chip. Previously, the memory upgrade consisted of three 128K chips for a toal of 384K. Tesch says the 512K upgrade requires additional wiring but does not require stacking chips and modifing the Geneve case to accomodate the triple stack.

"I have not decided if this is the easier and more time effective of the two methods, but I have the option," he said. He planned to continue testing the unit and then converting another Geneve for a followup test.

Modified DM3 works on Geneve

Dan Eicher posted the following on the TI list server in late December. —Ed.

Jerry C. tipped me off to the fact, DM3 was modified to work on the Geneve by Mike Dodd. I found a copy of this software on Beery Millers CDROM. I pulled this program and put it up on the Hugger BBS (317)782.9942.

From the bit of testing I have done, it works well with the Geneve, both floppy and ramdisk... If any of you have TI's and Myarc Floppy disk, I'd be interested in hearing about your results of use of this program, or, if you have a Geneve and a non-myarc controller! Eicher can be reached at EICHER@delphi.com.

c99 corrections

A couple of problems have been found in two Beginning c99 columns.

In the September/October column, the "TEXTFUN" program had a line of the source code which was wrapped, due to the new MICROpendium format. However, this line should **not** be typed in as two separate lines. The line in question is the introS[] array initializer. The text "This is an example of an array containing text."; should all appear on one line.

In the November./December issue, the Power function has a line which reads "power-;". This should have two minus signs, making it "power--;" (i.e. decrement the power variable).

OBITUARY

Cancer claims Ray Kazmer

Ray Kazmer of Sunland, California, died Sept. 9. He was known for his graphics and animation programs and contributed a number of articles and User Notes to MICROpendium.

In 1988, Kazmer spearheaded a drive for donations of TI equipment to help Sister Pat Taylor of Dubuque, Iowa, in her work with elderly residents of the nursing home where she lives.

He grew up in Ohio. He was an Air Force veteran and worked as a professional photographer. He also wrote screenplays.

Ken Gilliland, of Notung Software, for which Kazmer produced the program Star Trek Calendar, says:

"In mid-August, Ray was diagnosed with cancer in the prostate, liver and kidneys. Over the next several weeks, I and Bobbi (his ex-wife) helped him through the many bumps and difficult issues of settling matters. He slipped away quickly in early September due to complications from his ailments. I might add that in his conscious moments, he was the 'typical' Ray, kidding around and making the nurse's lives hell and not surprisingly, one of the very last words he breathed was 'computer'."

Gilliland adds, "After his passing, he was immediately cremated and his ashes left in my care for scattering, per his request."

EXTENDED BASIC

3TO5COLCAT offers convenience of multi-column disk catalogs

BY LEONARD W. TAFFS
SW99ERS, TUCSON, AZ.

Who needs a five-column cataloger that lists files downward in alphabetical order? Who needs yet another disk cataloger? The answer is people like me.

Why? Because my eyes tire from having to read the printout of catalogers that list alphabetically three or four across, as in MASTERDISK and Jack E. Evans Sr.'s three-column Cataloger. When you have more than 30 or 40 files on a disk, it is possible to miss seeing a title in reading across. And what about a disk like Nuts and Bolts by Jim Peterson that exceeds 100 files?

Most utility catalogers (DM1000, WRITER programs, etc.) print &downward but are limited to 1 or 2 columns. With up to 100 files or more you have either a long tape, or a page to have to fold to stick in the disk jacket.

To meet this need I wrote a program that will print all files downward in alphabetical order in your choice of three, four, or five columns. Printer codes/commands used are as for an NX10 printer. Change if necessary to suit your printer.

3TO5COLCAT (after entering disk number to catalog, and entering a date — date is optional — stores the disk content files in an array. You'll see the titles scrolling up your screen. When it has read a disk, you are given the option of reading the titles as many times as you wish without having to print a hardcopy (V/P).

Not shown on screen is the fact that you can choose another disk at this point by pressing "Q," after which you are prompted for another disk. Entering "Y" for "yes" and pressing Enter will rerun the program from beginning.

If, at the "V/P" prompt, you enter "P" you are asked to choose "3, 4, or 5 columns." Not shown on screen here is the fact you can escape from starting printing by pressing zero or 9. Pressing zero will return you to the option of viewing the contents again. Pressing 9 will terminate the program.

Note that a separate CALL KEY is included which allows you to stop the array scrolling temporarily by pressing any key when reviewing arrays.

When you have chosen to review the array the first time you are prompted to "Read Again or Print (A/P)." Press "A" to read again, or "P" to print your catalog.

When you have selected the number of columns you wish, then the screen informs you that the computer is initializing arrays for printing.

At this time the arrays are being set to appropriate columns. With a large number of files this will take a few seconds — a counter is displayed showing you the array counting (so you won't think your computer has locked up). If your printer is not hooked up, or is not turned on, or if off-line, the arrays will be processed for printing showing the counts, but nothing will happen and the program cannot continue until you have activated the printer or put it back on-line.

Finally you are prompted for another disk.

Note the arrays RS1$ through RS5$ in line 10 are set to 32. This assumes you will choose three, four, or five columns appropriate to the number of records you have. This means you will need to set these arrays to higher values if you choose to print more than 96 files in three-column fashion. The whole purpose of this program was to make it feasible to print the largest number of files in as little space as possible. It doesn't make a lot of sense to print 120 files in three columns when you'll have a much more compact listing in four or, even better, in five. For instance, Nuts and Bolts disk number 3 fits very nicely in five columns, on a sheet of paper, when folded, fits a disk jacket very nicely.

3TO5COLCAT uses conventional spacing and condensed print. By changing the appropriate printer lines you could get closer spacing. By changing print style you could even narrow the columns. But the program, as is, does what I wanted, so here it is.

As shown at the end of the program, I would appreciate any comments on this program.

3TO5COLCAT

```
1 ! [3TO5COLCAT] 10-17-97
2 !
3 ! 3 or 4 or 5 Column Disk
    Cataloger
4 !
5 ! by W. LeonardTaffs
    SW99ers, Tucson, Az.
7 !
8 ! For EXTENDED BASIC
9 !
10 DIM AR$(127),RS1$(32),RS2
$(32),RS3$(32),RS4$(32),RS5$
(32)
20 GOTO 100
30 A,A$,A2,A2$,A3,ADJ,AG$,B,
B$,C,C3,C4,C5,D,DATE$,DSC$,D
SK$,F,G,I,I$,I2,INC,INC$,INC
2,J,J1,J2,J3,J4,J5,I,K,K1,K1
$
40 L,L$,M,M$,N,P,P$,PR,R3$,R
4$,R5$,FS1$,FS2$,FS3$,FS4$,F
S5$,S,STP,T1$,T2$,T3$,T4$,T5
$,X,X$,X2,Z
50 CALL CLEAR :: CALL SCREEN
 :: CALL KEY
60 !@P-
100 CALL CLEAR :: CALL SCREE
N(4):: DISPLAY AT(2,8):"DISK
 CATALOGER"
110 DISPLAY AT(4,1):"By W. L
eonard Taffs, SW99ers"
120 DISPLAY AT(9,1):"Print 3
```

Continued on page 10

EXTENDED BASIC

EXTENDED BASIC

Continued from page 9

```
 to 5 Column CATALOGS": :"(C
olumns Down in Alfa Order)"
130 INPUT "DISK: ":DSC$ :: D
SK$="DSK"&DSC$&"." :: PRINT
:: INPUT "DATE: ":DATE$ :: C
ALL CLEAR
140 ! DIM AR$(127),T$(5)
150 T1$="D/F" :: T2$="D/V" :
: T3$="I/F" :: T4$="I/V" ::
T5$="PROGRM" :: R3$=RPT$("="
,77):: R4$=RPT$("=",107):: R
5$=RPT$("=",130)
160 M$=RPT$(" ",23):: M=LEN(
M$):: OPEN #1:DSK$,INPUT ,RE
LATIVE,INTERNAL
170 INPUT #1:A$,B,C,D
180 DSK$=A$&" US:"&STR$(C-D+
2)&" AV:"&STR$(D):: PRINT "D
SK: ";DSK$: : :: AR$(0)=DSK$
190 INPUT #1:A$,J,K,L :: INC
=INC+1 :: INC$=STR$(INC):: C
=LEN(INC$):: IF C=1 THEN INC
$="  "&INC$ ELSE IF C=2 THEN
 INC$="  "&INC$
200 IF A$="" THEN 330
210 ! J=TYPE   L=PARAM
220 K1$=STR$(K):: K1=LEN(K1$
):: IF K1=3 THEN K1$=K1$ ELS
E IF K1=2 THEN K1$="  "&K1$ E
LSE IF K1=1 THEN K1$="  "&K1
$
230 A2$=RPT$(" ",11):: A2=LE
N(A2$):: A=LEN(A$):: A3=A2-
A
 :: A$=A$&SEG$(A2$,1,A3)240
L$=STR$(L):: L=LEN(L$):: IF
L=2 THEN L$="  "&L$
250 IF J<0 THEN J=J*-1 :: P$
="  P" ELSE P$=""
260 IF J=1 THEN L$=T1$&L$ EL
SE IF J=2 THEN L$=T2$&L$ ELS
E IF J=3 THEN L$=T3$&L$ ELSE
 IF J=4 THEN L$=T4$&L$
270 IF J<>5 THEN B$=A$&K1$&"
 "&L$&P$ :: B=LEN(B$):: N=M-
B :: B$=B$&SEG$(M$,1,N)
280 IF J<>5 THEN AR$(INC)=B$
 :: PRINT INC$;"  ";B$ :: GOT
O 320
290 IF J=5 THEN L$=T5$
300 IF J<5 THEN B$=A$&K1$&"
 "&L$ :: PRINT INC$;"  ";B$ ::
 AR$(INC)=B$ :: GOTO 320310
B$=A$&K1$&"  "&L$&P$ :: B=LEN
(B$):: N=M-B :: B$=B$&SEG$(M
$,1,N):: PRINT INC$;"  ";B$ :
: AR$(INC)=B$
320 GOTO 190
330 REM ** EOA **
340 CLOSE #1
350 PRINT :"Disk Read.";INC-
1;"Files.": :
360 DISPLAY AT(24,1):"VIEW A
RRAY OR PRINT? (V/P)": :
370 CALL KEY(0,K,S):: IF S<1
 THEN 370
380 IF (K=86)+(K=118)THEN 39
0 ELSE IF K=13 THEN 370 ELSE
 IF (K=81)+(K=113)THEN 850 E
LSE 510
390 ! ** VIEW ARRAY **
400 CALL CLEAR :: A=1 :: DIS
PLAY AT(24,1):AR$(0): :
410 FOR I=1 TO INC
420 I$=STR$(I):: I2=LEN(I$):
```


EXTENDED BASIC

```
: IF I2=1 THEN I$="  "&I$ EL
SE IF I2=2 THEN I$="  "&I$
430 IF AR$(I)="" THEN 450
440 DISPLAY AT(A,1):AR$(I)::
 DISPLAY AT(A+2,1):RPT$(" ",
28):: A=A+1 :: IF A=23 THEN
A=1
```

Sample catalog output to a laser printer

```
      US:226 AV:1214    9 Files   1/2/97
--------------------------------------------------------------------------
3TO5-2      22 D/V163 ||  3TO5COLCAT  20 PROGRM || GIFTRANSFR  23 D/V 80 P
3TO55       26 D/V 80 ||  3TO5COLDOC  18 D/V 80 || SIDEBAR68   16 D/V 80
3TO5:LST80  32 D/V 80 ||  GIFTPHOTO    2 D/V 80 || TUT68       65 D/V 80
```

NEWSBYTE

Harrison upgrades three programs

Bruce Harrison has upgraded three of his programs, Font Designer, AMS Video Titler and AMS Slideshow.

Font Designer for those with 24-pin or Bubble Jet printers now has a built-in catalog function to make it easier to find the font files for editing or downloading, according to Harrison. The catalog function shows only files of the D/V 120 type, as created and used by this program.

The AMS Video Titler also no has a built-in catalog which will reveal only files suitable for use by the program, Harrison says. That includes TI-Artist pictures and the Harrison Drawing program's picture files. In both cases, after the catalog is presented, a selection cursor appears on the catalog screen with which the user can select a file from the list.

The AMS Slideshow's catalog function has been upgraded so it can show up to 127 file names on three screens, Harrison says. Also, the Time Delay between pictures can be zero for rapid change, a Pause feature allows pressing the space bar during timed showings to freeze the program on the current picture until the space bar's release, and the program now shows on the screen how many files have been selected for the "show."

All three programs are compatible with SCSI or other hard drives and have updated instructions on the disks, according to Harrison. The two AMS programs have been updated to be compatible with AMS cards up to 1 meg capacity, and automatically tailor themselves to the size of AMS in use.

Upgrades are $1 each, including shipping and handling. For new buyers, the AMS programs are $7 apiece. For information or to order, write Harrison at 5705 40th Place, Hyattsville, MD 20781.

The Ins and Outs of Instances

BY BRUCE HARRISON

Many moons ago, we generated some programs to deal with the problems of drawing Bit-Map images and printing them. These products include our own Drawing Program, and our TIAPRINT, for printing TI-Artist picture files. Both of those are now available in versions for both 9-pin and 24-pin printers.

As happens, we had some correspondence from our good friend Charles Kirkwood Jr. He ran into the need to deal with printing TI-Artist Instances.

At first, our reaction was not very positive. True, we'd made a provision in our Drawing Program to bring in TI-Artist Instance files and incorporate them into drawings, but otherwise hadn't done anything with them. What Charles was seeking was a way to use Instances as a "letterhead" for his correspondence. That seemed a reasonable thing to do, so when some free time was available, we sat down at our faithful TI and started writing source code.

The Formatter Problem

Others in the TI community have devised methods for printing Instances as part of a TI-Writer document, through the Formatter. This creates problems we don't need, mainly because the Formatter uses "printable" characters as control codes. Thus, if the binary content of a byte in the Instance happens to equal the "@" symbol, the Formatter will translate that into an escape sequence intended to put the printer into double-strike mode.

That's definitely not a good thing to have happen in the middle of a picture. Also, the Formatter cannot gracefully handle strings of characters longer than 80, and we thought that this too would be a killing limitation.

The Standalone Solution

Our idea, then, was to keep the Formatter out of the loop when printing an Instance as a letterhead. We devised a method that takes an Instance file, automatically centers it on the paper, re-maps the bits into the correct bit graphics form, and prints it as single-density graphics.

Using this simplest approach meant that, among other things, our program would be compatible with a broad range of 9-pin dot matrix printers. That includes most Epson, Star Micronics, and Panasonic models in use by the TI community. Since our stand-alone program opens the file to the printer as a D/V 254 file, and with the .CR option, we have no problem with sending the bytes we need without suffering any unwanted carriage returns or line feeds. Thus one can take any instance file and send it to the printer without fear.

The Drawbacks

There is one major drawback. After printing the Instance, you must turn off the printer and roll the paper back to the start of the sheet so that whatever is

THE ART OF ASSEMBLY PART 68

printing your text won't get confused about where pages begin and end. Also, of course, the text file must begin with some number of blank lines or carriage returns so that the letterhead won't get overprinted by the text.

We felt, though, that these two things would be fairly easy to do, so we've gone ahead and offered the product called PRNINST to the TI public as public domain software. The disk also includes versions for double-density bit graphics and for 24-pin printers, as PRNINST2 and PRNINST24.

But What About Editing?

We've never invested in a copy of TI-Artist. Most of the time we don't need it anyway, so why bother? Then we started playing around with Instances.

Back some time ago, Dr. Charles Good sent us about six disks full of very nice Instances from Lima's public domain library. We started printing these out as part of the testing for our PRNINST product. In doing so, we discovered that some of these had stray "artifacts" in them, and that they needed some editing to correct these small flaws. Without TI-Artist, we had no way of visually editing these Instance files. In some cases, the flaws were in obvious enough places that we could edit them using the Editor/Assembler Editor, but those were the exceptions.

What we needed was some method to see what we were doing to an instance directly on the screen, then save it back to disk in Instance (D/V 80) format.

Back to The Drawing Board

Or, rather, the source code from our Drawing program. We figured that with a few changes here and there, we could adapt the Drawing program into an editor for Instance files. It took a few days and nights of hard work, but we were able to produce an easy to use editor that would take care of instances so long as they were 22 characters or less in height, and 32 characters or less in width. Since most instances are in that category, we feel our effort was worthwhile. Besides allowing the direct input of Instance files, we kept the ability to load drawings in either our own format or in the TI-Artist Picture file format. That way, we could extract some neat part of an existing picture into an Instance that could then be used by itself or incorporated into some other picture.

In testing, we of course found many small bugs, and have corrected them, so the INSTED product was made available through the usual channels. This program, like the Drawing program that spawned it, is very versatile. One can, for example, load in font files, either of the CHARA1 type or of the TI-Artist type, and use those to type things on the instances. We mentioned the ability to bring in pictures, and that turned out to be very useful. Some of Charles Kirkwood's pictures were converted "in toto" from picture files into instances. We can also use this program to make new instances from scratch.

The Saving Process

Saving Bit-Mapped imagery as instances is a non-trivial matter, as you might

Continued on page 14

THE ART OF ASSEMBLY PART 68

Continued from page 13

guess. In the first place, we don't want the saved Instance to become 22 rows high by 32 columns wide. Thus we need to scan the contents of the Bit-Map pattern table to determine the required height and width of the instance. To do this quickly, we dump the entire pattern table, starting from Row 2 of the screen, and extending to row 23, into CPU memory. That's actually done when you come out of the editing mode, so that we can put the program's menu on the screen without losing the picture. Thus while we're looking at the menu, the pattern table is stashed away as >1600 bytes in the high memory. The save process scans through that memory, looking first from what would be the top of the picture, until it finds a byte with non-zero content. The row where a non-zero byte is found gets stashed away as the start row for saving. Now the program scans again, this time starting from the bottom of the stashed pattern table, working upwards until it finds a non-zero byte. This tells us the height of the instance.

This process of scanning is then repeated going sideways to find the leftmost column of image content, then its width. Now we have all the needed parameters of the image content to start actually saving the Instance file.

Structure of an Instance

Except for the first record in the file, which contains the width and height of the instance in character rows and columns, the content of the file is in groups of eight numbers, so that each record contains the numbers for one character definition. These numbers are written into the file in human-readable form, so that the file is editable (in theory) with the E/A editor as with any D/V 80 file. Thus a record in the file is eight numbers, separated by commas. Such a record looks like this:

0,0,126,67,0,14,253,85

Since each number represents a byte, the range of these numbers goes from 0 through 255. Knowing this, we're ready to save our Instance to disk.

The first order of business is to write that first record, which contains the width and height of the instance to follow. To do that, we take the width first, convert it from a number to a string, and write that string into a buffer in VDP. For this, we use an "undocumented" GPLLNK feature, given to us by Merle Vogt. Since these are all single byte numbers, we clear the byte at >835E and put the byte to be converted at >835F. We use the Warren/Miller GPLLNK routine, followed by the DATA >2F7C. That undocumented feature provides us with the number in the form of a string, but without the leading space that's usually reserved for a plus sign in the "normal" convert number to string routine. That's a fortunate circumstance, since that leading space is neither desired nor required.

After the width, we write a comma to the VDP Buffer, then the string form of the height. Now we're ready to send the first (actually 0th) record to the file.

THE ART OF ASSEMBLY PART 68

Somewhere in here, of course, we had to open the file as D/V 80, for output. The first record gets written. Meanwhile, we've stashed away the product of height and width, as that tells us how many total records (after the 0th one) we need to write to this file.

Next, we go back into the stashed pattern table, and take the bytes starting at the top row, left column. These get translated to strings and then sent out in groups of eight, separated by commas as shown above. We continue this for WIDTH repeats, then move down one row from our start point and do another set of WIDTH character definitions. When we've done HEIGHT times WIDTH records in this fashion, we're done, so we close the file and return to the main menu. All of this takes time, particularly if we're writing to a floppy disk. Thus we give the user a "PLEASE WAIT PATIENTLY" message on the screen while saving the Instance.

A Small Caution

For reasons we've not been able to determine, some of the Instances we've gotten from Charles Good come with one or more blank rows at their beginning and/or ending. We don't know why they were made that way, as extra rows of all zeros add nothing to the instance, but just take up space on the disk. If you edit such an instance with our program, those blank rows will be eliminated from the saved instance.

Today's Sidebar

The sidebar source code today is not a complete program, but just a section of the code from the double density bit graphics printing program PRNINST2. This section is the vital ingredient in converting an eight byte character pattern into sixteen bytes of Bit Graphics data for the printer. We thought this section, incomplete as it is, might prove helpful to those trying to work with this kind of transition. This works by shifting registers, and inserting a bit into the output byte only if there's a "carry" on the input byte. In other words, if the most significant bit of the input byte was a 1 before the shift, then a 1 gets placed in the least significant bit of the byte being prepared for output. It takes eight passes through all 8 bytes of one character pattern to prepare one 16 byte section of the output to the printer. Each such output section makes 16/120ths (or 8/60ths) of an inch character on the paper. We should point out that on the 9-pin printers the aspect ratio is incorrect, so circles on your instance will be slightly "ovaled" on the paper. They'll be a bit wider than their height. On 24-pin printers, the aspect ratio will be correct, since the special line feeds are in 60ths of an inch instead of in 72nds.

Just recently, we purchased a Canon Bubble Jet printer, and the PRNINST stuff works beautifully with that machine. Oddly, it makes no difference whether we use the single or double density versions, as the print quality is determined by the mode of the printer. In its HQ mode, either the single or

Continued on page 16

THE ART OF ASSEMBLY PART 68

Continued from page 15

double density graphics look superb, while in the printer's HS mode, the graphics look "thinned". For normal 9 or 24 pin printers, we'd recommend using the double density graphics (PRNINST2 or PRNINST24), as these produce much better looking renditions of the Instance.

Next month, among other things, we'll discuss some things outside the realm of Assembly programming. See you then.

SIDEBAR 68

```
* SIDEBAR 68
* RE-MAPPING FRAGMENT
* WE ENTER WHERE A RECORD HAS JUST BEEN READ
* THE STRINGS IN THE RECORD MUST BE CONVERTED TO
* NUMBERS, WHICH ARE STASHED IN INPBUF
*
RNREC1 LI   R0,INPPAB+5  LENGTH OF RECORD
       BLWP @VSBR        READ THAT
       MOVB R1,R2        MOVE TO R2
       SRL  R2,8         RT. JUST
       CLR  R1           R1=0
       LI   R9,INPBUF    POINT TO CPU INPUT BUFFER
       LI   R0,IBUFF     AND TO VDP INPUT BUFFER
       A    R2,R0        ADD LENGTH TO ADDR
       MOV  R0,R7        COPY ADDR TO R7
       BLWP @VSBW        WRITE A 0 AFTER RECORD
       S    R2,R0        BACK TO START OF RECORD
GNNUM  BL   @CONVN       GET A NUMBER
*
* SUBROUTINE CONVN USES XMLLNK TO CONVERT THE STRING IN
* THE VDP RAM BUFFER INTO A NUMBER AT FAC, THEN USES
* XMLLNK TO CONVERT FLOATING POINT TO AN
* INTEGER AT FAC.  WE TAKE ONLY THE LOW ORDER BYTE
*
       MOVB @FAC+1,*R9+  PUT INTO CPU INPUT BUFFER
SKCOM1 BLWP @VSBR        READ A BYTE
       INC  R0           POINT AHEAD ONE
       CB   R1,@COMMA    IS THAT A COMMA?
       JEQ  GNNUM        IF SO, JUMP BACK
       C    R0,R7        CHECK END OF RECORD
       JLT  SKCOM1       IF LESS, REPEAT
*
```


THE ART OF ASSEMBLY PART 68

```
* WHEN WE GET HERE, INPBUF WILL CONTAIN THE
* EIGHT BYTES REPRESENTING THE NUMBER STRINGS
* IN ONE INPUT RECORD
*
* CORE PROCESSING SECTION
* RE-MAPPING OF ONE CHARACTER DEFINITION
* FOR BIT GRAPHICS PRINTING
* THIS TAKES THE 8 BYTES THAT DEFINE
* A CHARACTER AND RE-MAPS THEM INTO
* 8 PAIRS OF BIT-GRAPHICS BYTES FOR THE PRINTER
* ON EACH PASS THROUGH INNER LOOP, THIS
* TAKES THE MSB OF EACH OF THE 8 BYTES
* IN INPBUF, PUTS THAT INTO R1, WHICH GETS
* SHIFTED LEFT BY ONE BIT ON EACH PASS
* THE INPUT BYTE ALSO GETS SHIFTED LEFT ONE BYTE,
* THEN PUT BACK IN INPBUF.
* WHEN INNER FINISHES, R1 HAS ONE BIT FROM EACH INPUT BYTE,
* EACH INPUT BYTE IS SHIFTED LEFT BY ONE BIT.
* REPEATING ALL THIS 8 TIMES
* CREATES 16 BYTES IN OUTBUF
* WHICH REPRESENT CONTENT OF THE CHARACTER
* RE-MAPPED INTO PRINTER BIT GRAPHICS FORMAT
* SUITABLE FOR DOUBLE DENSITY GRAPHICS MODE
*
       LI   R4,8         8 BYTES TO DO
       LI   R10,OUTBUF   R10 TO OUTPUT CPU BUFFER
MAPLOP LI   R5,8         8 BITS TO MAP
       LI   R9,INPBUF    R9 POINTS TO INPUT
       CLR  R1           R1=0
INNER  MOVB *R9,R3       GET AN INPUT BYTE
       SLA  R1,1         SHIFT R1 LEFT 1 BIT
       SLA  R3,1         SHIFT INPUT BYTE LEFT 1 BIT
       JNC  NOCAR        IF NO CARRY, LEFT BIT WAS 0
       ORI  R1,1         IF A ONE, PLACE 1 IN LOW BIT OF R1
NOCAR  MOVB R3,*R9+      PUT BACK INPUT BYTE, INC POINTER
* AT THIS POINT THE INPUT BYTE, SHIFTED LEFT BY ONE BIT,
* HAS BEEN PUT BACK INTO THE INPUT BUFFER LOCATION
* AND R9 POINTS AT THE NEXT BYTE OF INPBUF
       DEC  R5           DEC BIT COUNT
       JNE  INNER        IF NOT DONE, REPEAT
       SWPB R1           SWAP SO RESULT IS IN LEFT BYTE R1
```

Continued on page 18

THE ART OF ASSEMBLY — PART 68

Continued from page 17

```
        MOVB R1,*R10+     PLACE IN OUTPUT
        MOVB R1,*R10+     REPLICATE IN NEXT BYTE
        DEC  R4           DEC BYTE COUNT
        JNE  MAPLOP       IF NOT DONE, REPEAT
        LI   R0,OBUFF     POINT AT OUTPUT VDP BUFFER
        LI   R1,OUTBUF    AND CPU OUTPUT BUFFER
        LI   R2,16        16 BYTES (ONE CHARACTER)
        BLWP @VMBW        WRITE TO BUFFER
        BL   @WRTREC      SEND 16 BYTES TO PRINTER
*
* THIS PROCESS CONTINUES FOR ALL THE RECORDS
* COMPRISING ONE ROW OF THE INSTANCE, THEN IF MORE
* RECORDS REMAIN, IT GOES BACK TO AN EARLIER POINT,
* SENDS THE CONTROL CODES AND MARGIN BYTES, THEN
* STARTS SENDING THE NEXT ROW OF THE INSTANCE.
```

NEWSBYTES

CaDD to include SOB with new release

The next release of PC99 will include the OPA ROMs and GROMs from the Son of a Board by Oasis Pensive Abacutors, according to CaDD Electronics.

Mike Wright of CaDD says Gary Bowser of OPA gave permission for use of the ROMs and GROMs in mid-December.

"Since 'ROMs' and 'GROMs' are just files in PC99, it is very easy to switch between them — no hardware mods are needed! We recently patched the 99/4A ROM interrupt routine to include Jeff White's fixes. It is that easy," Wright says.

According to CaDD, the OPA ROMs and GROMs gives the OPA title screen and then access to its built-in disk manager and other features. CaDD will supply the OPA documentation with PC99.

The OPA files are released as freeware. There is a suggested donation if you use them with PC99.

The SOB menu allows users to catalog drives and run software. It provides a menu for selecting from multiple GROM cartridges. The SOB originally sold for about $50.

CPUG folds

The Central Pennsylvania Users Group in Harrisburg, Pennsylvania, disbanded as of its December meeting, according to its December newsletter, announced as being its last.

HARDWARE PROJECT

Using BASIC and a computer to control Stamps

BY GEOFF TROTT

One of the uses that a computer can be put to is that of controlling a system. For example, Ross Mudie has used one of his TI99/4As to control a model train set. Many other members are interested in using a computer as the basis for things like a home security system. Uses like these can make good use of an 'old' computer that is not used any more.

The main difficulty in doing any of these simple tasks is to get the signals into and out of the computer in some simple way. Ross has done this by using the cassette port, I think.

When I was asked by Lew Griffiths to help him with his desire to control a train set, I thought that it would be most useful to have an interface that could be used with almost any computer and was not special to the T199/4A. This lead to using the RS232 port as the communication path between the computer and the I/O data. Most computers have an RS232 port as standard, or it can be added as an extra

Using the RS232 Port

These days the RS232 port is most often used for a mouse or for a modem, although sometimes it is used for a printer. The nice thing about the RS232 is that its programming is a standard part of the computer's operating system so it can be accessed in any programming language, such as BASIC.

So I decided to make a device which could be connected to any computer's RS232 port and which would allow digital data to be transferred into and out of a computer.

I managed to interest one of my students, Rodney Stewart, in doing this job as a small project in the final year of his degree. He was given the task to find the cheapest way to make such a simple device and to test it with as many different computers as he could. He came up with a solution which cost about $100 to $200, depending on the cost of making a printed circuit, and which was demonstrated to work with a TI99/4A, an Amiga, a Macintosh, and a PC. The program for each computer was written in a form of BASIC and should be quite easy to modify to suit any application.

The system can handle up to 64 digital inputs and 64 digital outputs. It is modular in that the hardware can be expanded in eight-bit lumps at a time up to the maximum of 64. The inputs and outputs are grouped into eights and then can be accessed as single bits or eight bits at a time. Data are transferred by simple character commands from the computer.

For example, the command "O3.6=1" would set bit 6 of byte 3 to

Continued on page 20

HARDWARE PROJECT

Continued from page 19

a 1. Eight bits can be changed at a time with the command "O3=255", which would set all bits of byte 3 to 1. The response to an output command is "$" to indicate all is well or "#" if an error is detected.

For input, the command "I2" would cause a response of "$ddd," where "ddd" is the decimal value of the bits in byte 2, while "I2.3" would cause a response of "$b," where "b" is either 1 or 0 depending on the value of bit 3 in byte 2. In all cases, input and output, the "$" as the first character in the message indicates the command is successful while a "#" as the first character in the message indicates an error.

Testing The System

An Extended BASIC program for the T199/4A to test the system follows. (The program won't do anything unless you actually have input coming from the RS232 port.—Ed.)

```
100 OPEN #1:"RS232.EC.BA=960
0.PA=N.DA=8",OUTPUT !247
110 CALL KEY(0,K,S)!187
120 IF S0 THEN 150 !094
130 PRINT #1:"%" !053
140 GOTO 170 !249
150 LINPUT A$ !015
160 PRINT #1:A$ !173
170 OPEN #2:"RS232.EC.BA=960
0.PA=N.DA=8",INPUT !147
180 INPUT #2:X$ !187
190 CLOSE #2 !152
200 COMM$=SEG$(X$,1,1)!041
210 LENGTH=LEN(X$)-1 !250
220 IF COMM$="*" THEN 270 !0
12
230 IF COMM$="!" THEN 340 !0
73
240 IF COMM$="$" THEN 290 !0
26
250 IF COMM$="#" THEN PRINT
"ERROR" :: GOTO 110 !094
260 GOTO 110 !189
270 PRINT "STAMP OK" !170
280 GOTO 110 !189
290 IF LENGTH=0 THEN PRINT "
OUTPUT SUCCESSFUL" :: GOTO 1
10 !113
300 NO$=SEG$(X$,2,LENGTH)!09
9
310 NO=VAL(NO$)!099
320 PRINT NO !057
330 GOTO 110 !189
340 PORT$=SEG$(X$,2,1)!067
350 PORT=VAL(PORT$)!179
360 PRINT "PORT NUMBER: ";PO
RT !241
370 FOR I=3 TO LENGTH !002
380 IF SEG$(X$,I,1)="." THEN
 430 !202
390 NO$=SEG$(X$,I,1)!233
400 NO=VAL(NO$)!099
410 PRINT NO;!237
420 IF SEG$(X$,I+1,1)="." OR
 I=LENGTH THEN PRINT !044
430 PORT$=SEG$(X$,I+1,1)!076
440 PORT=VAL(PORT$)!179
450 I=I+1 !011
460 PRINT "PORT NUMBER: ",PO
RT !240
470 NEXT I !223
480 GOTO 110 !189
```

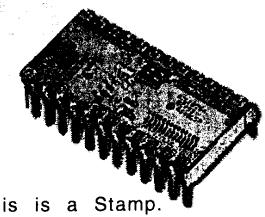
The RS232 is a bit interesting for the TI99/4A, so that it must be opened twice, once for output and once for input. Line 100 opens it for output and this remains open while the program runs. Line 170 opens it for input to read one line and then closes it in line 190. This is because the RS232 does not like half duplex connections.

The program waits in a loop for a key to be pressed (110). Until a key is pressed, the system is polled (a "%" character sent) and the response to the poll is input into X$ (180). When a key is pressed, the input command is put in a string A$ (150) and then sent to the system. The response to the command is read into X$ (180). The first character of the response is put in COMM$ and the program performs different functions depending on that character.

Receipt of "#" means an error has occurred. Receipt of "$" means success and, if the command was an output, there are no more characters. For an input command, the value is extracted from the message after the "$", converted to a number, and printed on the screen.

Receipt of "*" means no inputs have changed while receipt of "!" means that some inputs have changed and the new values follow the "!" character with the port number and bit numbers. These last two characters are the possible responses to a poll and can be thought of as a type of interrupt, where changes in input values can be notified to the computer without reading all the inputs to find out whether any have changed.

This simple program shows how the data is handled by the program in the computer and the system. The system itself is a microcontroller called a "Stamp 2." This is a single chip microcontroller based on a PICC microcontroller, but with a BASIC interpreter in ROM. The BASIC interpreter allows the Stamp to be programmed in its own version of BASIC, which leads to fast program development. The Stamp program is developed on a PC using another program, which does all the checking for errors and then allows the code to be downloaded to the Stamp for execution.



This is a Stamp. Stamps have a variety of uses and turn up frequently in amateur robotic experiments.

Joy changes address

Joy Electronics Inc., a dealer in TI products, has changed its mailing address to 11258 Goodnight Lane, Ste. 110, Dallas, TX 75229-3395.

Dealing with memory requirements

BY VERN JENSEN

As you become more experienced with c99 and start writing larger programs, you will discover that it is important to keep your code as compact as possible. This is because both the code for your program and all of your variables must load into the TI's memory at run time. The longer your code, the less room you have for variables, and vice-versa. So writing compact code would be especially important for applications that need a lot of data space, such as a word processor.

However, it is important for all applications, since if you don't watch out, your program will eventually become too large to load. This happened when I was developing Virus Attack. I added so much to the game that it wouldn't fit into the TI's memory until Bruce Harrison wrote a special routine that would load it into both high and low memory.

Simple Tricks

While we have the luxury of writing in a simple, fast language, it does have its drawbacks. One is that a single c99 statement may be compiled into many assembly language statements, making it difficult to write compact code. However, by keeping this in mind, there are some tricks you can do to make your program shorter. For one, putting repetitive statements in loops can help immensely.

Consider the task of setting up the graphics for a game. In Extended BASIC you would typically make repeated CALL CHAR statements. However, in c99, function calls are probably one of the most "expensive" statements in terms of the amount of assembly code that is needed for each c99 statement. Instead of making repeated ChrDef calls, if we place the data for each group of characters in an array, we can loop through the array and only have to write a single ChrDef call, which results in fewer assembly statements than there would be if we made several separate ChrDef calls.

However, we now run into another problem; to my knowledge, c99 provides no way to initialize character arrays with strings longer than one line, thus limiting your array data to 80 characters. However, with a little trickery, we can get around this limitation. We can create one big array by placing several small arrays in a row, one right after the other. Then we can access the first one as if it were as big as the sum total off all the arrays combined. When we get outside the bounds of the first array, we drop into the second one.

One thing to keep in mind though is that if you initialize your arrays with character string constants, as I do below, then each array will be appended with

a 0 to mark the end of the string. This means than an array with 64 characters will actually take up 65 bytes.

In addition, if you take a look at the assembled output of these lines after they are compiled, you will notice each array is ended with the assembly language EVEN statement. I must confess my complete assembly language ignorance here. However, my guess is that the EVEN statement aligns the next block of data so that it starts on an even memory address. And since 65 is not an even number, the EVEN statement will place the next block of data 1 byte beyond the end of the previous block of data. This means that if you have a 64 character array like the one below, and you want to start reading from the next array, you should read from element 66. In other words, cData[66] below is equivalent to cData2[0]. The program below actually compiles. Try it out if you want.

```
#include "DSK2.GRF1;H"

char chrNum[] = {65,69,73,77,0};

char cData[] =
  "00000000000000001111111111111112222222222222223333333333333333";
char cData2[] =
  "44444444444444445555555555555556666666666666667777777777777777";
char cData3[] =
  "88888888888888889999999999999999AAAAAAAAAAAAAAAABBBBBBBBBBBBBBBB";
char cData4[] =
  "CCCCCCCCCCCCCCCCDDDDDDDDDDDDDDDDEEEEEEEEEEEEEEEEFFFFFFFFFFFFFFFF";

main()
{
  int n;
  Grf1();

  for (n=0; n <= 15; n++)
    VChar(5,n+10,65+n,15);

  InitChar();

  while (1);
}

InitChar()
{
  int c;
  c = 0;
```

Continued on page 24

Continued from page 23

```
  while (chrNum[c] != 0)
  {
    ChrDef(chrNum[c],&cData[n*66]);
    c++;
  }
}
```

When InitChar is called, the code loops through all elements of the chrNum array until it hits 0. The chrNum array is used to specify the character set for each block of data. Next, the address of the appropriate section of the cData array is passed to the ChrDef function. When the element number exceeds the bounds of cData, it spills into cData2, cData3, and cData4.

Normally you don't have to use tricks like this in standard C, since standard C provides a way to break long strings up between several lines, but since we don't have this capability in c99, we must make do with what we have. Of course, if you don't have room for any Char data at all in your program, you could move it to a separate file, then have your program open that file and reads its contents when starting up. However, this technique generally isn't too popular, since it adds an additional delay to the program's loading time, even if it saves a little memory. I imagine most users should find the technique described above more than adequate for keeping your InitChar function as compact as possible.

If you've never done it before, you should compile a c99 program sometime with the "Include C Source" option turned on. Then open up the resulting assembly language output file with the editor and take a look at the number of assembly language statements necessary for each c99 statement. (The program above would be a perfect program to do this with.) Knowing how much each c99 statement "costs" in terms of program length will help you when writing your program, since you will know how to write code in the most efficient way possible. Typically, simple variable assignments, if statements, and arithmetic operations require fewer assembly language statements than the "bigger" c99 statements, such as function calls or complicated if statements.

Bits and Bytes

Writing more compact code isn't the only way to make a program take up less memory. If you need to keep track of lots of "on/off" choices, such as the preferences settings a user has selected, you can change the individual bits of a number, where each bit corresponds to one of your choices, rather than devoting an entire variable to each choice.

For example, a while back I was considering working on a c99 game where the player would walk around in a maze of rooms, each room filling the screen. The rooms would contain various objects that the player could pick up. When

the player leaves a room and comes back, the objects he picked up should not reappear.

I wanted the game to contain around 50 or so rooms, and I wanted each room to be able to contain at least 10 objects. If I were to create a char array with one element for each object, each element specifying whether the corresponding object had been picked up, my array would take up 500 bytes! This would obviously leave me less room for actual program code, since the data for the objects alone takes up a good chunk of memory.

A better way to do it would be to dedicate a single int variable to each room, and set the individual bits of that variable to indicate whether the corresponding object has been picked up. A 0 bit would mean it hasn't been picked up yet, while a 1 bit would indicate that it has. And since an int has 16 bits, we could have 16 objects per room, while only requiring a total of 100 bytes (2 bytes per room * 50 rooms).

When each room is loaded from disk, my program would scan through the room and store the position of each object in a two-dimensional array with 16 elements in the first dimension (one element for each object), and 2 elements in the second dimension. The second dimension of the array would be used to store each object's row and column on the screen. So the code for scanning a level might look something like this:

```
char objArray[16][2];

ScanLevel()
{
  int row, col, char, objNum;
  objNum = 0;

  for (row=1; row <= 24; row++)
  {
    for (col=2; col <= 30; col++)
    {
      char = GChar(row,col);
      if (char == kObj1 || char == kObj2 ||
          char == kObj3 || char == kObj4)
      {
        objArray[objNum][0] = row;
        objArray[objNum][1] = col;
        objNum++;
      }
    }
  }
}
```

Once an object is picked up, this array would be scanned to find the array

Continued on page 26

BEGINNING C99 PART 8

Continued from page 25

element with the matching row and column. This array element is the object's "ID". This method ensures that each object on the screen gets its own unique ID, and these IDs will be the same each time the level is loaded, since the room is always scanned in the same order. Once we know an object's ID, all we have to do to mark that object as being removed is to set the corresponding bit in the variable that keeps track of objects for the current level. That code might look something like this:

```
  /* Below is an array of 50 integers, one for each room, */
  /* for keeping track of which objects have been picked up. */
  /* You'd want to set each element to 0 before starting the
game. */
int ObjData[50];

StoreObject(room, row, col)
int room, row, col;
{
  char objID;
  objID = 0;

    /* Scan objArray to find the ID for this object */
  for (objID=0; objID <= 15; objID++)
  {
    if (objArray[objID][0] == row &&
        objArray[objID][1] == col)
      break;
  }

    /* Mark this object as having been removed. */
  SetBit(&ObjData[room], objID, 1);
}
```

The SetBit function is covered later in this article. You simply pass it the address of an integer, the position of the bit you wish to change (0-15), and whether you want the bit changed to a 1 or a 0.

To keep objects from reappearing when the player reenters a room, we must first build the array that keeps track of the position of all objects in the room (the objArray), then we must look at each bit in the variable that keeps track of what objects in the room have been picked up. Naturally, each object that hasn't been picked up shouldn't be there, so we erase it with a call to the HChar statement:

```
EraseObjects()
{
  int row, col, char, objID, r, c;
  objID = 0;
```


BEGINNING C99 PART 8

```
    /* Scan entire room for objects */
  for (row=1; row <= 24; row++)
  {
    for (col=2; col <= 30; col++)
    {
      char = GChar(row,col);

        /* Did we find an object? */
      if (char == kObj1 || char == kObj2 ||
          char == kObj3 || char == kObj4)
      {
          /* Determine the object ID for this object */
        for (objID=0; objID <= 15; objID++)
        {
          if (objArray[objID][0] == row &&
              objArray[objID][1] == col)
            break;
        }

          /* Has this object been picked up? */
        if (GetBit(ObjData[room],objID) == 1)
        {
            /* If so, erase it! */
          r = objArray[objID][0];
          c = objArray[objID][1];
          HChar(r,c,32,1);
        }
      }
    }
  }
}
```

Once again, the GetBit() function is covered later in this article. It simply returns the value of a single bit from an integer. You pass the integer to the function, and the position of the bit you wish to read (0-15). Don't spend a lot of time trying to understand the example functions above. If they don't make sense, it doesn't matter - the important part is what is coming below. The code above is provided simply to demonstrate one situation where you might want to set the individual bits of a number, rather than dedicating an entire variable to each on/off choice. I should also warn you that the functions above are not tested. They should work, although it's possible I may have made a mistake when writing the code. (Nobody's perfect.)

A Little Background

As you probably know, variables are stored in the computer as a series of

Continued on page 28

BEGINNING C99 PART 8

Continued from page 27

bits, which are simply "on/off" switches with a value of either 0 or 1. A Char variable takes up a Byte, which is a term meaning 8 bits. An int takes up 2 bytes, so it contains 16 bits. In Base-2, or Binary format, the right-most bit of a number is the one's place, the bit to the left of that is the two's place, the next is worth 4, the next 8, then 16, 32, 64, 128, 256, and so on. So this is what the number 11 would look like in Binary format:

```
00001011
```

Look at which bits are on and which are off. The bits in the 1, 2, and 8 places are turned on. So 1+2+8 = 11. Here's what a Char variable would look like with all its bits turned on:

```
11111111
```

That would be 1+2+4+8+16+32+64+128, or 255, the maximum value of a Char variable. Here's another example, with the binary equivalent of 76 (64+8+4):

```
01001100
```

Binary Operators

The C language provides several operators for changing a number at the bit level. The simplest of these is the shift operator, which we will look at first. There are actually two shift operators, "<<" and ">>", which shift the bits of a number left or right a specified number of bits, filling vacated bits with 0. Here's an example that shifts the bits stored in x left two positions. The result is then copied back into x:

```
x = x << 2;
```

If x had been 00101110 (46), it would become 10111000 (184). Or if x had been 01001010 (74), it would become 00101000 (40). Notice that we lose the 1 bit on the far left because it is shifted right out of the number. Shifting a number left 2 places is the same as multiplying the number by 4. (4x46 = 184. And 4x74 = 296, although as that number exceeds the bounds of a char variable, the result is 296-256, or 40.) Shifting a number left by only 1 is the same as multiplying that number by 2.

Right shifting a number is similar to left shifting. Here's an example that shifts the bits in x 4 places to the right:

```
x = x >> 4;
```

That would turn 11110000 into 00001111, and 10101011 into 00001010. Next we come to the "one's complement" operator ~ (FCTN-W on the keyboard), which converts each 0 bit into a 1 bit and vice versa. So if n below is 00100111, it would be turned into 11011000:

```
n = ~n;
```

You can also use this operator in combination with other statements. Here's an example that shifts the bits in b right c positions, then inverts the bits, storing the result in a. It should be noted that b and c are not changed by this operation;

BEGINNING C99 PART 8

a is the only variable that is changed.

```
a = ~(b>>c);
```

Next we come to the bitwise AND operator "&", not to be confused with the logical AND operator commonly used in IF statements ("&&"). The bitwise AND operator compares two numbers, setting each destination bit to 1 only if the corresponding bit of both numbers is also 1. For example, the statement "n = a&b" would set n to the result of the following operation, where a and b are the top two numbers:

```
10111001 (a)
00011101 (b)
--------
00011001 (n)
```

Here you can clearly see how each bit in the result is turned on only if the corresponding bit in both a and b is on. The bitwise AND operator is usually used to mask off a certain set of bits. For instance, the statement "n = a&7" sets n to the first three right-most bits of a. (Remember that 7 is equal to 00000111 in binary.) That is, if a was 11010110, then n becomes 00000110.

You can use the & and >> operators together to determine whether a specific bit of a number is on or off. Here's a function that returns whether the specified bit is on or off:

```
GetBit(theNum, theBit)
int theNum;
char theBit;
{
  if (theBit == 0)
    return myNum & 1;
  else
    return (theNum >> theBit) & 1;
}
```

To call this function, simply pass an integer and the position of the bit you would like to read. Keep in mind that since integers can be both positive and negative, the leftmost bit is used to specify whether the number is negative or positive. Passing a value of 0 as the theBit parameter will read the right-most bit (the 1's place), a value of 1 will read the bit to the left of that, and so on. Since there are 16 bits in an int, you can pass any value between 0 and 15 as the theBit parameter.

Now lets take a look at how the functions works. Lets say you call the function with an integer of 99 and a theBit parameter of 5. (Remember that since the first bit position is 0, a value of 5 will read the 6th bit.) A value of 99 is the following in Binary (64+32+2+1):

```
0000000001100011
```

As you can see, the 6th bit from the right is a 1, so the function should return

Continued on page 30

BEGINNING C99 PART 8

Continued from page 29

true. But first, let's take a look at how the function works. First the bits in theNum are shifted right by the value in the theBit parameter, resulting in this number:

```
0000000000000011
```

Next, this number is used with the AND operation to get the right-most bit:

```
0000000000000011
0000000000000001
----------------
0000000000000001
```

The resulting number is returned to the user, which in this case is a 1. If the theBit parameter had been a 4 instead of a 5, a 0 would have been returned. You may have noticed that a special case is added for when the theBit parameter is a 0. This is because for some reason, the shift operators don't work correctly on the TI when used with a value of 0. Normally, shifting a number right or left 0 bit positions should do absolutely nothing, but attempting to do this on the TI results in code that doesn't work right. By adding the special case to the GetBit function, we avoid this problem.

The OR Operators

By now you may be wondering how one would write a function to set a particular bit. This leads us to the OR operators. The first version, the bitwise OR operator |, turns the destination bit on when the corresponding bit in either or both of the source numbers is on. Here's an example:

```
char a, b, c;

a = 150;
b = 227;

c = a | b;
```

Again, the bitwise OR operator "|" is not to be confused with the logical OR operator "||" that you use in IF statements. In the operation above, the bits in C are turned on when the same bits in either a or b are on. Here's an illustration:

```
  10010110 (a)
  11100011 (b)
  --------
  11110111 (c)
```

The other OR operator ^, called the exclusive OR operator, turns the destination bit on when the same bit in either a or b is on, but not when both bits are on:

```
c = a ^ b;
10010110 (a)
11100011 (b)
--------
01110101 (c)
```

Below is a function which uses the bitwise OR operator to turn a single bit on or off, leaving the other bits unchanged:

```
SetBit(theNum, theBit, onOff)
int *theNum, onOff;
char theBit;
{
  if (onOff)
  {
    if (theBit == 0)
      *theNum = *theNum | 1;
    else
      *theNum = *theNum | (1 << theBit);
  }
  else
  {
    if (theBit == 0)
      *theNum = *theNum & ~1;
    else
      *theNum = *theNum & ~(1 << theBit);
  }
}
```

To call this function, pass the address of an integer you wish to change, the position of the bit to be modified, and a 1 or a 0. If the bit is to be set to 1, the left shift operator is used to move a 1 into the correct bit position, then the OR operator is used to add that bit to the number. If the bit is to be set to 0, the one's compliment operator is used to get a mask that has all bits turned on except for the bit to be set to 0, and this mask is used with the AND statement to turn that bit in the number off.

Again, remember that when you call this function, the theBit parameter can be any value from 0-15, where 0 indicates the first bit on the right, 1 indicates the second bit from the right, and so on. And just like earlier, we must add a special case for when the theBit parameter is 0, to avoid shifting a number 0 positions.

A Test Program

Together, the left and right shift operators, the AND and OR operators, and the one's complement operator provide everything needed to operate on a number at the bit level. To conclude this article is a program which lets you toggle the individual bits of a number between 1 and 0, and then shows the result in both binary and decimal format.

Before getting started, you'll need to copy the ACCNUM/O file from Bruce Harrison's c99 utilities disk (which comes with my c99 Starter Kit) to your work

Continued on page 32

BEGINNING C99 PART 8

BEGINNING C99 PART 8

Continued from page 31

disk in drive 2. You should also copy the PUTNUM/O file to your work disk if you didn't last issue.

In addition, you'll need to type in the BPHONK assembly code below, saving it in its own file called BPHONK on your work disk. As you may have guessed, it contains two useful routines for generating the standard beep and honk sounds. The BPHONK file was originally included with c99 release 4, but somehow didn't make it into my starter kit. To use the routines just add this line to your program:

```
#include "DSK2.BPHONK"
```

Then you can invoke a beep or a honk simply by calling the function with the appropriate name, such as Beep(), or Honk(). Since the code is compiled along with your program, there is no need to load a separate library at run time.

BPHONK

```
#asm
 REF C$GPLL
BEEP BL @C$GPLL
 DATA >34
 B *13
HONK BL @C$GPLL
 DATA >36
 B *13
#endasm
```

BITTEST/L

```
DSK2.CSUP
DSK2.GRF1
DSK2.ACCNUM/O
DSK2.PUTNUM/O
DSK2.BITTEST/O
```

BITTEST/C

```
/* BITTEST/C - FROM THE JAN/FEB */
/* 1998 ISSUE OF MICROPENDIUM   */

#include "DSK2.BPHONK"
#include "DSK2.GRF1;H"
extern PutNum(), AccNum(), DisStr();

int theBit, oldBit;
int theNum = 0;

main()
```


BEGINNING C99 PART 8

```
{
  int s;

  Grf1();
  Screen(8);

  DisStr(3,6,"B I T   T E S T E R");
  DisStr(8,2,"Modify which bit? (0-15):");
  DisStr(13,2,"Decimal");
  DisStr(17,2,"Binary");

  while (1) /* Endless loop */
  {
      /* Display the number in both */
      /* decimal and binary format. */
    PutNum(13,10,theNum,0);
    PutBinary(17,10,theNum);

    Beep(); /* Part of BPHonk */
    do
    {
        /* Get the bit to be changed */
      theBit = AccNum(8,28,1);

        /* Wait for enter key to be released.    */
        /* (AccNum should do this, but doesn't.) */
      do
      {
        Key(0,&s);
      } while (s);

        /* Honk if invalid number */
      if (theBit < 0 || theBit > 15)
        Honk();
    } while (theBit < 0 || theBit > 15);

      /* Toggle bit between 0 and 1 */
    oldBit = GetBit(theNum,theBit);
    SetBit(&theNum,theBit,!oldBit);
  }
}

PutBinary(myRow, myCol, myNum)
char myRow, myCol;
```

Continued on page 34

BEGINNING C99 PART 8

Continued from page 33

```
int  myNum;
{
  char n;
  int myBit;

    /* Draw each of the 16 bits */
  for (n=0; n < 15; n++)
  {
    myBit = GetBit(myNum, 15-n);
    HChar(myRow, myCol+n, '0' + myBit, 1);
  }
}

GetBit(myNum, myBit)
int  myNum;
char myBit;
{
  if (myBit == 0)
    return myNum & 1;
  else
    return (myNum >> myBit) & 1;
}

SetBit(myNum, myBit, onOff)
int *myNum, onOff;
char myBit;
{
  if (onOff)
  {
    if (myBit == 0)
      *myNum = *myNum | 1;
    else
      *myNum = *myNum | (1 << myBit);
  }
  else
  {
    if (myBit = 0)
      *myNum = *myNum & ~1;
    else
      *myNum = *myNum & ~(1 << myBit);
  }
}
```


HARDWARE PROJECT

Connecting a third drive to your TI

By JIM WIEGAND

The following article has appeared in a number of user group newsletters. — Ed.

Connecting a third disk drive to your TI is relatively easy. A power "Y" connector and the proper data ribbon cable will put you into operation. I used a Radio Shack Tandy Color Computer drive. It has a power supply and a single-sided disk drive. I replaced it with a double-sided half-height drive.

Most disk drive data cables will have a guide inserted into the controller connector between pins 3-4 and 5-6 (pin 1 always has a colored wire on the ribbon cable). The TI disk controller card has an external connector for an additional drive.

This connector has a guide slot between pins 9-10 and 11-12. If this special cable cannot be found, you can cut the guide out of the ribbon connector. Be careful to not damage the contacts. If you make this modification, remember to have the colored wire at the bottom when hooking it up. If you are technically inclined, you can remove the disk controller card and cut a slot at the proper location. Don't cut too deeply if you do this. Watch for printed circuits in the line of the cut.

You can also use a 3.5-inch disk drive. Adapters to allow mounting these in the 5.25-inch drive bay may not be easy to find. I located some in a catalog from MCM Electronics of Centerville, Ohio (800-543-4330). A call will probably get you a catalog. These kits contain the power adapter and a data connector adapter.

Jumper Blocks

A little information about disk drives may be helpful at this point. The TI disk controller uses connector pins 10, 12, and 14 as the drive select (DS) lines to select disk drives 1, 2, and 3, respectively. Most 3.5-inch drives are set up as drive "B" (DSK2). Some of these drives will have DS jumper blocks allowing you to select its drive number. This block usually consists of a row, or maybe two rows, of five or six pins with a jumper across two of them. Look around the sides near the connector end. If no such selector can be found, then the unit is probably hard-wired as drive "B." A data cable modification will be required. Here are directions:

- From the colored wire on the ribbon cable (pin 1), count wires to 12 and 14.
- Cut these wires about an inch from the disk drive connector.
- Solder a wire between pin 14 from the disk controller and pin 12 from the disk drive. These connections should be insulated. You can use tape, tubing, or other suitable material.

The odd-numbered pins are common (grounded) and are all on one side of the connector.

Protection schemes last only so long

The identity of the author of the following article is not known. We found it in M.U.N.C.H., the newsletter of the Massachusetts Users of the Ninety-Nine and Computer Hobbyists.—Ed.

The TI proprietary protection scheme simply wrote an uppercase "P" at a certain byte. UALPHA P is ASCII hex 50. Using a sector editor, simply change the >50 to >20. The >20 is decimal 32, which is the space character. Do this for all of your files. Now they are all unprotected.

I am assuming some familiarity with a sector editor and that you have to write the changes back to disk to make them permanent.

Extended BASIC protection is just about as simple. It usually consists of adding eight to the file type byte in the FDR (file descriptor record). Sometimes you may have to XOR words 2 and 4. So, simply subtract eight from the file type byte and your program should be unprotected.

How do you XOR words >2 and >4? The simplest way is to change both words to binary. Line up the columns. If both columns are different, i.e. 0/1, 1/0, write a 1 beneath those two columns. If the columns are the same, write a 0. Ignore any long line of 0s you may have to the left. Start with the first 1 and continue writing left to right as you would any number. Store the results in bytes 0 and 1.

Perhaps an example will make it better understood. Let us assume that word

H2	=	37C$	=	0011	0111	1100	1110	and that
		37CB	=	0011	0111	1100	1011	word H4.

The result is 0000 0000 0000 0101. 101 is 5 in decimal and hex. So, in bytes 0 and 1, we will write H00 and H05.

The next protection scheme I would like to discuss is almost as simple as writing "P." This type of protection uses a program that can detect bad sectors. It depends on only initializing a certain number of tracks. Let us say you have a disk manager such as CorComp's which will allow you to initialize a certain number of tracks. Let us say you initialize eight tracks. Eight tracks times nine sectors/track equals 72 sectors. The program checks sector 72 and, if it is good, returns you to the title screen.

Why 72? Remember in "computer speak" we almost always start counting at 0. So eight tracks would initialize 0 through 72 for a total of 72. Sector number 72 would be on track nine.

Another method that was widely used until everyone caught on was the spiral sector. In this scheme, not all of the bytes should be written to sector 0 are written to sector 0. Let us say you break this sector up into eight-byte

blocks — 256/8 = 32, so your last eight-byte block will be written to sector 31.

Why not 32? Remember we start counting at 0. This is called a spiral sector because of the pattern formed as it moves inward towards 31. Meanwhile, 31 is moving outward in the same pattern, and so on. Of course, once copier programs were written which could copy spiral sectors, authors and publishers quit using them.

Another scheme which had some success is to zero out sectors 0 and 1. Some authors also zero out the bytes count for each line of an Extended BASIC program, which lets it run but makes it unlistable. Once again when programs were developed that could restore the byte count and sectors 0 and 1 programmers were forced to try other schemes.

Ultimately all these schemes fail because at some point the disk controller card has to be able to read the disk. How many protection schemes are really possible? Since TI decided to follow IBM's lead in this matter, there are exactly 34.

One is to tell a track editor the sectors are IBM-type sectors with 512 bytes per sector. Another is to tell it there are no sectors on this track when the track actually has sectors on it. Another is to misnumber sectors and tracks so that we have sector 2000 on track 400 when it is actually sector one on track zero. Almost anyone can come up with some kind of protection scheme.

The real fun lies in writing a short program that in effect tells the FDC (floppy disk controller) that everything is all right.

Another interesting tidbit — I once examined a protected disk with a track editor and found the names and addresses of three people written in the tracks throughout the disk.

Needless to say, the disk would not boot or, if force-booted, would not run unless the names and addresses were there. Fortunately, I had a track copier that could copy the names, so I was able to make my backup copy.

Ultimately all these schemes fail because at some point the disk controller card has to be able to read the disk. How many protection schemes are really possible? Since TI decided to follow IBM's lead in this matter, there are exactly 34.

Another gem — tell the backup program that the tracks are alternating single- and double-density!

MICROREVIEWS

Sound Generator, Byte Magazine and Scott Foresman page scans, JM Base

BY CHARLES GOOD

A new service to readers of my column.

For several years I have offered to mail anyone for $1 any of the shareware or freeware disks I review in my MICROpendium column. The vast profits from this service have gone into my Florida vacation fund and have finally accumulated enough to pay my way to the edge of town. So, from now on, I am willing to e-mail any of the disks I have ever reviewed for free, as well as send them through the US mail to those who send me $1 for the cost of postage and disk.

To receive free software from me by e-mail you have to have an e-mail account that accepts attached binary files. I can e-mail disks in either of two ways. If you own PC99 I can send the disk in PC99 format which can be directly used on a PC by the recipient without any manipulation.

If you have a double-density disk controller and own the commercial software PC-Transfer, then I can e-mail the disk with a TIFILES header. You will need PC-Transfer to move the TIFILES file you receive onto a 99/4A disk and then you will need Archiver to unpack the files.

PC-Transfer is available for purchase from Ramcharged Computers. PC99 is available from CaDD Electronics.

JM BASE
by John Martin

In a recent column I mentioned that there are people who run businesses using 99/4A computers. JM Base was written in Extended BASIC by such an individual. It is a general data base with a business orientation. You can keep most of the records needed for a small business or you can just use JM Base as a name and address database.

The entire software package requires four disks, the program disk and three data disks. Disks are recognized by volume name so you can put any disk in any drive, and you only need the data disks on line (in a drive) if you are actually going to use them. The data disks are for inventory, books, and mail list.

You can start the software directly or read a short documentation file which then automatically starts the software. After a short introductory screen you get to the main menu, which has the following options:

1. customer list
2. print customer list
3. last file search
4. inventory
5. print inventory
6. books
7. print books
8. print labels
9. phone call list
0. setup.

MICROREVIEWS

On the "Books" disk you can set up 10 years worth of data one month at a time. Each monthly data set includes input for several sources of income, and the following expenses: sales tax, supplies, auto, advertising, misc, shipping, and telephone. There are also some user-definable expense categories. These categories can be edited as needed. Net income for the month is automatically calculated as data is added or changed. State sales tax is automatically calculated. Year-end totals can be calculated.

The "List" disk includes name, address, phone, and fields for up to four user-defined pieces of information about each person in the data base. You can store up to 10 separate lists on the same List disk. The number of individuals in each list is apparently limited only by the capacity of the media. From this data you can print the whole list as a tabular report on 8.5 x 11 paper, or mailing labels onto fanfold 1.5 x 3-inch labels. You can also print a list of telephone numbers.

The "Inventory" disk lets you keep up to 10 inventory lists on the same disk. Each data block in each list lets you enter part number, description, quantity, wholesale cost, and retail cost. Your cost ("quantity" times "wholesale cost") is then calculated and stored on disk.

JM Base is designed to keep data for a small business on a minimal 99/4A disk system. You don't need hard drives or RAMdisks. Everything runs off of floppy drives and the various floppy disks can be in any drive. You do need at least two DSSD drives, since the program disk must always be on line, and you must put a data disk in another drive. Four drives work best and will give you seamless access to all your data without manually switching disks. The software will work on RAMdisks if your RAMdisks can be recognized by volume name. Horizon RAMdisks at an address other than >1000 may not let you do this.

JM Base comes on a DSSD disk which includes a short documentation file. The title screen says that the program is not freeware. But it is. I was given the disk by Rich Gilbertson, who told me that the author John Martin is now allowing the software to be freely distributed.

SOUND GENERATOR
by Walid Maalouli

This is a really fun Extended BASIC program that helps you to compose sounds and music in the Extended BASIC environment. No longer do you have to repeatedly type long CALL SOUND statements in command mode to discover what a particular combination of three sound channels and the noise generator will sound like. Sound Generator gives you a nice screen display showing each of the parameters available for a single CALL SOUND statement. You can modify any of the parameters, then play the resulting sound, then modify another parameter and play the sound again

Continued on page 40

Continued from page 39

as many times as you want.

For each of the three channels that can be accessed in a single CALL SOUND statement you can alter the frequency. For the noise generator you can alter the type of noise (noise types 1-8), and for both the three channels and the noise generator you can independently set the volume. You can also set the duration to any value for the combination of sounds. You get a screen display of all the numbers, which you can increase or decrease in single digits or in multiples of 50 using keyboard input. You can play the resulting sound once, or have it play repeatedly until you tell the computer to stop playing. Once you get a sound you like you can send the resulting values to a printer.

One of the most unusual aspects of Sound Generator is that you can use the Mechatronics mouse instead of keyboard input to increase or decrease the values of the various sound parameters. Other than Monty Schmidt's Command DOS, which was released many years ago, I know of no other software that uses the Mechatronics mouse. I don't have such a mouse, so I didn't test this feature.

I really enjoyed playing around with Sound Generator, seeing what slight adjustments in this or that parameter would do to the resulting sound. It is particularly interesting to listen to the effect of the noise generator on sound quality. Please understand that Sound Generator only works with the equivalent of one CALL SOUND statement at a time. From within Sound Generator you cannot write music by combining several successive CALL SOUNDs.

Sound Generator is public domain. I'll be glad to e-mail it to you, or you can send me $1 and I will put it on a TI disk and use the postal service to get it to you.

BYTE MAGAZINE and SCOTT FORESMAN page scans by Bill Gaskill

Bill Gaskill is an unofficial 99/4A historian. He has one of the largest collections of 99/4A documentation I know of and will be speaking about 99/4A history at Fest West '98. In order to help finance his trip to Lubbock for Fest West '98 he is selling digitized scans of some of the interesting documents in his collection. Details of his offerings are at http://www.gj.net/~lucky7/ which is Bill's Internet web site. This is, by the way, the most detailed, most informative historical 99/4A web site I have ever seen. Vast quantities of 99/4A information and photographs can be found here. To view the scanned documents you need a PC that runs Windows 3.1 or higher. Here is what is offered:

Byte magazine has given Bill permission to sell scans of 1978-1980 magazine pages with TI information leading up to the official release of the 99/4. You can read rumors about TI's impending entry into the personal computing market, the newly released chip set for Speak and Spell toys, the FCC's crackdown on TV interference from computers that hook to TVs, some 1979 full page ads from dealers selling the 99/4 even though they probably didn't actually have the product yet, a discussion of how the fate of the 99/4 is in transition as well as mention of the (never released) 99/7, a book about TI's magnetic bubble memory, problems TI had getting the FCC to accept its RF modulator, and the January 1980 official company announcement of the 99/4 with pricing photo and options list.

The Scott Foresman documents are also sold with permission. In many cases these documents explicitly state that they may be freely copied. Most of these documents relate to the School Management Applications packages. You could run an entire school district with this software, which came in command module form and required a 99/4 with disk system. Payroll, accounting, purchasing, and classroom management were all possible with these command modules. The company offered to sell both the software and the needed 99/4 hardware.

The documents include complete price lists for the Scott Foresman software and TI hardware. There are also full page ads for Scott Foresman mathematics and reading education command modules, some of which may not be familiar to you. Scott Foresman put product numbers on these command modules that are different from the PHMxxxx numbers used by TI for the same product. One of my favorites is the 1984 SF sale advertisement offering all its command titles at the close out price of $4.95 each. Some of the $4.95 titles I have never actually seen as an actual command module. All now cost more than $4.95, if you can find them.

These two sets of scanned pages are different from the Gaskill scans I offered through this column in early 1997. Bill is asking $6.95 for each of the two document sets. This includes postage and a copy of the Visioneer Viewer software needed to view and print the documents on a PC running Windows. If you can receive e-mail with attached files then Bill can e-mail you the documents. Otherwise he will use the postal service to mail the files to you on 3.5-inch PC formatted, high-density disks.

ACCESS

Bill Gaskill (source of scanned Byte and Scott Foresman pages) 2310 Cypress Court; Grand Junction CO 81506; e-mail lucky7@gj.net; TI related web page at www.gj.net/~lucky7

Charles Good (source of JM Base and Sound Generator); P.O. Box 647; Venedocia OH 45894; Phone (419) 667-3131; e-mail good.6@osu.edu

Ramcharged Computers (source of PC Transfer, needed to transfer e-mailed TIFILES TI software onto TI disks) 6467 E. Vancey Dr.; Brookpark OH 44142; Phone (216) 243-1244; e-mail RMarkus847@aol.com

CADD Electronics (source of PC99, needed to run TI software e-mailed in PC99 format) 45 Centerville Dr.; Salem NH 03979; e-mail mjmw@xyvision.com

GRAPHICS

Transferring digitized photos between a TI and a PC

BY ROGER PRICE

The transfer of a digitized photographs to the TI99/4A has been something that I have wanted to do ever since a photo processor started digitizing photographs several years ago. I have never seen any step-by-step method of how to do it. A person not having a PC may not know how or what to ask a friend to do to help him transfer a picture. This article is designed for that purpose.

To do the following process, you need a full TI system and either a PC or a friend with a PC and a cable to connect the RS232 ports, the program Telco or other terminal program on the TI and a terminal program on the PC. The programs GIFMANIA and TI-Artist, or a program to print out or view artist pictures are also needed.

Start by planning your picture so the main object has good contrast with the background. A blue car against a green hill background will likely not show up very well. Putting the car up against a white garage, for example, will turn out much better. With my PC I can use a digitized picture from a film processor, I can grab a picture from a videotape or I can use a video camera to digitize a picture or an object. At this point let us say that we have our picture digitized and it is on a 1.44-mb PC-type disk.

I start by loading the picture into a PC graphics program called Photoworks and use it to change the file to 256 colors. I then load the file into Windows 95 Paint. On my computer Paint cannot handle more than 256 colors. Now you can do any artwork you need to do, such as wiping out a background you do not like or puttinb in lettering. Then I save the picture back to the bitmap file.

Next I reload the picture back into Photoworks and check the contrast and brightness. I can also resize and crop the picture so I have the main object in the picture and resized to about 275 pixels wide by about 200 pixels high. At this point I save the picture as a GIF.

Cable Pinouts

The next step is to connect a cable between the RS232 ports on the TI and the PC. On my laptop I use the following connections:

PC Pin	TI Pin
2	3
3	2
4	6
5	7
6	20

A total of five wires are switched. Some people say you can get by with only two or three wires. Some people will say my pinouts are wrong, but a laptop has a nine-pin port while a desktop computer has a 25-pin RS232. The connectors are a nine-pin D-sub female on the PC and a 25-pin male D-sub on the TI. The connector cord I use was originally made so I could download disks from the TI-99/4A to the PC using the PC99 emulator.

GRAPHICS

Fig. 1

Now I load in Telco on my TI and, using Windows, I load in Hyperterminal on the PC. I set Telco to Terminal mode. Hyperterminal starts in the terminal mode and will auto connect to the terminal program in Telco. I do this to make sure there is a linkage of the two programs to start with.

A way to check is to type a letter on one computer. The connection is good if the letter shows on the screen of the other computer.

Once you are sure of the connection, select Fctn-9 on Telco and then select Download. On the PC select Transfer.

With a disk in the 4A ready to receive the picture file, use send in Hyperterminal to download the file. I use XMODEM, 8N1, ANSI and CRC checking. Other setups may work. The program in Windows called Direct-to-Direct Connect will not work. I think this works with PC-to-PC connections only.

What Next?

At this point you have your picture on your TI and you can tell your friend thanks for the download. Another way to download the picture is to have your friend upload the picture to a BBS and then you can download, but this requires a modem.

GIFMANIA is our link to TI-Artist and it loads in XBASIC, TI-Writer, or the Editor/Assembler cartridge. Select No. 1 from the GIFMANIA menu and load your GIF picture.

There are several options to view the picture. However, you must use the "M" option to load it for saving into TI-Artist for printout. For screen viewing in TI-Artist you can experiment with the options. Use function No. 2 to save the picture to a TI-Artist format.

Now you can load TI-Artist and load your picture. The process is now complete and your picture is in TI-Artist where you can print out the picture or do whatever. I added the caption "49 Ford" with TI-Artist. (See Fig. 1.)

Does your Artist picture have a line down the middle? Try CR on the end of your print code instead of LF.

If you need more instruction, help on downloading, or do not understand a part of this process, contact me at 1015 River Dr., Marion, IN 46952; or call me at (765) 664-6001.

USER NOTES

TIFILES headers

You might find the following information useful, or least illuminating. It was posted by Tim Tesch on the TI list server. —Ed.

A while back, someone asked for docs on the TIFILES header. Well, here we go....

The TIFILES header is used by the TI/Geneve to recreate a file and its type. This header allows us to transfer files from TI to TI, TI to Geneve, TI to IBM, or IBM to TI/Geneve all the while keeping the file intact and its format correct

How is it done? Well, the TIFILES header, used first in FasTerm, is set up as follows:

HEADER	BYTE >07	start of header, (MXT Ymodem uses >08)
TIFILES	TEXT "TIFILES"	

The next 8 bytes of the record are taken from the disk DSR's Addition Information structure. This data is read/written using DSR Opcodes >14, >15, >24, and >25; floppy read/write and hard drive read/write. It is structured as follows:

FILESECTORS	DATA xxxx	total sectors
STATUSFLAGS	BYTE xx	status, type of file, etc.
RECPERSECTOR	BYTE xx	records per sector
BYTESLASTSEC	BYTE xx	bytes used in last sector (pgm image)
LOGRECLEN	BYTE xx	logical rec length
RECSECUSED	DATA xxxx	records or sectors used

This is all that is required to properly send/receive files. However, certain extensions to the format were made. Mass Transfer first modified this record to make multiple file transfers possible. How?

If the byte at position 26 was not equal to zero, MXT knew that additional files were being sent. A "cheap" Ymodem, it allows TI/Geneve users to transfer multiple files without intervention.

Port (a terminal program for the Geneve) expands on this idea by including a flag at offset 28. If this flag is set to >FFFF, the following 8 words of data from position 30 through 37 are used to transfer the date and time stamp information for the transferred file. In this manner Port preserves date/time stamps.

The header is sent either as a 128-byte block or a 1024-byte block depending upon which protocol is in use.

A primer on resistors

Dick Bulmer wrote the following, which appeared in the Kawartha Kronicle.

Have you ever looked at the insides of your TI console, or your TV or VCR for that matter, and wondered what the colored bands on the resistors mean?

USER NOTES

Resistors?

Resistors are the little cylinders with a wire coming out of each end. They are the part of an electric circuit used to provide resistance. Here's a quick look at the codes, not intended to be complete, but may be enough to satisfy your curiosity.

Each resistor has at least three colored bands at one end.These are used to identify the value of the resistor. A fourth band is sometimes used to indicate

Continued on page 46

USER NOTES

Continued from page 45

wattage rating. At the other end, you will usually see another band (D in the graphic) that is colored gold or silver, indicating manufacturing tolerances.

> Each resistor has at least three colored bands at one end.

Black	0	Green	5
Brown	1	Blue	6
Red	2	Violet	7
Orange	3	Gray	8
Yellow	4	White	9

Gold identifies a 5 percent tolerance.

Silver identifies a 10 percent tolerance.

The color of band "A" gives the value of the first figure, band "B" that of the second and band "C" tells how many zeroes follow B. For example, if "A" is brown, "B" is orange and "C" is red, the value of the resistor is 1,300 ohms, or 1.3K ohms. A 27K (27,000) ohms resistor would be coded red, violet and orange.

Fest West '98 update

Here is the tentative schedule for the "TI Experience" at TI Lubbock facility:

8 to 8:30 a.m.	Registration/Sign-in at the facility
8:30 to 9 a.m.	Introductions and other "housekeeping"
9 to 10 a.m.	Lee Kitchens, former TI manufacturing engineering manager speaks.
10 to 10:30 a.m.	Break
10:30 to 11:30	TI99/4A historian Bill Gaskill speaks.
11:30 to noon	Finish up and leave the TI facility.

During the afternoon, seminars wills be conducted at the Sheraton Four Points Hotel. Vendors will also be present. A hospitality room will be available until 10 p.m.

As part of the speakers' time, there will be a question and answer session.

Refreshments will be served courtesy of Texas Instruments and there will be tours of the facility for those who are interested. All TI99ers are invited to this morning session, as are all employees of Texas Instruments and residents of Lubbock.

Texas Instruments has agreed to allow an official photographer at the TI Experience and a VCR recorder. The photographer will be Gary Cox who has taken pictures at many TI99 fairs. Otherwise, cameras are not permitted on the premesis.

A brunch will be held Sunday, Feb. 15, at the Four Points Hotel. The cost is $7.95 per person.

DMIII version works with Geneve

The following comes from the TI list server. It was submitted by Dan Eicher.

Jerry Coffey tipped me off to the fact that Disk Manager III was modified to work on the Geneve by Mike Dodd. I found a copy of this software on Beery Millers CD-ROM. I pulled this program and put it up on the Hugger BBS (317-782.9942).

From the bit of testing I have done, it works well with the Geneve, both with floppy drives and RAMdisk.

USER NOTES

SCSI DSR progress

David Nieters, SCSI DSR developer for Western Horizon Technologies, reported progress on the SCSI DSR recently.

In a posting to the TI list server he said "I believe I've fixed the bug that corrupts the VIB when trying to create a directory after the first sector of the bitmap is full. Since there are over 2,000 bits in the first sector of the bitmap and the first sector is only for file and directory headers (not actual file data — that goes in later in the bitmap), this bug does not rear its ugly head until after you have over 2,000 files and/or directories on your SCSI drive."

"We are testing this DSR now (hate to fix one thing and break 10 others). If you are using a version 1.1-x of the DSR and have close to 2,000 files on your SCSI drive, you may want to stop creating new files until you get this new DSR."

HRD crash recovery

We found this in the newsletter of the SouthWest 99ers. The author is unknown.

If your Horizon RAMdisk locks up and you can't access DSK1, but the disk conroller and HRD LEDs on the PEB are lit, try the following:

- Turn the PEB and console off.
- Insert the E/A module in the console.
- Turn on the console. (Turning the console on before the PEB seems to be critical to success.)
- Hold the shift key down and turn on the PEB.
- Select option 5 from E/A and load the HRD CFG file from DSK1.

Disk access should reappear and you should be able to access your RAMdisk directories.

Next, reload the ROS you normally use — do not throw out the existing information. Exit CFG and everything should be fine.