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 ❑ ❑ ❑

Card No.____________________
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.

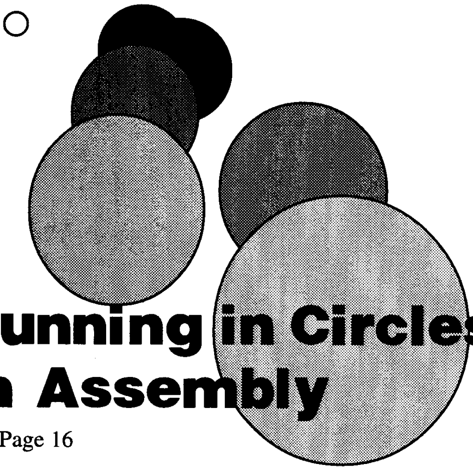
PERIODICALS

A 11/97 T
LEONARD CUMMINGS
815 NORTH BLACKHOOF
WAPAKONETA OH 45895

Covering the TI99/4A and Geneve home computers

MICROpendium

Volume 14 Number 5 September/October 1997 $6



Running in Circles in Assembly

See Page 16

Pointers – What are they?

See Page 32

INSIDE
Delphi dumps its TI forum
Fest West '98 is ready for Lubbock
Plus, news, views and reviews

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.

No information published in the pages of MICROpendium may be used without permission of the publisher, Burns-Koloen Communications Inc. Only computer user groups

While all efforts are directed at providing factual and true information in published articles, the publisher cannot accept responsibility for errors that appear in advertising or text appearing in MICROpendium. The inclusion of brand names in text does not constitute an endorsement of any product by the publisher. Statements published by MICROpendium which reflect erroneously on individuals, products or companies will be corrected upon contacting the publisher.

Unless the author specifies, letters will be treated as unconditionally assigned for publication, copyright purposes and use in any other publication or brochure and are subject to MICROpendium's unrestricted right to edit and comment.

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

THE CHICAGO TI USERS GROUP

presents the

15th Annual
Chicago TI International
World Faire
supporting TI 994A and 9640

November 8, 1997

EVANSTON PUBLIC LIBRARY
Evanston, Illinois
(corner Church & Orrington)
9:30 a.m. - 4:00 p.m.
Vendors-demonstrations-Seminars-User Groups
door prize drawings

----

for further information contact: **Hal Shanafield** (847)864-8644

COMMENTS

A format change

Sharp readers will notice that MICROpendium looks and feels a little different. We're doing it to save some money. Given our current subscription level, we were throwing out half of the pressrun. The minimum is more than double our subscriber list.

We're running 56 pages, which is the equivalent of about 28 of the full-size pages you're used to. Given the options, it's the best way for now. We thought about using full-sized, three-hole drilled sheets but that would have required 9x12 envelopes for mailing.

What's next for MICROpendium? Well, we don't expect Microsoft to buy a block of non-voting stock. We had a choice to make — either cut our printing cost dramatically, or cease publication.

We'll see how this works. Of course, it all depends on how many of you decide to continue to subscribe to MICROpendium. We will keep it going in one form or another as long as there is enough money to pay the bills. As always, we appreciate your continued support.

I also have a new email address: jkoloen@earthlink.net; and our web page is now at another location: www.earthlink.net/~jkoloen.

DELPHI CLOSES TI NET

TI99/4A users who attempted to log on to TI NET, Delphi's SIG for the TI99/4A and Geneve, found it closed as of Sept. 5.

According to messages on the TI list server maintained by Tom Wills, Delphi decided on discontinuing the service despite efforts by sysop Jerry Coffey. The text-only service was accessible by TI99/4As and Geneve computers.

UPDATE FROM RICH GILBERTSON

Good news and not so good news from Richard "RXB" Gilbertson. His system went down at Fest West, the victim of a grounding problem. As a word to the wise, he recommends grounding separate power supplies (such as for disk drives) through the plug and not the cable. What happened to his system, which was grounded through the cable, was that the ground ran back from the drive into several cards, burning out the interface chips. Fortunately, the chips are available at Radio Shack and are cheap, costing about $5 per card.

The delay in the availability of the RXB module will continue for a while. Don O'Neil, who is responsible for the card design, is busy with work and is working hard on the SCSI DSRs. The RXB module is next in line. Don, and Bud Mills, took a hit to the pocketbook because they had developed and produced 50 RXB boards only to find that everything was backwards and upside down. This cost about $1500. Ouch!

Version 1005 of RXB was supposed to be the last freeware/shareware version available. Somehow V1006 was pirated and made available. However, docs are not available for this version, and I'm not going to talk about all the nifty things it does. Let it suffice that the current version is 1011 and includes support for AMS, editor/assembler, and a very nifty disk manager. This version, or a later one, will

COMMENTS

be used in the RXB module when it comes out.

Extended BASIC users now you can delete, merge, and save files in Extended BASIC. RXB also lets users copy files, rename files and director, create directories, delete directories, protect and unprotect files. There's even a command to copy or rename all files or a single file from one device to another. It supports path names up to 256 characters long. By contrast, the Myarc Hard and Floppy Disk Controller has a limit of 39 characters. By the way, RXB disk manager routines work with any floppy or hard disk controllers. There's also a potentially dangerous command called CUT DIRectory that will automatically erase all the files in any directories that you want to delete. With the remove directory command, you first have to delete the files in the directory before the directory itself can be deleted. CUT DIR does it all in one press of the key. Imagine typing CUT DIR C:/ and pressing enter by mistake. Your entire disk drive can disappear. Gilbertson is aware of this and has included a verification line that asks the user whether he really wants the CUT DIR command executed. In the hands of a responsible user, this type of command can save a lot of time. But you definitely don't want to tell your younger children about this command.

At this point in its evolution, RXB includes a super-fortified version of Extended BASIC, editor/assembler, and a disk manager. Coming next is a sector editor that will work with any disk controller. Gilbertson says he can do this because he's writing it in GPL, which results in very compact code.

On the horizon for Gilbertson, as soon as he gets his system working again, is a project to rewrite GROMs for the Western Horizon Technologies keyboard interface. One of the things he's planning to do is a 16-bit emulation of AMS in the keyboard. It should be quite speedy.

Responding to a criticism about RXB's powerup routine, which appeared in the July/August edition of MICROpendium, Gilbertson says that the RXB powerup routine takes the lowest priority of any powerup routine. All other powerup routines, including RAMdisks and even BASIC, happen before RXB powers-up. This is because, he boasts, he is the only programmer to follow the TI standards for powerup routines.

Comparisons of Super Extended BASIC to RXB also leave Gilbertson a little weary. He notes that SXB is basically GRAM Kracker Extended BASIC with the addition of a plotting routine. GKXB was written by Danny Michaels. "RXB was done from scratch," Gilbertson says, proudly.

—JK

OBITUARY

Earl Raguse of Huntington Beach, California, died July 21. He held offices in the Users Group of Orange County and edited its newsletter, the UGOC ROM, and was the author of a number of articles on the TI99/4A.

On becoming ill, he donated much of his TI collection for the Southern California Computer Group for its school program, according to the SCCG newsletter, *The Computer Voice*.

Adventures with CALL KEY

BY EARL RAGUSE

The following article was originally published under the column XBASIC Miscellany.—Ed.

One of the things I learned while reworking TIPS 1.6 to 1.6/ER and writing TIPSLABEL was that TI didn't tell us in the Extended BASIC manual, or the later addenda, all we should know about Extended BASIC. At least in what I could find. In an article I had written about TIPS 1.6, I wrote that after I had converted TIPS to using CALL KEY(3,K,S), DISPLAY AT, and ACCEPT AT, I could not enter lowercase characters in ACCEPT AT.

I had some recollection that I had done it once upon a time, but I was not sure about it. Then I remembered that XB does not have a command to restore the lowercase character set once the set has been redefined. CHARSET does not do it. It only restores the uppercase set. That presumably was because the early versions of XB did not have a lowercase set. I then reasoned that since that was true, it made sense that ACCEPT AT would accept only uppercase characters.

I had plans for writing an assembly routine to LINK that would do it. I had once written an assembly program to take keyboard text input and, further, I knew that Adrian Robinson had written in the ROM newsletter a very detailed ACCEPT AT routine in assembly. My problem was that I didn't know how to get into Irwin Hott's LOADER program for TIPS. That is where the assembly routines are hidden, submerged below XB.

How wrong I was! I did not know until I got a call from Adrian (Robbie) Robinson that the problem was not with ACCEPT AT, but the fact that I had used CALL KEY(3,K,S) to ensure that all entries to CALL KEY would be in uppercase, instead of running them all through Ron Wolcott's assembly routine for converting inputs to uppercase. I didn't recall where I learned that CALL KEY(3) did that. Surely not in the XB manual. But I knew it. It turns out it was the Users Reference Guide.

What I didn't know was that once you do a CALL KEY(3,x,y), all — and I mean ALL — keyboard input thereafter, for CALL KEY, INPUT, LINPUT, ACCEPT AT, etc., is restricted to uppercase.

I had used that fact for CALL KEY in my DIRectory program. I didn't know that it stayed that way until you returned to the title screen. I also didn't know that you must do a CALL KEY(5,x,y) to restore normal upper and lowercase before any statement that calls for keyboard input. It matters not what "x" and "y" are, so long as they are legal numeric variables. Lowercase character redefinition has nothing to do with this. That is another story, where again Robbie used his assembly knowledge to help me out of an XB problem with CHARSET.

After the phone call, I searched everything I had on XB, to no avail. I could find nothing to tell me this. The best source on the keyboard is the User's Reference Guide (the "green" book), but it does not even imply that CALL KEY works that way.

About two days later I got a letter from Australia, from the Hunter Valley assembly guru Ross Mudie, telling me the same thing Robbie did. I then got suspicious. Why are the only people who know this the assembly guys? I then scoured the TI editor/assembler manual.

First, I found a reference to the User's Reference Guide. However, there was a discussion (see page 250) about the fact that the keyboard "device" was selected by placing a number — they discuss only numbers 0-3 — into >8374. (Hex numbers are indicated by preceding with ">" as in >8374.) Now this discussion makes no reference to CALL KEY. It is generic, and therefore refers to all keyboard input. Also, once a number is loaded into location >8374, it stays there until changed.

I can now assume that the XB CALL KEY does, among other things, a CALL LOAD of the key number into >8374 that requires a new CALL KEY or CALL LOAD statement to change >8374 to a new number. I have tried to test this theory in XB, but to no avail. Robbie says it works, but it won't work for me. If I were working in assembly, this would be rather understandable, but to the average reader of the XB manual, TI left it totally unexplained.

So what does all this mean? If you wish XB to return uppercase only, do a CALL KEY(3,x,y). To restore lowercase, do a CALL KEY(5,x,y). To keep the previous state (i.e., don't disturb the keyboard device previously selected), use CALL KEY(0,x,y). I note that most XB programmers use CALL KEY(0,x,y) almost exclusively. They are then not taking advantage of the computer's (and XB's) capabilities. I hope after this you will.

Harrison releases SCSI cataloger

Bruce Harrison has produced a 16-sector program called SCSICAT, which will catalog any root directory or subdirectory on a SCSI drive. By pressing P, the user has the option of printing out the contents. Instructions are included on the disk.

Harrison says that, although the utility is designed mainly for SCSI owners, it will work for any disk drive, including floppies and RAMdisks.

The program is available for $1 from Harrison at 5705 40th Place, Hyattsville, MD 20781, or from the Lima Users Group, P.O. Box 647, Venedocia, OH 45894.

MY-SIDEPRINT lets Geneve users print booklets

BY JIM UZZELL

The following program will print an ASCII file rotated 90 degrees in two columns of 40 characters, each with 60 lines, using graphic commmands, producing a full page.

There are some requirements to use this program. First, you must change the graphics commands to match your printer, which are explained below.

Second, the file MUST be exactly 120 lines long and the width cannot exceed 40 characters unless you use A4 paper.

It is recommended that MY-Word be used to create or reformat a file to fit the above limits. The file cannot have special characters, including the carriage return or the tab info, so you must save the file using the PF C option of MY-Word.

The following lines MUST be changed to match the printer you are using:

Line 170 defines the graphic mode to normal density with a maximum of 240 dots. This is the "K" graphics command. The 240 dots equal four inches, which equals 30 characters times 2 equals 60 characters, or 480 dots, and is the maximum for the "K" mode.

Line 180 sets the line spacing to 24/216ths. Equivalents are 40/360, 20/180, and 8/72.

Line190 sets one tab at 40.

This program looks at each line of text and pads each line to exactly 40 characters if necessary. This is is done in lines 230-250. If you use A4 paper, you can increase the character width to use the extra length of paper.

After you have replaced the graphic commands, do the following: In MY-Word create an ASCII file of 40 characters and 120 lines using the uppercase O. Then remove the "REM" in line 580. This will allow you to print two lines of the file so you can adjust the line spacing. After that, put the REM back and test the complete file.

The ESC key is active during the printing process, in case there is a need to stop the printing, and will help reduce the paper waste. This program can be used to produce a pamphlet. To do this, create or reformat a document. Then divide the total lines by 60. If it does not divide equally, add blank lines to the end of the file. Then save each 60 lines to a file. Then merge the files, i.e last and first, second and next to last, etc.

Before you start merging, you must determine which files to merge, depending on whether you want page 1 to be actually page 3 by using a blank sheet of

paper as the front and back cover. A method that works for me is using the tractor feed strips. I use pieces of these to represent total sheets of paper required. Each sheet equals two files (front and back). I fold all of these together, with the open end to the right. Then I number them, which tells me the files I need to merge together.

Example — one strip equals four pages and page 3 would be merged with page 2. Also page 1 would be merged with page 4. The final format would be 2-3, 4-1, which is two files of 120 lines.

If you want page numbers, you can divide your document by 58, then add a blank line and a page number line. Using page numbers requires you to determine if the first page is actually page 1, i.e. a cover sheet with info inside or a table of contents. If you do not center the page numbers, the odd numbers are toward the right margin, even numbers are toward the left margin.

This program is slow because the recalculation of character patterns is 100 percent MY-BASIC code.

MY-SIDEPRINT

```
100 !MYSIDEPRINT
110 !by DDI SOFTWARE
120 !COPYRIGHT 1997
130 DIM X$(120)
140 BL$=RPT$(CHR$(0),8)
150 Z,S=0
160 CLS :: GOTO 670
170 DEF DF2$=CHR$(27)&CHR$(7
5)&CHR$(240)&CHR$(0)
180 DEF DF3$=CHR$(27)&CHR$(5
1)&CHR$(24)
190 DEF DF4$=CHR$(27)&CHR$(6
8)&CHR$(40)&CHR$(0)
200 OPEN #2:DISK$,INPUT ,DIS
PLAY ,VARIABLE 80
210 FOR X=1 TO 120
220 LINPUT #2:X$(X)
230 IF LEN(X$(X))=1 THEN IF
ASC(X$(X))=32 THEN X$(X)=RPT
$(" ",40)
240 IF LEN(X$(X))=0 THEN X$(
X)=RPT$(" ",40)
250 IF LEN(X$(X))<40 THEN X$
(X)=X$(X)&RPT$(" ",40-LEN(X$
(X)))
260 NEXT X
270 CLOSE #2 :: GOSUB 380
280 FOR X=1 TO 40 :: GS$=""
:: GS1$="" :: Y$=""
290 FOR Y=60 TO 1 STEP -1
300 Y$=Y$&SEG$(X$(Y+S),X,1)
310 NEXT Y
320 IF LEN(Y$)<60 THEN Y$=Y$
&RPT$(" ",60-LEN(Y$))
330 FOR Y=1 TO 30
340 A1=ASC(SEG$(Y$,Y,1)) ::
A2=ASC(SEG$(Y$,Y+30,1))
350 IF A1<33 THEN GS$=GS$&BL
$
360 IF A2<33 THEN GS1$=GS1$&
BL$
370 GOTO 420
380 OPEN #1:"PIO.CR",VARIABL
E 254
390 PRINT #1:DF4$;DF3$
400 PRINT #1:
410 RETURN
```

Continued on page 10

MY-BASIC

Continued from page 9

```
420 IF A1>32 THEN CALL CHARP
AT(A1,CH$)
430 IF A2>32 THEN CALL CHARP
AT(A2,CH1$)
440 FOR L=15 TO 1 STEP -2
450 IF A1>32 THEN B1=VALHEX(

SEG$(CH$,L,2))
460 IF A2>32 THEN B2=VALHEX(
SEG$(CH1$,L,2))
470 IF A1>32 THEN GS$=GS$&CH
R$(B1)
480 IF A2>32 THEN GS1$=GS1$&
CHR$(B2)
490 NEXT L :: NEXT Y
500 PRINT #1:DF2$;
510 FOR C=1 TO 239 :: PRINT
#1:SEG$(GS$,C,1);
520 NEXT C :: PRINT #1:SEG$(
GS$,240,1)
530 PRINT #1:CHR$(27);CHR$(1
0);
540 PRINT #1:CHR$(9);DF2$;
550 FOR C=1 TO 239 :: PRINT
#1:SEG$(GS1$,C,1); :: NEXT C
560 PRINT #1:SEG$(GS1$,240,1
)
570 CALL KEY(0,K,SS) :: IF K
=155 THEN 610
580 !IF X=2 THEN 610
590 NEXT X :: IF S=60 OR Z=1
 THEN 610
600 Z=1 :: S=60 :: PRINT #1:
 :: PRINT #1: :: GOTO 280
610 PRINT #1:CHR$(12);CHR$(2
7);"@";
620 CLOSE #1 :: GOTO 640
630 CLS :: END
640 DISPLAY AT(10,1):"PRINT
ANOTHER DOCUMENT N"
650 ACCEPT AT(10,24)SIZE(-1)
:N$
660 IF N$="N" THEN 630
670 DISPLAY AT(6,1):"  MYSID
EPRINT"
680 DISPLAY AT(8,1):"By DDI
SOFTWARE"
690 DISPLAY AT(12,1):"ENTER
path.filename"
700 ACCEPT AT(13,1)BEEP :DIS
K$
710 CLS :: CALL MEMSET(X$(),
"") :: Z,S=0
720 GOTO 200
```

NOTEWORTHY

An Extended BASIC game with a musical theme

This Extended BASIC game, called Noteworthy, was written by R. Trueman.

The object of the game is to direct a critter to eat musical notes and ascend through various levels to the top. It uses keyboard input, specifically the S and D keys to move and the P key to jump. As it is, it runs slowly but the graphics are quite well done. It requires a memory expansion.

```
100 CALL CLEAR :: CALL CHARS
ET :: CALL SCREEN(2):: RANDO
MIZE :: CALL MAGNIFY(3):: CA
LL CHARPAT(89,Y$):: CALL CHA
```

NOTEWORTHY

```
R(74,Y$)!251
110 CALL COLOR(3,7,11,4,7,11
,5,5,8,6,5,8,7,5,8,1,16,2,2,
13,11,8,4,15,11,2,16)!208
120 CALL CHAR(119,"080C0B083
878783",81,"",95,"",124,"",1
25,"0E115111F38777F",126,""
,127,"7088A888F81CEEFE")!110
130 CALL CHAR(92,"1833777F7E
7C7839391C0E47A78FFF7F0080C0
E0202020A022257FFF0780C2FE")
!023
140 CALL CHAR(136,"000103070
404040544A4FEFFE001437F18CCE
EFE7E3E1E9C9C3870E2E5F1FFFE"
)!148
150 CALL CHAR(106,"18247E16F
7E7C7FF",40,"4B4B4B4B4B4B4B4
B012452ACB6DBBFFFF8A4DAFAE8D
AF4E110461AA4532E1B57",118,"
30787838080B0C08")!004
160 CALL CHAR(107,"18247E5ED
7C3F7FF",44,"089228D5564AAF6
DF0C4E9B4CAFBFDFF270B532F97B
7EFFF",116,"FFFAF2E2C2828282
")!217
170 CALL CHAR(36,"FF818181FF
000000FF81818181FF0000FF8181
818181FF00FF818181818181FF",
33,"FFFF",34,"FF81FF",35,"FF
8181FF")!146
180 CALL CHAR(88,"FFFDDB6D35
4A2480872F5B172F5B251FFFF7ED
E9F4CAB0E4FFBFDF532D97230F")
!252
190 CALL CHAR(117,"FF5F4F474
3414141",140,"01010101010101
01FF403F100F0403018080808080
808080FF02FC08F020C08")!066
200 CALL CHAR(60,"0E1115111F
3D777BFCFFFF7F7F3F1F0F7088A8
88F8B8EEDE3FFFFFFEFEFCF8E",1
04,"18247E76E7F73CFF",105,"1
8247E66F7EFECFF")!209
210 CALL CHAR(128,"071F3F716
EEFEDF7F9FFFF7F7F3F1F07E0F8F
CC6829B93C7FFFFE5C194FCFCE")
!080
220 CALL CHAR(132,"071F3F634
1D9C9E3FFFFA783293F3F07E0F8F
C8E76F7B7EF9FFFFFFEFEFCF8E")
!067
230 CALL CHAR(59,"6C547EC6FE
FE7C38",108,"03060C0C1F3F7F7
FFFFFFF7F7F3F1F07C020E0A0F8E
CFEDEE0FFFFFEFEFCF8E")!170
240 CALL CHAR(112,"030407050
71F377F7B07FF7F7F3F1F07C0603
030F8FCFEFEFFFFFFFEFEFCF8E")
!227
250 CALL CHAR(96,"0728382909
78F8FFBFC0F1F1F9FD7F2AE014DC
F4B0DE1FFFFD038F8F9FBFFE54")
!015
260 CALL CHAR(120,"07283B2F0
D7BF8FFBFC0F1F1F9FD7F25E0141
C94901E1FFFFD038F8F9FBFFE54"
)!024
270 CALL CHAR(58,"",100,"1E2
32333231F0373DB8F0703318BC77
F000000000080848AC8C8E8F8F8F
CFEFF")!156
280 SC,BO,CO=0 :: ME=3 :: LE
,WA=1 :: CALL CHAR(104,"0000
000000121511313171F1F3F7FFF
78C4C4CCC4F8C0CEDBF1E0C08CD1
E3FE")!159
290 DISPLAY AT(2,1):"ZXXX[ Z
[ ZXXXX[ ZXXXX[ ZXXX[*+),Y *
Y *+)),Y -),+). *+)).*Y *Y *
Y *Y  *Y   *Y   *Y   *Y *Y *
Y *Y  *Y   *Y   *[X[" !117
300 DISPLAY AT(6,1):"*Y *Y *
```

Continued on page 12

NOTEWORTHY

Continued from page 11

```
Y *Y  *Y   *Y   *+). *Y *Y *
Y *Y  *Y   *Y   *Y   *Y *[XZ
Y *[XXZY   *Y   *[XX[-. -)))
. -)))).   -.   -)))." !153
310 CALL HCHAR(20,1,81,160)!
017
320 DISPLAY AT(12,1):"!
  !    !!! `'' ! ! ` `#
  #    # #  `  # # ` ` $
 $     $ $  &  $$$ & & %
 %  %  %%%  %  % % %%%" !128
330 DISPLAY AT(16,1):"  & &
&  $ $ &&   $  & &  $    ` `
`  # # ` `            #   ``''
`   !  `  ` !!!!!!!!!" !173
340 DISPLAY AT(20,1):"QQQQQQ
FROMQQRQTRUEMANQQQQQQQQQQQQQ
QQTHEQBADDIESQQQQQQQQQ1QROBO
IDSQQQ2QSLUGQCREATURES3QGRUM
PIESQQ4QMADQDOGSQQQQQQ" :: G
OSUB 1090 !158
350 CALL KEY(0,K,S):: IF S=0
 THEN CALL SOUND(-90,(RND*20
)+110,19):: GOTO 350 !235
360 CALL KEY(0,K,S):: IF S=-
1 THEN CALL SOUND(-90,(RND*2
0)+110,19):: GOTO 360 ELSE C
ALL HCHAR(20,1,81,160):: DIS
PLAY AT(20,1):"SQANDQDQTOQMO
VEQQLEAPQWITHQP" !182
370 DISPLAY AT(21,1):"WHENQU
NDERQLARGESTQPANELSQTOGOQTOQ
HIGHERQFLOORQQQPICKQUPALLQNO
TESQFORQAQBIGGERQBONUS" :: G
OSUB 1090 !248
380 CALL KEY(0,K,S):: IF S=0
 THEN CALL SOUND(-90,(RND*20
)+110,19):: GOTO 380 ELSE CA
LL CLEAR :: CALL COLOR(1,10,
16):: CALL SCREEN(16)!058
390 CALL VCHAR(3,1,42,22)::
CALL VCHAR(3,31,42,22):: CAL
L VCHAR(3,2,89,22):: CALL VC
HAR(3,32,89,22):: FOR Y=3 TO
 23 STEP 5 !151
400 CALL HCHAR(Y,2,91):: CAL
L HCHAR(Y+1,2,43):: CALL HCH
AR(Y,31,90):: CALL HCHAR(Y+1
,31,44):: NEXT Y !121
410 DISPLAY AT(1,1):"QQMEN";
TAB(14);"SCORE";SC:"LEVEL";L
E;TAB(14);"BONUS";LE*10 :: C
ALL HCHAR(1,9,59,ME):: ON WA
 GOTO 420,510,560,610 !079
420 DISPLAY AT(3,1):"XXXXXXX
XXXXXXXXXXXXXXXX[``ZX))))))
)))))))))))tu))). -)

                          vv
 w v  w   vw  w  w   " !236
430 DISPLAY AT(8,1):"X[``ZXX
XXXXXXXX[$$ZXXXXXXXXXX). -))
tu)))))). -)tu))tu))

                          v
 wvv      w  v    v" !161
440 DISPLAY AT(13,1):"XXXX[`
`&&%%$$ZXX[&&##%%ZXXXX)))).
       -)).      -))))

                          v
 w  w    v ww  w v w" !130
450 DISPLAY AT(18,1):"XXXXXX
XX[``Z[$$ZXXXX[!!''!!Z))))))
tu.  -.  -))tu.       -

                         w w
w   vv   w   v   v  wv" !219
460 RESTORE 930 !002
470 DISPLAY AT(23,1):"XXXXXX
XXXXXXXXXXXXXXXXXXXXXX))))))
)))))))))))))))))))))))" !171
480 FOR Y=2 TO 7 :: READ A,B
 :: CALL SPRITE(#Y,140,14,A*
8+1,B*8+1):: NEXT Y :: READ
A,B,D,FT,HT :: CALL SPRITE(#
8,HT,D,A,B,0,1+LE)!093
```

NOTEWORTHY

```
490 READ A,B :: M=A :: N=B :
: CALL SPRITE(#1,108,9,M,N)!
096
500 CALL KEY(0,K,S).:: IF S=0
 THEN CALL SOUND(-90,(RND*20
00)+2000,10):: GOTO 500 ELSE
 710 !029
510 DISPLAY AT(3,1):"XXXXXXX
XXX[``ZXXXXXXXXXXXXXX)))))))
)tu.  -)))))))))))))

                      wv w v
ww    v    ww    v  v" !120
520 DISPLAY AT(8,1):"XXXXXXX
XXXXX[``&&%%$$####!!Z)))tu))
)))tu.                -

                          vv
wv w   vwww  v  w v" !105
530 DISPLAY AT(13,1):"XXXXXX
XXXXXXXXX[``ZX[$$&&ZXX))tutu
)))))))tu.  -).    -))

                          wv
 wv w          v   vv" !208
540 DISPLAY AT(18,1):"X[``##
!!''&&ZXXXX[&&$$ZX[``Z).
      -)))).    -).  -

                         v w
 v   wwv         vv   w" !060
550 RESTORE 940 :: GOTO 470
!180
560 DISPLAY AT(3,1):"X[``ZX[
&&ZXXX[##%%ZXXXXXXXXX).  -).
  -))).    -)))))))))

                        wv  v
   wwvw   vvv  wv v" !029
570 DISPLAY AT(8,1):"XXXXXXX
XXXXXXXX[!!%%''%%!!ZX)))))))
tututu)).             -)

                      wvwvv
        wvv  w  w  v" !153
580 DISPLAY AT(13,1):"[##''%
%ZXXXXXXXXXXXXXXXXXXXXX.
 -))tu))tu)))))))))))))

                           v
vv  www    vv v   www" !163
590 DISPLAY AT(18,1):"X[``ZX
XXXXXXXXX[``ZXXX[$$&&Z). -)
))))))))). -)tu.     -

                          vw
vwv   w vv   wwv   v" !155
600 RESTORE 950 :: CALL SPRI
TE(#9,128,15,41,220,0,1+LE):
: GOTO 480 !147
610 DISPLAY AT(3,1):"XXXXXXX
XXXXXXXXXXXXXXX[``ZXX)))))))
)))tu)tu)tu)tu). -))

                        v  wwv
   wv vw vv ww    wv" !064
620 DISPLAY AT(8,1):"X[``ZXX
XXXXXXXXXXXXXXXXXXXXX). -))
))tu)))))))))tu))))))

                        wv  v
  w  v wv w  w  w v" !243
630 DISPLAY AT(13,1):"X[$$%%
ZXX[&&ZXXXX[##!!ZX[``Z).
-)). -)))).     -). -

                          w v
vw   wvw    wvwv" !052
640 DISPLAY AT(18,1):"X[``$$
ZXX[##Z[%%ZXXXX[!!%%ZX).
-)). -. -)))).    -)

                          wv
w  w v   wwv   vw  wv" !051
650 RESTORE 960 :: CALL SPRI
TE(#9,92,5,161,1,0,3+LE):: G
OTO 480 !011
660 CALL MOTION(#1,0,0):: CA
LL PATTERN(#1,60)!080
670 Y=(RND*18)+2 :: IF Y>7 T
HEN 690 ELSE CALL POSITION(#
Y,A,B):: CALL LOCATE(#Y,A+8,
B):: CALL PEEK(-31877,O):: I
F O AND 32 THEN 970 !024
680 CALL LOCATE(#Y,A,B):: GO
```

Continued on page 14

NOTEWORTHY

Continued from page 13

```
TO 710 !104
690 IF Y<17 THEN 710 :: IF Y
<18 THEN 700 :: CALL MOTION(
#8,0,-1+(LE-(LE*2)),#9,0,-1+
(LE-(LE*2))):: CALL PATTERN(
#8,FT,#9,FT):: GOTO 710 !131
700 CALL MOTION(#8,0,1+LE,#9
,0,1+LE):: CALL PATTERN(#8,H
T,#9,HT)!123
710 CALL PEEK(-31877,O):: IF
 O AND 32 THEN 970 !027
720 CALL KEY(0,K,S):: IF S=0
 THEN 660 ELSE ON POS("PpSDs
d",CHR$(K),1)+1 GOTO 660,770
,770,730,750,730,750 !079
730 CALL POSITION(#1,A,B)::
IF B<20 THEN CALL MOTION(#1,
0,0):: GOTO 670 ELSE CALL MO
TION(#1,0,-4):: CALL PATTERN
(#1,112)!070
740 K=INT(A/8)+1 :: Y=INT(B/
8)+1 :: CALL GCHAR(K,Y,O)::
IF O<118 THEN 670 ELSE GOSUB
 990 :: CALL HCHAR(K,Y,32)::
 GOTO 670 !097
750 CALL POSITION(#1,A,B)::
IF B>220 THEN CALL MOTION(#1
,0,0):: GOTO 670 ELSE CALL M
OTION(#1,0,4):: CALL PATTERN
(#1,108)!189
760 K=INT(A/8)+1 :: Y=INT(B/
8)+3 :: CALL GCHAR(K,Y,O)::
IF O<118 THEN 670 ELSE GOSUB
 990 :: CALL HCHAR(K,Y,32)::
 GOTO 670 !099
770 CALL MOTION(#1,0,0):: CA
LL POSITION(#1,M,N):: CALL G
CHAR(M/8-2,N/8+1,O):: IF (O>
33)*(O<40)THEN 780 ELSE 670
!102
780 CALL GCHAR(M/8-2,N/8+2,P
):: IF P<>O THEN 670 ELSE CA
LL COLOR(#1,10):: Y=O :: O=M
-42 :: T=M/8-2 !146
790 IF M<17 THEN 1050 ELSE M
=M-7 :: CALL LOCATE(#1,M,N):
: Y=Y-1 :: CALL HCHAR(T,N/8+
1,Y,2):: IF Y=33 THEN 800 EL
SE 790 !070
800 IF M=O THEN 830 ELSE CAL
L PEEK(-31877,K):: IF K AND
32 THEN 930 !004
810 FOR Y=M TO O+42 :: CALL
PATTERN(#1,108,#1,112):: CAL
L LOCATE(#1,Y,N):: CALL PATT
ERN(#1,60):: NEXT Y :: M=Y :
: CALL COLOR(#1,9)!104
820 GOTO 670 !239
830 CALL COLOR(#1,9):: M=M+2
 :: CALL LOCATE(#1,M,N):: GO
TO 670 !019
840 CO=11 :: CALL HCHAR(18,1
4,32,3):: CALL HCHAR(19,14,3
2,3):: CALL SPRITE(#1,60,9,1
21,141,#15,108,9,121,1)!152
850 CALL SPRITE(#14,92,8,121
,16,#13,128,11,121,32,#12,10
4,5,121,48,#11,96,14,121,64,
0,8)!214
860 CALL COINC(#CO,121,113,4
,O):: IF O=-1 THEN 870 ELSE
860 !003
870 CALL MOTION(#CO,16,0)::
FOR Y=610 TO 110 STEP -50 ::
 CALL SOUND(-60,Y,0):: NEXT
Y :: CALL DELSPRITE(#CO):: C
O=CO+1 !185
880 IF CO<15 THEN CALL MOTIO
N(#CO,0,8):: GOTO 860 ELSE C
ALL MOTION(#15,0,8)!062
890 CALL COINC(#1,#15,16,O):
: IF O=0 THEN 890 ELSE CALL
MOTION(#15,0,0):: CALL PATTE
RN(#1,112)!056
```

NOTEWORTHY

```
900 CALL CHAR(100,"1C3E7FFFF
FFFFFFF7F7F3F1F0F070100387CF
EFFFFFFFFFFFEFEFCF8F0E08"):: 
 CALL SPRITE(#9,100,7,100,13
2,-2,0)!081
910 ME=ME+1 :: FOR Y=1 TO 7
:: FOR T=610 TO 1110 STEP 50
 :: CALL SOUND(-50,T,0):: NE
XT T :: CALL HCHAR(1,9,32,8)
:: CALL HCHAR(1,9,59,ME)!164
920 NEXT Y :: CALL DELSPRITE
(ALL):: CALL CHAR(100,"1E232
333231F0373DB8F0703318BC77F0
0000000080848AC8C8E8F8F8FCF
EFF"):: GOTO 410 !010
930 DATA 3,20,8,9,8,26,8,22,
18,8,18,20,121,129,14,120,96
,161,81 !107
940 DATA 3,10,8,12,8,5,13,4,
13,6,13,15,161,1,5,100,104,1
61,209 !019
950 DATA 8,9,8,11,8,13,13,12
,13,16,18,22,41,100,11,132,1
28,161,192 !235
960 DATA 3,12,3,15,3,18,3,21
,8,11,8,22,121,192,8,136,92,
161,209 !092
970 Y=1 :: CALL SOUND(-90,-7
,0):: SC=SC+BO :: DISPLAY AT
(1,19):SC :: CALL MOTION(#1,
0,0):: CALL PATTERN(#1,124):
: BO,CO=0 :: ME=ME-1 !167
980 CALL COLOR(#1,(RND*13)+2
):: Y=Y+1 :: IF Y<20 THEN 98
0 :: CALL DELSPRITE(ALL):: C
ALL HCHAR(1,9+ME,32):: IF ME
=0 THEN 1000 ELSE 410 !104
990 BO=BO+(LE*10):: T=(RND*2
000)+1000 :: CALL SOUND(-50,
T,0):: RETURN !148
1000 CALL DELSPRITE(#1,#7)::
 FOR Y=10 TO 16 :: CALL HCHA
R(Y,10,81,14):: NEXT Y :: DI
SPLAY AT(11,10)SIZE(10):"GAM
EQQOVER" :: HI=MAX(HI,SC)!01
8
1010 DISPLAY AT(13,10):"HIGH
";:: DISPLAY AT(13,15):"Q"&S
TR$(HI)&"Q";:: DISPLAY AT(15
,9)SIZE(12):"REPLAJQJQ/QN" !
192
1020 CALL KEY(0,K,S):: IF K=
78 OR K=110 THEN 1100 ELSE I
F K=121 OR K=89 THEN 1030 EL
SE 1020 !035
1030 FOR Y=1 TO 20 :: CALL S
CREEN(RND*13+2):: NEXT Y ::
CALL SCREEN(16):: ME=3 :: SC
,BO,CO=0 :: LE,WA=1 !016
1040 CALL DELSPRITE(ALL):: C
ALL CLEAR :: GOTO 390 !030
1050 CALL DELSPRITE(#1,#8,#9
):: SC=SC+(LE*1000):: FOR Y=
1 TO 21 STEP 2 :: CALL SOUND
(-90,2000,Y):: NEXT Y :: DIS
PLAY AT(1,19):SC !072
1060 DISPLAY AT(2,19):BO ::
FOR Y=1 TO 10 :: CALL HCHAR(
2,21,32,8):: DISPLAY AT(2,19
):BO :: CALL SOUND(-10,(2000
)+2000,0):: NEXT Y !228
1070 SC=SC+BO :: DISPLAY AT(
1,19):SC :: BO=0 :: DISPLAY
AT(2,19):BO !092
1080 LE=LE+1 :: WA=WA+1 :: C
ALL DELSPRITE(ALL):: IF WA<5
 THEN 410 ELSE WA=1 :: GOTO
840 !023
1090 DISPLAY AT(24,1)BEEP:"Q
QPRESSQSOMETHINGQTOQSTARTQQ"
 :: RETURN !015
1100 CALL CLEAR :: END !222
```

Running in circles

BY BRUCE HARRISON

Some time back, we wrote about drawing a straight line on the computer screen, and presented a method based on an algorithm developed by a man named Bresenham. That algorithm uses only very simple integer math to draw an optimal straight line on our screen.

That article caused some reaction among our readers about who this fellow Bresenham was and how his method actually worked. We received correspondence from John H. Bull and Phil Van Nordstrand, both of whom started searching for the mysterious Bresenham. Van Nordstrand found a reference in Dr. Dobbs' Journal about a circle-drawing algorithm by the very same Bresenham. Now that really made us curious. Could this genius Bresenham have made a "simple math" way of generating circles as well? How could he avoid using sines or cosines or square roots and still generate a circle? Yes, he could, and in a way that's even more mysterious than the straight line.

Van Nordstrand was able to find one of the books referenced in the Dr. Dobbs source, in the Houston Public Library. He sent along copies of a few pages, which had the algorithm as implemented in Pascal code. After studying this for a few minutes, it became evident that this would be easy to translate from Pascal to TI assembly, and that it would execute fairly fast. The algorithm requires that some numbers be multiplied, but always by powers of two, and thus simple SLA instructions would accomplish the needed multiplications. Other than that all the math is simple integer comparison, addition, and subtraction.

THE ALGORITHM

We enter the algorithm with three numbers. These are the X and Y coordinates of the circle's center and its radius. The algorithm itself calculates points to be plotted only for one-eighth of a circle, so we have to devise our own method of replicating the points for the rest. The algorithm uses a single parameter which is initially derived from the radius by this formula:

P=3-2*R

For the moment, we'll ignore the center coordinates. The first point to be plotted is at the top of the circle, so X is zero and Y is equal to R. (In our implementation, we add the center coordinates for each point as it gets plotted.) We now plot the point at the top of the circle. After each plotted point, we examine the parameter P, and adjust its value in one of two ways.

If P<0, we don't change Y, but adjust the value of P by this formula:

P=P+4*X+6

If P>=0, we calculate the new value of P by the formula:

P=P+4*(X-Y)+10

and then subtract one from Y. Note the formula uses the values of X and Y from the previous point, so Y is adjusted after calculating the new value of P.

In either case, we increment X after calculating the new value of P. We continue doing this until X is greater than Y, which indicates that we've finished one-eighth of a circle. Notice that there are multiplications involved in the derivation and adjustment of P, but these are by 2 or 4 in all cases, so that in our assembly version we can accomplish the multiply by shifting a register left by one or two bits.

OUR ASSEMBLY IMPLEMENTATION

In our implementation, which is in the sidebar at label BCIRC, we've stashed away the center's coordinates, so the algorithm only needs to deal in "deltas" for X and Y. At the outset, we establish four such deltas, called DELXP, DELXM, DELYP and DELYM. The DELX's are set to zero, and the DELYP and DELYM are set to plus and minus the value of the Radius, respectively.

For each pass through the algorithm, then, we plot four points, two in the top quarter circle and two in the bottom. Thus our circle grows from top and bottom centers both left and right, so that when we reach the one-eighth circle limit, we've got a half circle, one quarter at the top and one quarter at the bottom. To complete the circle, we repeat the entire process (at label HALF2) with the roles of DELX and DELY interchanged, so the quarter circles at either side get drawn. All of this is being done in bit-map mode, so the circle is a single-pixel thickness. Each point gets plotted by the PLOT subroutine, which dates back to our first column on bit-map operation (Part 42).

As with the Bresenham line algorithm, we can see that this works, and it makes optimally round circles on our screen, but we don't know why it works. If we knew where to reach him, we'd ask Bresenham, but probably wouldn't understand his answer. John Bull was able to discover that Bresenham was a mathematician who worked for IBM. He actually found the original papers containing derivations and proofs of the algorithm's effectiveness. As John and I suspected, these are pretty heavy going.

TODAY'S SIDEBAR

Yes, it's a complete program in E/A source code, with as much annotation as we could stand to do. This program uses modified versions of our old standby SETBM and SETGM subroutines, the old PLOT subroutine, and of course the BCIRC subroutine, which uses PLOT to place the pixels on the circles. The action starts with a circle at the center of the screen with radius 10, then keeps adding ten to the radius until the screen is filled with concentric circles. Note that the outer ones would run off the screen edges, but we've put in a limit check before actually plotting each point, so the outer circles go up to but not past the screen edges. In some cases you might want your circles to wrap around from edge to edge of the screen, but we'll leave the method for doing so to the serious student of assembly. (Hint: this is easier than you might think.)

Continued on page 18

Continued from page 17

Before the mail comes, we'll confess that the code in the sidebar is nowhere near optimized in any respect. It's the result of a quick and easy attempt to test and apply the algorithm, so it may even appear crude and wasteful of memory. Nonetheless, it shows that this works, and quite well.

The algorithm itself is crash-proof. You can, for example, start with radius=0, and the screen will just get a single pixel plotted at the center. For those of you who want to try this, just go into the sidebar and find label CIRCLE. Make that label say either CLR R0 or LI R0,0, then assemble the result. When run, you'll see a single dot at the center of the family of concentric circles. That's the circle of radius 0. Radius 1 will produce a single pixel open space surrounded by four black pixels in a diamond pattern.

The sidebar program doesn't do much. It puts the computer into bit map mode, then creates a series of concentric circles starting with radius 10 and growing by 10 on each iteration. Only the first nine will fit completely on the screen. The rest are shown partially only so far as their pixels fit on the screen. The code at label CPLM makes sure that we don't try putting any points off the edges of the screen.

For the benefit of those who get MICROpendium on disk, we've included the object file SIDE66/O along with this submission. As in the case with Bresenham's line drawing algorithm, we've also done a simple Extended BASIC implementation of the circle, as shown in the sidebar in 28-column listing. This XB program should also be on your MICROpendium disk as CIRCXB. This will give you a chance to play around with the algorithm without having to use assembly code. Since this XB version goes slowly and puts a very coarse representation on the screen with cursor characters, you'll be able to see more clearly what's happening as the circle gets generated.

The circle algorithm, pretty much as shown here, has been incorporated into our drawing program, in both the 9-pin and 24-pin versions. As we mentioned last month, those programs allow circles from the screen to be printed as circles on paper. If you're using an old version of our drawing program, now might be a good time to order an updated copy from our friends in Lima, Ohio.

That's it for this time. Next time we'll be discussing the topic of "sound lists," and some new ideas and programs to make that concept easier to grasp and use. See you then.

Sidebar 66

```
* SIDEBAR 66
* BITMAP CIRCLES
* CODE BY Bruce Harrison
* 20 JUL 1995
* PUBLIC DOMAIN
```

```
*
* A COMPLETE PROGRAM THAT PUTS THE COMPUTER
* INTO BIT-MAP MODE AND DRAWS A SERIES OF
* CONCENTRIC CIRCLES USING BRESENHAM'S ALGORITHM
*
        DEF  START          DEFINE ENTRY POINT
        REF  VWTR,KSCAN,VMBW,VMBR,VSBW,VSBR
START   LWPI WS             LOAD OUR WORKSPACE
        LI   R0,>380        POINT AT COLOR TABLE
        LI   R1,SAVCLR      AND AT STORAGE SPACE
        LI   R2,32          32 BYTES TO GET
        BLWP @VMBR          READ COLOR TABLE INTO STORAGE
        MOV  @>8370,R0      GET VDP ADDR FROM >8370
        LI   R1,ANYKEY+1    POINT AT STORAGE BUFFER
        LI   R2,6           SIX BYTES TO READ
        BLWP @VMBR          READ THOSE INTO BUFFER
        LI   R0,>800        POINT AT CHARACTER TABLE
        LI   R1,CHRTBL      AND AT BUFFER STORAGE
        LI   R2,256*8       256 CHARACTER DEFINITIONS
        BLWP @VMBR          STASH CHARACTER DEFS
        BL   @SETBM         BIT-MAP MODE
CIRCLE  LI   R0,10          RADIUS 10
        LI   R8,95          CENTER DOT-ROW
        LI   R7,127         CENTER DOT-COLUMN
        CLR  R9             BLACK ON WHITE
        MOV  R7,@SAVR7      STASH CENTER COLUMN
        MOV  R8,@SAVR8      STASH CENTER ROW
CIRCLP  MOV  R0,@SAVR0      STASH RADIUS
        BL   @BCIRC         DRAW A CIRCLE
        MOV  @SAVR0,R0      GET RADIUS BACK
        MOV  @SAVR7,R7      GET CENTER COL
        MOV  @SAVR8,R8      GET CENTER ROW
        AI   R0,10          ADD 10 TO RADIUS
        CI   R0,160         COMPARE TO 160
        JLT  CIRCLP         IF LESS, REPEAT
KEYLOO  BLWP @KSCAN         SCAN KEYBOARD
        LIMI 2              INTS ON
        LIMI 0              INTS OFF
        CB   @ANYKEY,@>837C KEY PRESSED?
        JNE  KEYLOO         IF NOT, REPEAT
        BL   @SETGM         SET GRAPHICS MODE
EXIT    MOV  @>8370,R0      GET BACK >8370 ADDRESS
        LI   R1,ANYKEY+1    POINT AT BUFFER STORAGE
        LI   R2,6           SIX BYTES
```

Continued on page 20

Continued from page 19

```
        BLWP @VMBW       WRITE THOSE BACK TO VDP
        LWPI >83E0       LOAD GPL WORKSPACE
        B    @>6A        RETURN TO GPL INTERPRETER
*
*
* SUBROUTINES FOR HANDLING BIT-MAP
* OPERATIONS AND TRANSITIONS
*
* FOLLOWING SECTION SETS COMPUTER INTO BIT-MAP MODE
*
SETBM   LI   R0,>1A0     SET TO BLANK
        BLWP @VWTR       BLANK OUT SCREEN
        LI   R0,>206     SET TO WRITE VDP REGISTER 2
        BLWP @VWTR       SIT TO >1800  (SCREEN IMAGE TABLE)
        LI   R0,>403     SET TO WRITE TO VDP REG. 4
        BLWP @VWTR       PDT TO >0000  (PATTERN DESCRIPTOR TABLE)
        LI   R0,>3FF     SET TO WRITE TO VDP REG 3
        BLWP @VWTR       CT TO >2000   (COLOR TABLE)
        LI   R0,>607     SET TO WRITE VDP REG 6
        BLWP @VWTR       Sprite descritor table to >3800
        LI   R0,>570     SET TO WRITE VDP REG 7
        BLWP @VWTR       Sprite atribute list to >3800
        LI   R0,>58      INITIALIZE SCREEN IMAGE TABLE (SIT) (AT >1800)
        MOVB R0,@>8C02   WRITE LOW BYTE VDP ADDRESS
        SWPB R0          SWAP R0
        MOVB R0,@>8C02   WRITE HIGH BYTE VDP ADDRESS
        LI   R0,3        THREE TABLES OF 256 BYTES EACH
        CLR  R1          START WITH ZERO
SIT     MOVB R1,@>8C00   WRITE TO VDP (SELF-INCREMENTING)
        AI   R1,>100     ADD 1 TO HIGH BYTE R1
        JNE  SIT         IF NOT ZERO, REPEAT
        DEC  R0          ELSE DEC COUNT
        JNE  SIT         IF NOT ZERO, REPEAT
        LI   R0,>60      INIT COLOR TABLE  (CT) AT >2000
        MOVB R0,@>8C02   WRITE LOW BYTE OF ADDRESS
        SWPB R0          SWAP R0
        MOVB R0,@>8C02   WRITE HIGH BYTE OF ADDRESS
        LI   R0,>1800    >1800 BYTES TO WRITE
        LI   R1,>1F00    COLORS ALL BLACK ON WHITE
CT      MOVB R1,@>8C00   WRITE ONE BYTE
        DEC  R0          DEC COUNT
```

```
        JNE  CT          IF NOT ZERO, REPEAT
CPDT    LI   R0,>40      CLEAR PATTERN DESCRIPTOR TABLE (PDT) AT >0000
        MOVB R0,@>8C02   WRITE LOW BYTE ADDR
        SWPB R0          SWAP
        MOVB R0,@>8C02   WRITE HIGH BYTE ADDRESS
        LI   R0,>1800    >1800 BYTES TO WRITE
        CLR  R1          ALL ZEROS
PDT     MOVB R1,@>8C00   WRITE ONE
        DEC  R0          DEC COUNT
        JNE  PDT         IF NOT ZERO, REPEAT
        LI   R0,2        SET R0 TO WRITE 2 TO VDP REGISTER ZERO
        BLWP @VWTR       SET TO M3 MODE (BIT MAP)
        LI   R0,>1E0     UNBLANK
        BLWP @VWTR       WRITE THAT
        RT
*
* FOLLOWING SETS COMPUTER BACK TO NORMAL GRAPHICS MODE
*
SETGM   LI   R0,>1A0     SET TO WRITE VDP REG 1 (BLANK SCREEN)
        BLWP @VWTR       WRITE
        LI   R0,>200     SET TO WRITE VDP REG 2
        BLWP @VWTR       WRITE
        LI   R0,>401     SET TO WRITE VDP REG 4
        BLWP @VWTR       WRITE
        LI   R0,>30E     VDP REG 3
        BLWP @VWTR       WRITE
        LI   R0,>600     VDP REG 6
        BLWP @VWTR       WRITE
        LI   R0,>506     VDP REG 5
        BLWP @VWTR       WRITE
        LI   R0,>380     POINT AT COLOR TABLE
        LI   R1,SAVCLR   AND AT SAVED COLOR DATA
        LI   R2,32       32 BYTES
        BLWP @VMBW       WRITE THE COLOR TABLE BACK
        LI   R0,>800     POINT AT GRAPHICS CHAR TABLE
        LI   R1,CHRTBL   AND AT STORED CHARACTER DATA
        LI   R2,256*8    256 CHARACTERS
        BLWP @VMBW       WRITE CHARACTER DEFS BACK
        LI   R2,768      768 BYTES
        LI   R1,>2000    SPACE CHAR
        CLR  R0          ZERO IN R0
```

Continued on page 22

THE ART OF ASSEMBLY Part 66

Continued from page 21

```
        BLWP @VWTR         CANCEL BIT MAP
        MOVB R0,@>837A     NO SPRITE MOTION
CLSLOP  BLWP @VSBW         WRITE A SPACE
        INC  R0            NEXT ADDR
        DEC  R2            DEC COUNT
        JNE  CLSLOP        RPT IF NOT 0
        LI   R1,>D000      "DELETE" SPRITE #0
        BLWP @VSBW         BY VDP WRITE
        LI   R0,>1E0       GRAPHICS MODE
        BLWP @VWTR         UNBLANK SCREEN
        RT                 RETURN
*
* Bresenham's Circle Algorithm
* in TI Assembly Language
* on entry, R8=Y POSITION OF CENTER
*           R7=X POSITION OF CENTER
*           R0=RADIUS
* WITH THANKS TO PHIL VAN NORDSTRAND
*
BCIRC  MOV  R11,R13        SAVE RETURN ADDR
       MOV  R8,@CY         SAVE CENTER Y
       MOV  R7,@CX         SAVE CENTER X
       MOV  R0,@RADIUS     SAVE RADIUS
       MOV  R0,@DELYP      MAKE INITIAL TOP DELY=RADIUS
       NEG  R0             R0=-R0
       MOV  R0,@DELYM      MAKE INITIAL BOTTOM DELY=-RADIUS
       CLR  @DELXP         INITIAL DELXPLUS = 0
       CLR  @DELXM         INITIAL DELXMINUS = 0
       SLA  R0,1           DOUBLE R0 (R0=-2*RADIUS)
       AI   R0,3           ADD 3
       MOV  R0,@PARAM      INITIAL PARAM = 3 - 2*RADIUS
*
* FIRST LOOP DOES QUARTER CIRCLES AT THE
* TOP AND BOTTOM
*
PLACE  C    @DELXP,@DELYP CHECK DELTA X VS DELTA Y
       JGT  HALF2          IF GREATER, WE'RE DONE
       MOV  @DELYP,R8      GET TOP DELTA Y
       A    @CY,R8         ADD CENTER Y COORDINATE
       MOV  @DELXP,R7      GET POS DELTA X
       A    @CX,R7         ADD CENTER X COORDINATE
       BL   @CPLM          PLOT ONE POINT
       MOV  @DELXM,R7      GET NEGATIVE DELTA X
       A    @CX,R7         ADD CENTER X
```

THE ART OF ASSEMBLY Part 66

```
       BL   @CPLM          PLOT LEFT POINT
       MOV  @DELYM,R8      GET BOTTOM DELTA Y
       A    @CY,R8         ADD CENTER Y
       BL   @CPLM          PLOT BOTTOM LEFT
       MOV  @DELXP,R7      GET POS DELTA X
       A    @CX,R7         ADD CENTER X
       BL   @CPLM          PLOT BOTTOM RIGHT
       MOV  @PARAM,R0      GET CURRENT PARAMETER
       JLT  ADJP           IF LESS THAN ZERO, JUMP
       MOV  @DELXP,R3      GET POS DELTA X
       S    @DELYP,R3      SUBTRACT POS DELTA Y
       SLA  R3,2           MULTIPLY RESULT BY 4
       A    R3,R0          ADD TO PREVIOUS PARAM
       AI   R0,10          THEN ADD 10
       MOV  R0,@PARAM      REPLACE PARAMETER WITH P+4(DELX-
DELY)+10
       DEC  @DELYP         DEC POS DELY
       INC  @DELYM         INC NEG DELY
       JMP  ADDX           THEN JUMP
*
* CASE FOR PARAM <0
*
ADJP   MOV  @DELXP,R3      GET POS DELX
       SLA  R3,2           MULTIPLY BY 4
       A    R3,R0          ADD TO PARAM
       AI   R0,6           ADD 6 TO RESULT
       MOV  R0,@PARAM      REPLACE PARAM WITH P+4*DELX+6
ADDX   INC  @DELXP         INC POS DELX
       DEC  @DELXM         DEC NEG DELX
       JMP  PLACE          DO ANOTHER SET OF POINTS
*
* HALF2 DOES THE QUARTER CIRCLES ON THE SIDES
* BY REPEATING THE ALGORITHM WITH X AND Y INTERCHANGED
*

HALF2  MOV  @RADIUS,R0     GET BACK RADIUS
       MOV  R0,@DELXP      MAKE INITIAL POS DELX=RADIUS
       NEG  R0             R0=-R0
       MOV  R0,@DELXM      MAKE INITIAL NEG DELX=-RADIUS
       CLR  @DELYP         INITIAL DELYPLUS = 0
       CLR  @DELYM         INITIAL DELYMINUS = 0
       SLA  R0,1           DOUBLE R0 (R0=-2*RADIUS)
       AI   R0,3           ADD 3
       MOV  R0,@PARAM      INITIAL PARAM = 3 - 2*RADIUS
PLACE2 C    @DELYP,@DELXP CHECK DELTA Y VS DELTA X
```

Continued on page 24

THE ART OF ASSEMBLY Part 66

Continued from page 23

```
        JGT   CIRCX         IF GREATER, WE'RE DONE
        MOV   @DELYP,R8     GET POS DELTA Y
        A     @CY,R8        ADD CENTER Y COORDINATE
        MOV   @DELXP,R7     GET POS DELTA X
        A     @CX,R7        ADD CENTER X COORDINATE
        BL    @CPLM         PLOT UPPER RIGHT
        MOV   @DELYM,R8     GET NEGATIVE DELTA Y
        A     @CY,R8        ADD CENTER Y
        BL    @CPLM         PLOT LOWER RIGHT
        MOV   @DELXM,R7     GET LEFT DELTA X
        A     @CX,R7        ADD CENTER X
        BL    @CPLM         PLOT LOWER LEFT
        MOV   @DELYP,R8     GET POS DELTA Y
        A     @CY,R8        ADD CENTER Y
        BL    @CPLM         PLOT UPPER LEFT
        MOV   @PARAM,R0     GET CURRENT PARAMETER
        JLT   ADJP2         IF LESS THAN ZERO, JUMP
        MOV   @DELYP,R3     GET POS DELTA Y
        S     @DELXP,R3     SUBTRACT POS DELTA X
        SLA   R3,2          MULTIPLY RESULT BY 4
        A     R3,R0         ADD TO PREVIOUS PARAM
        AI    R0,10         THEN ADD 10
        MOV   R0,@PARAM     REPLACE PARAMETER WITH P+4(DELY-DELX)+10
        DEC   @DELXP        DEC POS DELX
        INC   @DELXM        INC NEG DELX
        JMP   ADDY          THEN JUMP
*
* CASE FOR PARAM <0
*
ADJP2   MOV   @DELYP,R3     GET POS DELY
        SLA   R3,2          MULTIPLY BY 4
        A     R3,R0         ADD TO PARAM
        AI    R0,6          ADD 6 TO RESULT
        MOV   R0,@PARAM     REPLACE PARAM WITH P+4*DELY+6
ADDY    INC   @DELYP        INC POS DELY
        DEC   @DELYM        DEC NEG DELY
        JMP   PLACE2        DO ANOTHER SET OF POINTS
CIRCX   B     *R13          RETURN TO CALLER
*
* FOLLOWING CHECKS SCREEN LIMITS BEFORE
* ALLOWING A POINT TO BE PLOTTED ON SCREEN
*
CPLM    MOV   R7,R7         CHECK COL FOR <0
```

THE ART OF ASSEMBLY Part 66

```
        JLT   NPLEX         IF <0, JUMP
        MOV   R8,R8         CHECK ROW FOR <0
        JLT   NPLEX         IF <0, JUMP
        CI    R7,255        CHECK UPPER COL LIMIT
        JGT   NPLEX         IF GREATER, JUMP
        CI    R8,191        CHECK UPPER ROW LIMIT
        JGT   NPLEX         IF GREATER, JUMP
        JMP   PLOT          ELSE PLOT THE POINT
NPLEX   RT                  RETURN
*
* FOLLOWING WRITES ONE PIXEL TO SCREEN AT LOCATION POINTED BY
* R8 (DOT ROW) AND R7 (DOT COLUMN)
*
PLOT    MOV   R7,R3         MOVE DOT COLUMN TO R3
        MOV   R8,R4         AND DOT ROW TO R4
        MOV   R4,R5         DOT ROW ALSO IN R5
        ANDI  R5,7          R5 HAS DOT ROW MODULO 8
        SZC   R5,R4         SO DOES R4
        SLA   R4,5          MULTIPLY R4 BY 32
        A     R5,R4         ADD R5, SO R4 HAS DR MOD. 8 * 32 + DR MOD 8
        MOV   R3,R0         MOVE DOT COL TO R0
        ANDI  R0,>FFF8      R0 HAS DC - DC MOD 8
        S     R0,R3         R3 HAS DC MOD 8
        A     R4,R0         ADD R4
        SWPB  R0            SWAP BYTES
        MOVB  R0,@>8C02     WRITE LOW ADDRESS BYTE
        SWPB  R0            SWAP
        MOVB  R0,@>8C02     WRITE HIGH ADDRESS BYTE
        NOP                 WASTE TIME
        MOVB  @>8800,R1     READ THE BYTE
        SOCB  @M(R3),R1     OVERLAY MASK FROM TABLE M
        ORI   R0,>4000      SET THE 4000 BIT IN R0
        SWPB  R0            SWAP
        MOVB  R0,@>8C02     WRITE LOW BYTE OF ADDRESS
        SWPB  R0            SWAP
        MOVB  R0,@>8C02     WRITE HIGH BYTE OF ADDRESS
        NOP                 WASTE TIME
        MOVB  R1,@>8C00     WRITE MODIFIED BYTE BACK TO VDP
        MOV   R9,R9         IS COLOR TO BE SET?
        JEQ   PLOTX         IF NOT, JUMP AHEAD
        ANDI  R0,>3FFF      STRIP OFF "4" FROM R0
        AI    R0,>2000      ADD >2000 TO POINT AT COLOR TABLE ENTRY
```

Continued on page 26

THE ART OF ASSEMBLY Part 66

Continued from page 25

```
        BLWP @VSBR          READ THAT BYTE INTO R1
        MOVB R1,R2          MOVE THE BYTE TO R2
        ANDI R2,>F000       STRIP ALL BUT LEFT NYBBLE
        CB   R2,R9          COMPARE TO LEFT BYTE R9
        JEQ  PLOTX          IF EQUAL, COLOR ALREADY SET
        ANDI R1,>0F00       ELSE STRIP OFF LEFT NYBBLE R1
        AB   R9,R1          REPLACE WITH LEFT NYBBLE R9
        BLWP @VSBW          THEN WRITE COLOR BYTE BACK
PLOTX   RT                  RETURN
*
*
* DATA SECTION
*
WS      BSS  >20            OUR WORKSPACE
M       DATA >8040,>2010,>0804,>0201  MASK DATA
CX      DATA 0              CENTER X POSITION
CY      DATA 0              CENTER Y POSITION
RADIUS  DATA 0              CIRCLE RADIUS
DELXP   DATA 0              POSITIVE DELTA X
DELYP   DATA 0              POSITIVE DELTA Y
DELYM   DATA 0              NEGATIVE DELTA Y
DELXM   DATA 0              NEGATIVE DELTA X
PARAM   DATA 0              PARAMETER
SAVR0   DATA 0              STORAGE FOR R0
SAVR7   DATA 0              STORAGE FOR R7
SAVR8   DATA 0              STORAGE FOR R8
ANYKEY  BYTE >20            COMPARISON BYTE FOR KEYSTROKE
        BSS  6              STORAGE FOR DSR DATA FROM VDP RAM
SAVCLR  BSS  32             STORAGE FOR GRAPHICS COLOR TABLE
CHRTBL  BSS  256*8          STORAGE FOR GRAPHICS CHARACTER
DEFINITIONS
        END
```

CIRCXB

Following is an Extended BASIC program called CIRCXB to illustrate the circle algorithm:

```
10 CALL CLEAR :: INPUT "RADI
US ":RADIUS :: RADIUS=INT(RA
DIUS):: IF RADIUS<0 OR RADIU
S>11 THEN 10
20 CALL CLEAR :: CX=14 :: CY
=12
30 P=3-2*RADIUS :: DELYP=RAD
US :: DELYM=-RADIUS :: DELX
P,DELXM=0
40 IF DELXP>DELYP THEN 100
50 DISPLAY AT(CY+DELYP,CX+DE
LXP):CHR$(30);:: DISPLAY AT(
CY+DELYP,CX+DELXM):CHR$(30);
60 DISPLAY AT(CY+DELYM,CX+DE
LXP):CHR$(30);:: DISPLAY AT(
```

```
CY+DELYM,CX+DELXM):CHR$(30);
70 IF P<0 THEN P=P+(4*DELXP)
+6 :: GOTO 90
80 P=P+(4*(DELXP-DELYP))+10
 :: DELYP=DELYP-1 ::
DELYM=DE
LYM+1
90 DELXP=DELXP+1 :: DELXM=DE
LXM-1 :: GOTO 40
100 P=3-2*RADIUS :: DELXP=RA
DIUS :: DELXM=-RADIUS :: DEL
 YP,DELYM=0
110 IF DELYP>DELXP THEN 170
120 DISPLAY AT(CY+DELYP,CX+D
ELXP):CHR$(30);:: DISPLAY AT
(CY+DELYP,CX+DELXM):CHR$(30)
 ;
130 DISPLAY AT(CY+DELYM,CX+D
ELXP):CHR$(30);:: DISPLAY AT
(CY+DELYM,CX+DELXM):CHR$(30)
 ;
140 IF P<0 THEN P=P+(4*DELYP
)+6 :: GOTO 160
150 P=P+(4*(DELYP-DELXP))+10
 :: DELXP=DELXP-1 ::
DELXM=DE
LXM+1
160 DELYP=DELYP+1 :: DELYM=D
ELYM-1 :: GOTO 110
170 DISPLAY AT(24,7):"PRESS
R TO REPEAT";
180 CALL KEY(0,K,S):: IF S<1
THEN 180 ELSE IF K=82 OR K=
114 THEN 10
```

HARDWARE RS-232

TI RS-232 configuration

BY BOB CARMANY

As with all hardware modifications, continue at your own risk. If you blow something up, tough! This author doesn't warrant that this modification will fulfill the needs of your system nor is any liability for this project assumed by the author.

Tired of the Y-cable hanging out the back of your RS-232 card? Or, do you have a second parallel printer or additional serial device that you would like to have attached to your system. Maybe, like me, you have "maxxed out" the capabilities of a single RS-232 card. You can add a second card to your system with very little effort — even if you have a TI RS-232 card. All you need is a second card, a small Phillips screwdriver, a desoldering iron, a soldering iron, some needle-nosed pliers and a bit of solder.

The TI99/4A is capable of supporting two RS-232 cards at one time. The primary card occupies CRU >1300 and is designated RS232 and PIO. With a Y-cable, the serial port can be split into RS232/1 and RS232/2. The secondary RS-232 card occupies CRU >1500 and is designated RS232/3 and PIO/2. With a Y-cable, the secondary port can be split into RS232/3 and RS232/4.

If you read the manual that came with the card, it says that the card can be sent to TI in Lubbock for free, but "PERMANENT MODIFICATION," as the RS-232 manual's addendum states. *(Modifications are now made by Cecure Electronics. — Ed.)*

Continued on page 28

Continued from page 27

The required modification is hardly permanent except for the fact that it does require that a resistor on the card be unsoldered and relocated to the empty set of holes immediately below it. First, remove the two spring clips on the top edge of the clamshell case. Then, take out the four screws at each of the corners. The clamshell case should come apart letting you access the circuit board. Find the chip marked U15 and the resistor R5 immediately below it. With the desoldering iron carefully remove the solder holding each of the two legs in place and remove it with the pliers making careful note which end is which. Move it down to the next set of holes marked PTH1 on the board. Solder the resistor in place and the modification is complete.

To test your work, put the card in an empty P-Box slot observing all the usual precautions. You should be able to access the card (the light will come on) as RS232/3 or PIO/2. The easiest way is to try to send a file from F'WEB or TI-Writer. Since you don't have a device attached to it, the light will come on and nothing else will happen but you can check the address that way.

Just remember, in the primary mode, the ports are addressed as RS232 if a single port is used or RS232/1 and RS232/2 if the port is split by use of a Y-cable. The parallel port is addressed as PIO or PIO/1 (either designation can be used).

In the secondary mode, the ports are addressed as RS232/3 if a single port is used or RS232/3 and RS232/4 if a Y-cable is used to split the second port. The parallel port is addressed as PIO/2.

This procedure can be infinitely reversed although a strapping arrangement similar to the CorComp card would provide better protection for the circuit components if frequent changes are anticipated. (Note: A switch would be even better.)

PR-BASE

Step-by-step approach to database output

BY MARY PHILLIPS

This article appeared in the March 1996 issue of the Ozark 99er News. — *Ed.*

In the February newsletter for the Cleveland Area Users Groups, help was asked for output for PR-Base to 1) print to disk rather than printer and 2) change from D/V 132 to D/V 80. Here is what I do.

At the first screen:

* DATABASE MANAGER *
PRESS:
1 FOR DATABASE CREATION

2 DATA MANAGEMENT
3 EXIT

Press 1 and Enter.

Press Enter at Creation title page.

CREATION MENU
PRESS:
1 TO Select Data Disk Drive
2 Format Data Diskette
3 Design Data Screen
4 Design Tabular Reports
5 Design Mailing Labels
6 Set Printer Codes
7 Set System Options
8 Exit

Press 1 and select any drive 1-5. *This must be done each time you load the Creation Menu.*

Press 2 to format your *data disks only* as PR-Base formats in a special way and a SS disk cataloged with DM1000 shows DSK2.eeeeeeeee and Used=0265 Free=8588. That totals 8,853 sectors? A data disk formatted with DM1000 onto which headers and records have been copied shows a total of 1,869 sectors? Either will work with PR-Base but neither will build a directory of files. Disks formatted with PR-Base will not accept regular programs or text files, but can be named with DM1000 so you know what they are.

Press 3 to design or edit your screen, but note that the key repeat function doesn't in PR-Base. A chart of key presses for lines and boxes appears at the bottom of the screen. Use brackets [] (Fctn R and Fctn T) for information to be in all capital letters, and use braces { } (Fctn F and Fctn G) for mixed characters and numerals. Press Fctn 6 and enter the database filename: UG96 and press Enter, PIO. or DSKn.filename (Enter). (If you want to print this D/V 80 file to disk, you may remove the program disk and put in a regularly formatted disk. *You cannot print to the data disk.*) Print this screen? (Y/N), and press PROCD (Fctn 6) to write to disk. The number of fields and number of characters is listed. You may now fill in the blanks in each record.

Press 4 to design or edit 1 to 5 tabular reports based on the information in the database.

DESIGN TABULAR REPORTS

Report Number:	(1-5)	2
Number of Columns	(80/132) 80	
Number of Lines	(1-6)	2

Report Title:
ALTERNATE 1996 ROSTER

Continued on page 30

Continued from page 29

Enter Control Code ASCII Values

27 78 6 0 0 0

Three spaces are allowed for each of six ASCII values for printer control codes. No. 27 (Escape) means Special Character Mode; 78 means Skip perforation; and 6 means give 6 blank lines before the skip. You must put in one space before 27 and 78, and two spaces in front of the 6 and each zero or you may want to put in other codes. You must also press Enter after each number to tab to the next location. If you have selected 132 characters per line (condensed print), put No. 15 as the first number.

On the Report Format Design page, PR-Base lists the numerical locations of the fields as you placed them when you designed your data screen.

At the prompt Log Device, if you want to print out the screen, you must enter PIO or whatever your printer address is. If you're not going to print the screen, you may press Enter.

DESIGN TABULAR REPORT ALTERNATE 1996 ROSTER

PR-Base designs a report for you based on the number of lines you told it you want and asks: Print this Screen? (Y/N) If you select Y, you are asked for Log Device again and you give it your printer address. When the screen has been printed or if you pressed N, the cursor jumps to the first screen location number for editing if you wish to do so. Press PROCD (Fctn 6) when finished or to go on.

Number of lines used: 3
Number of lines desired 3

In my report that is condensed print, two lines are used, and I type 1 for Number of lines desired and it works. Two spaces are allowed for the numbers, and, again, it is necessary to space once before a single digit.

Enter Column Header Line:

This gets tricky because there are only 40 columns on the screen and it becomes necessary to count over to where you want your header to start if it's any place other than column 1. Or, type your header and then insert spaces to where you want it to start. Press Enter and you're returned to the menu.

In most places, you can return to the Create Menu by pressing Fctn 9, but once you're in the design phase, you have to go on through in order to get out as Fctn 9 is temporarily disabled.

Press 6 to design or edit mailing labels. Number of lines per label: 6 is what I use for standard 15/16th-inch address labels.

Report Format Design is the same as we saw when we designed a report.

DESIGN MAILING LABELS

As in Design Tabular Report, PR-Base sets up a label for you in three rows. You may print the screen if you choose, and you have the opportunity to edit it. Press PROCD (Fctn 6) when finished.

Number of lines used: 3
Number of lines desired 6

Pressing 6 to Set Printer Codes allows you to enter up to five code sequences to which you may refer as needed.

Press 7 to Set System Options. *(You must do this!)*

Database Name: DSK2.96UG
Printer Name: PIO.
Number of Sides on Data Disks: 1
Left label starting column: 02
Right label starting column: 00

When I used two-across labels, the right label starting column was 45, but now I use single-row labels.

Press 8, Exit the Creation Program.

Reboot PR-Base. Press 2, DATA MANAGEMENT DSK2? (Type the data disk number and a question mark and the database will be loaded — you don't have to remember what you called it.)

After data have been entered in records, press S for Sort in order to print either a report or labels. The number of total records in the database is shown on the upper right-hand corner.

To print label information to a D/V 80 disk file, Press O for Output Device.

Data Disk Drive: 2 Press Enter.

Output Device: Change PIO. to DSK1.96UG. Unless you want the file on your program disk, remove the program disk, but leave in the data disk. I put my D/V 80 file on my RAMdisk so I can use it with Holiday Tips to put seasonal graphics on newsletter labels.

Pressing H (Help Screen) in the Command Mode brings up a list of all the commands and key presses. This is printed out in the docs, and I made a copy which I keep taped to my PE-box as well as on a flipstrip.

On the program disk is an Extended BASIC utility program, PRBUTIL/BAS and BRBUTL/DOC which I use each March to make a new data base of paid/active members. The menu is:

PRESS	**CHOICE**
1	Copy database header
2	Copy a group of records
3	Copy a single record
4	Search & select records
5	Sort & rewrite to copy
6	Configure drives
7	Exit program

A BOOT! menu could be used to put both PRBASE and PRBUTL/BAS on a menu.

Pointers - What are they?

BY VERN JENSEN

Last issue we mentioned that there are two ways of returning values from a function: the first is by using the return statement, and the second is by using pointers. But what is a pointer? It is simply a variable that contains the address of another variable's location in memory; thus the name "pointer", since it "points to" another variable. Pointers must be declared differently than other variables. To declare a variable as a pointer, you must put an asterisk in front of the variable when declaring it:

```
char *myPtr;
```

The asterisk makes only the variable it is in front of a pointer, so in this example, the variable intPtr becomes a pointer, while the other variables do not:

```
int key, *intPtr, status;
```

When declaring a pointer, you determine what type of variable the pointer can point to, whether char or int. In the examples above, myPtr can point to other char variables, and intPtr can point to int variables. To assign the address of a variable to a pointer, you use the & operator, which gives the address of a variable:

```
intPtr = &status;
```

This will assign the address of the status variable to intPtr. Keep in mind that this is quite different than assigning the contents of the status variable to intPtr. If we wanted to do that, we would simply use "intPtr = status". Once we have the address of a variable stored in a pointer, we can access that variable through the pointer by using the dereferencing operator "*":

```
myNum = *intPtr;   /* "myNum = status" */
```

This will assign to myNum whatever intPtr pointers to. In this case, it points to the variable status, so the contents of status will be assigned to myNum. Things can go in the other direction as well. For instance, this assigns the contents of myNum to whatever variable intPtr points to:

```
*intPtr = myNum;   /* "status = myNum" */
```

Since intPtr points to status, *intPtr can appear anywhere status could, so statements such as these are possible:

```
*intPtr = *intPtr * 5; /* Multiply status by 5 */
(*intPtr)++;           /* Increment status */
```

The parentheses are necessary in the last example because without them, intPtr would be incremented instead of what it points to, because operators like ++ and * associate right to left. And as you might expect, you can assign the contents of one pointer to another, so if otherP is also an int pointer, this would assign the contents of intPtr to otherP, making otherP point to status as well:

```
otherP = intPtr;
```

FUNCTIONS AND POINTERS

"So what's all the fuss about?", you may wonder. "Sure, pointers are cool, but how do they help me?" Well for starters, you can pass the address of a variable to a function. This allows the function to modify the original copy of that variable. Below is an example of a function that accepts pointers to two variables and swaps the contents of the original variables. Notice that the function parameters are declared as pointers:

```
void  Swap(a, b)
int *a, *b;
{
      int temp;

      temp = *a; /* Save contents of a */
      *a = *b;   /* assign contents of b to a */
      *b = temp; / *assign old a value to b */
}
```

To call this function, you would use

```
int myA, myB;
...
Swap(&myA, &myB);
```

and the contents of myA and myB would be swapped. Another way to call this would be like so:

```
int   *aPtr, *bPtr;

aPtr = &myA;
bPtr = &myB;
Swap(aPtr, bPtr);
```

In both cases, the addresses of myA and myB are passed to the function. The function can then dereference the pointers to access the variables they are pointing to, allowing the function to change values it otherwise would be unable to change. But how about a "real life" example? Here's something you'll likely use every time you write a program:

```
int char, status;

char = Key(0, &status);
```

Here the address of the status variable is passed to the Key function (part of the GRF1 library) so the function can place the current status in that variable. In addition, this function also makes use of the ability to return a value with the return statement, and returns the key's character code that way. Another example of pointers at work is the Joyst function, which returns x, y, and status values:

Continued on page 34

BEGINNING c99

Continued from page 33

```
int s, x, y;

s = Joyst(0, &x, &y);
```

POINTERS AND ARRAYS

Pointers and arrays are closely related. In fact, any operation that can be performed by array subscripting can also be done with pointers. We have seen that you can assign the address of a variable to a pointer. You can also assign the address of an array, or any element of the array, to a pointer. Here's an example:

```
char a, *myPtr, array[50];

myPtr = &array[0]; /* myPtr points to array element 0
*/
a = *myPtr;          /* a = array[0] */
*myPtr = 7;          /* array[0] = 7 */
```

First we assign the address of the first element of the array to the pointer, then we dereference the pointer by using the "*" (dereferencing) operator, so that variable a is given the contents of whatever myPtr points to. Finally, we use the dereferencing operator again to assign 7 to whatever myPtr points to. If we didn't use the dereferencing operator, myPtr itself would have 7 assigned to it, instead of array[0]. If you then tried to use myPtr as a pointer, you would run into problems, since you would be accessing whatever is kept at memory location 7, which certainly isn't the address of one of your variables.

There is another way to get the address of an array, and that is by simply using the array's name without the "&" operator and without subscripting. For instance, this would set myPtr to point to array element 0, just as the code above does:

```
myPtr = array;
```

This is because the name of an array contains the address of the first element of that array. When you add a subscript, such as "array[5]", the C compiler uses the address contained in the array name to access the desired element of the array. Subscripting is not limited to arrays, however. You can also subscript pointers:

```
myPtr[5];
```

This tells the compiler to access the fifth object after what myPtr points to, which in this case is array[5]. So by assigning the address of array to myPtr, we can use myPtr just as if it is the array. This can be very useful when calling functions, since you can pass the address of your array to the function, and then that function can access any element of the array. Keep in mind that while you can get the address of an array using the array's name, you can't change it. So this wouldn't work:

```
array = myPtr;
```

BEGINNING c99

However, you can change the value in myPtr. When you add or subtract a value from a pointer, it changes what the pointer points to. For instance, adding 1 to a pointer makes it point to the very next object in memory. If the pointer is a Char, the compiler is smart enough to move it up one byte in memory (since a Char takes up one byte). If the pointer is an Int, then adding one will move it up two bytes in memory (an Int takes up two bytes). This allows you to increment a pointer that points to an array, and it will point to the next element, regardless of whether the array is of type Char or Int.

Here's an example that sets a to the value contained in array[5]. First we add 5 to myPtr, then we dereference the new address:

```
a = *(myPtr+5);  /* Same as a = array[5] */
```

Or if you want to change the pointer itself, you could do this:

```
myPtr = myPtr + 5;  /* myPtr now points to element 5
*/
```

Now accessing myPtr as an array will mean that myPtr[0] is the same as array[5], myPtr[1] is the same as array[6], etc. This trick could be useful when passing the address of an array to a function if you wanted the function to think a particular element of the array was actually the beginning of the array. (You could pass something like "&array[5]" as the parameter to your function.)

ENOUGH!

Enough confusing nonsense for right now. Here's an example function that will help clear things up. It scans an array for the requested ascii value, and returns the element of the array in which the value was found. For the sake of simplicity, it is assumed that the value is contained in the array, so the array's boundaries will not be exceeded:

```
ScanArray(myChar, myArray)
char myChar, *myArray;
{
  int n;

  for (n=0; myArray[n] != myChar; n++)
    ;

  return n;
}
```

Here we have an example of an "empty" loop. The for loop executes no statements, because all the necessary statements are provided in the loop itself. (Just try to do that in Extended Basic!) The function receives the address of the array in its pointer (myArray), and uses the pointer as if it were an array. It starts with element 0 and increments the current element until myChar is found, at which point the loop ends and the current element is returned to the caller. The

Continued on page 36

BEGINNING c99

Continued from page 35

function above could also be written without any array subscripting, although it might be a little more confusing:

```
ScanArray(myChar, myArray)
char myChar, *myArray;
{
  int n;

  for (n=0; *myArray != myChar; n++, myArray++)
    ;

  return n;
}
```

The key to remember is that the statement "*myArray" gives the value that myArray points to, and the statement "myArray++" increments the myArray pointer so that it points to the next element. Normally you wouldn't write your code like this, since it is a little harder to understand, but this type of code does have its place. To call the ScanArray function, you might use something like this:

```
element = ScanArray('E', "HELLO");
```

The expression 'E' should be familiar - it is converted by the compiler into the corresponding ascii value (69). However, the "HELLO" text in quotes is new. This is called a string constant, meaning it's a string of char type data that can't be changed. When you use a string constant as a parameter to a function, the address of the string is passed to the function. The function will then see an array filled with these values:

```
[72][69][76][76][79][0]
```

As you can see, a string constant is represented in memory with ascii values for each character in the string, terminated by a 0 to indicate the end of the string. Each character in a string constant is only one byte (the size of a char), meaning that if you assign the address of a string constant to a pointer, the pointer must be a char pointer. You may be wondering by now if C has the equivalent of XB's strings. After all, while string constants are nice, you can't change them. The answer is that you use arrays to store the contents of your strings, and if your array is a char array, you can use a string constant to initialize it, creating an array which contains your string, but can be modified:

```
char myString[] = "Pretty neat, huh?";
```

Notice that I didn't put a number between the brackets of the array. That's because if you leave it out, the size of the array is calculated based on the size of the string you provide - it will have the same number of elements as the number of characters in the string plus one, to account for the "end of string" character, ascii value 0. If you do specify the number of elements in the array when

BEGINNING c99

initializing it with a string constant, it must be larger than the number of characters in your string. Extra elements not in your string will be filled with 0's. It is also possible to initialize an array with numbers:

```
int prime[] = {1,3,5,7,11,13,17};
```

Keep in mind that since a string constant is not used in this second example, the array will not be terminated with a zero - that's something you'll have to add if you want it. Array initializers like the ones described above will not work if the array is a local variable, since local variables come and go as the function is entered and exited. For the examples above to work, they'd have to be global arrays (meaning they are defined at the beginning of your program, not in a function). And unfortunately, you can't assign a string constant to an array that has already been created. For example, the following code will not work:

```
char myArray[5];
myArray[0] = "Hello";   /* Won't work. */
```

You can, however, assign the address of a string constant to a char pointer:

```
char *myPtr;
myPtr = "Hello";
```

In this example, myPtr will point to the address of the first character in the string, allowing you to access that string like a char array. However, you must remember that the string is still a constant - you can read those characters, but you can't change them. On the other hand, you can change the pointer so that it points to something else, although you would then lose the address of the "Hello" data.

A SILLY PROGRAM

To conclude our article is a program that demonstrates the topics we've just covered. The program is pretty pointless - it displays text on the screen both horizontally and vertically, but it does cover a number of important points. Most importantly, it gives you an example of how to use what we've covered in a real problem. I say this is important because I ran into quite a few compile errors when I tried to make it. If I had this much trouble making it, I can't imagine how much trouble a beginner would have trying to use what we've just covered if an example program wasn't provided!

This issue we're going to do something a little different - I'm going to walk through the source code, explaining each line.

```
char helloS[] = {72,69,76,76,79,0};
char introS[] = "This is an example of an array con-
taining text.";
char arrayS[] = "And this is another array";
```

Continued on page 38

BEGINNING c99

Continued from page 37

```
char *constP = "This is a pointer toa string con-
stant";
```

These lines demonstrate three different ways of saving text. The first uses numbers to initialize the array. This is a pretty stupid example, because a much easier way would be to use "helloS[] = "HELLO". However, I wanted to show how this is done, since there will be times when you'll want to initialize an array with numbers. The next two lines initialize two arrays with text, and the last line creates a char pointer that points to the first character of a string constant. Remember that this is quite different from an array, since you can't change the contents of the string constant.

```
main()
{
  int set, status;

  Grf1();
  Screen(2);

    /* White on black */
  for (set=0; set<16; set++)
    Color(set,16,1);
```

Next we initialize the GRF1 library, set the screen to black, and the text to white.

```
  Display(1,14,&helloS[0]);
  Display(2,5,introS);
```

Here we use two different methods of passing the address of the first element of an array to a function. Remember that "introS" is the same as "&introS[0]".

```
  VDisplay(5,8,"This is a string    constant");
  VDisplay(5,13,arrayS);
  VDisplay(5,18,constP);
```

After displaying our horizontal text, we call VDisplay to display our vertical text. The first call places a string constant directly into the function call. The address of the string constant will be passed to the function. (Note: no & operator is necessary when using string constants.) The next two functions display the contents of an array, and the contents of the string constant that constP points to. I know it may seem silly to use all these different methods in one program, but I just wanted to show each technique.

```
  do
    Key(0,&status);
  while (status != 1);
}
```

After putting the text on the screen, we wait for a new key to be pressed, allowing the user to read everything on the screen before the program quits.

BEGINNING c99

```
Display(myRow, myCol, string)
int myRow, myCol;
char *string;
{
  int n, chr;
  n = 0;
  chr = string[n];

  while (chr != 0)
  {

    HChar(myRow, myCol, chr, 1);
    myCol++;
    n++;
    chr = string[n];

      /* Bump to next row? */
    if (myCol > 28)
    {
      myCol = 3;
      myRow++;
    }
  }
}
```

Next we come to our Display function, which displays text horizontally across the screen, much like Extended Basic's Display At. We could have just called Locate(myRow,myCol) and then PutS(string), but I wanted to demonstrate how pointers are used, so I wrote a function that does everything from scratch. (For more information about the Locate and PutS functions, see the c99 manual. These functions are included as part of the CSUP library, and since CSUP must be included by all c99 programs, these functions can be used by any program.)

One of my first mistakes when making this function was to use char variables for myRow and myCol. When the program didn't work correctly, I looked up the documentation for HChar and found that it expects int variables for the row and column. This means it was reading two bytes of data, and I was only providing one. After fixing this, the text appeared in the correct place on the screen, but was all scrambled. I then discovered that HChar expects an int for the character it is drawing as well. I fixed this by defining the chr variable as an int, and copying each value from the char array into chr before passing it to

Continued on page 40

Continued from page 39

HChar. This is done by the line "chr = string[n]", and then chr is passed to HCHar, rather than string[n]. It is perfectly legal to assign a char to an int or an int to a char, although if you do the latter, you may get unexpected results if your int is larger than 255.

```
VDisplay(startRow, myCol, string)
int startRow, myCol;
char *string;
{
  int myRow, chr;

  myRow = startRow;
  chr = *string;

  while (chr != 0)
  {
    HChar(myRow, myCol, *string, 1);
    myRow++;
    string++;
    chr = *string;

    if (myRow > 24)
    {
      myRow = startRow;
      myCol++;
    }
  }
}
```

Finally we come to the VDisplay function. To keep you from getting bored, we did this differently than the other function. Rather than assigning string[0] to chr, we assign *string to chr, which does exactly the same thing. (Remember that *string access whatever string points to, which in this case is the first element of an array.) Then instead of incrementing n, we increment string, so it points to the next character in the array. This method may not be as easy to understand, but it's more efficient, since the statement "chr = *string" results in fewer assembly lines than "chr = string[n]". (The latter is actually converted into "chr = *(string+n)" before it is assembled.)

I hope this isn't too confusing. If you get stuck, just take a break and then read this article again, carefully going over anything you don't understand, and hopefully it will all fall into place. The concept of pointers may be hard to understand at first, but they are quite easy to use and very helpful once you

know how to use them.

TEXTFUN;C

```
/*************************/
/* TEXTFUN;C. SEP/OCT '97 */
/* ISSUE OF MICROPENDIUM  */
/* BY VERN JENSEN         */
/*************************/

#include "DSK2.GRF1;H"

char helloS[] = {72,69,76,76,79,0};
char introS[] = "This is an example of an array con-
taining text.";
char arrayS[] = "And this is another array";
char *constP = "This is a pointer toa string con-
stant";

main()
{
  int set, status;

  Grf1();
  Screen(2);

    /* White on black */
  for (set=0; set<16; set++)
    Color(set,16,1);

  Display(1,14,&helloS[0]);
  Display(2,5,introS);

  VDisplay(5,8,"This is a string    constant");
  VDisplay(5,13,arrayS);
  VDisplay(5,18,constP);

  do
    Key(0,&status);
  while (status != 1);
}
```

Continued on page 42

Continued from page 41

```
Display(myRow, myCol, string)
int myRow, myCol;
char *string;
{
  int n, chr;
  n = 0;
  chr = string[n];

  while (chr != 0)
  {

    HChar(myRow, myCol, chr, 1);
    myCol++;
    n++;
    chr = string[n];

      /* Bump to next row? */
    if (myCol > 28)
    {
      myCol = 3;
      myRow++;
    }
  }
}


VDisplay(startRow, myCol, string)
int startRow, myCol;
char *string;
{
  int myRow, chr;

  myRow = startRow;
  chr = *string;

  while (chr != 0)
  {
    HChar(myRow, myCol, *string, 1);
    myRow++;
```

```
    string++;
    chr = *string;

    if (myRow > 24)
    {
      myRow = startRow;
      myCol++;
    }
  }
}
```

Lima group to host 1998 MUG

The Lima User Group will again sponsor the all TI/Geneve Multi-User Group (MUG) Conference in 1998. It is likely that this will be the last MUG Conference the Lima group will be able to host. According to Charles Good, the Cleveland user groups will probably sponsor the event in 1999.

The event is scheduled for the Ohio State University Lima Campus, May 15 and 16. This is the weekend before Memorial Day weekend.

"I will soon start a web page for MUG 1998," Good said. The web page for the 1997 MUG conference (www.bright.net/~cgood/mug1997.html) will close in October.

COMPETITION COMPUTER (800)471-1600
350 MARCELLA WAY,MILLBRAE CA 94030
CARTRIDGE SALE! $19.95+SHIPPING
SHIPPING: $3 FOR UP TO 6 CARTS
ADD $1 FOR EACH ADDITIONAL CART
EXAMPLE: 7 CARTS=$4 SHIPPING

4A Flyer	**Escape**	**Barrage**	**Great Word Race**
Beyond Parsec	**Junkman Jr.**	**Black Hole**	**Midnight Mason**
Boxer	**Munchman II**	**Breakout**	**Protyper**
Burger Builder	**Star Runner**	**D Station**	**Strike 3**
D Station II	**TI Toad**		

ADDITIONAL CARTRIDGE SPECIALS
Qbert $9.95 Hangman $9.95 Amazeing $4.95 Alpiner $4.95
Car Wars Or Chisholm Trail $2.95 Adventure(disk+cart) $2.95
Hunt The Wumpus $2.95 Munchman Or Ti Invaders $2.95
Tombstone City/The Attack $2.95 Parsec $1.99 TE II $2.95
Over 240 cart titles in stock!! Send your want lists/trades ok!

New tricks from PC99, SCSI Cat, Textloader, Basic Builder, Extended BASIC V2.5 and V2.6

BY CHARLES GOOD

It is possible to input text from a text file directly into the Extended BASIC editor, just as if you had typed this text into Extended BASIC. The first three products I review this month help you do this. Using these products you can edit XB programs on a 99/4A or Geneve using any version of TI-Writer or you can edit your XB program on a PC word processor such as those that come with Windows 95. You can then automatically put the edited XB program text file source code into XB and have it run just as if you manually typed it in. You can also create a text batch file to run from command mode that can run a specific sequence of programs or that will do a CALL FILES(1) and NEW and then run a large memory image XB program. The possibilities are really interesting. RXB can do this, as can Super BASIC. However RXB requires a GRAM device or a Geneve, and the commercial program Super BASIC requires a hardware key that you plug into the cassette port. The first three software products I am reviewing this month allow users without special hardware to enter text files into XB.

PC99 (again). by CaDD Electronics

OK, I lied. You do need special hardware to run PC99, namely a 486 or highter PC. But you don't need any special TI hardware and you don't need any special TI software. Running PC99 as a Windows 95 DOS window allows you to copy and paste any text into the XB or TI BASIC editor running under PC99. I figured out how to do this after finishing my PC99 review published in the previous MICROpendium, so I need to discuss PC99 again.

You need to set up a shortcut to PC99 and put this shortcut on your Windows 95 desktop in a specific way. This procedure doesn't work with Windows 3.1. Using Windows 95's My Computer, find PC99L or PC99A on your computer's hard drive and, using the mouse, hold down the left button and drag the little picture onto your desktop. The computer will ask you if you want to set up a shortcut. Answer "yes." Now left click once on the desktop shortcut to PC99 and then press the right mouse button. Select Properties. Select the Program tab and set "Run" to "minimized," and put a check mark in the "close on exit" box. Now select the Screen tab and put a dot in the usage circle next to the word "window." Make sure all the boxes on the Screen tab are checked, leave everything else at the default setting and click on OK. You are now ready to input text into the BASIC editor running under PC99.

With PC99 you can input only from a text file one line of code at time into

the BASIC editor. For a program listing this means one line number at a time, not the whole program at once. For command mode this means one command or series of commands separated by double colons. You input into the 99/4A BASIC editor running under PC99 whatever you would normally type just before pressing Enter to have the editor accept your input.

Start the program that has your text file. This can be an email message, a notepad or wordpad or other word processor text file, an Acrobat reader file such as CaDD Electronics' scanned in TI cartridge documentation booklets, or almost any type of Windows 95-compatible program that contains text. Position the mouse cursor at the beginning of the text you want to input, press and hold the left mouse button, drag the mouse to the end of the text, and release the left mouse button. This will highlight the portion of text you wish to input into PC99. Move the cursor up to the top of the Windows 95 screen and click on Edit. Then click on "copy." Your text fragment is now in the Windows clipboard.

Now start PC99 by clicking on the shortcut. To see PC99 you may need to click on its name on the taskbar at the bottom of the desktop. Select Extended BASIC or TI BASIC, and wait until you see the flashing TI cursor in BASIC command mode at the bottom of your PC99 TI screen. If you have an SVGA monitor and properly sized windows, you can see both PC99 and your text application on the monitor screen at the same time.

Move the mouse pointer to the tool bar at the top of the PC99 window and point to the paste tool. Its looks like a clipboard partially covered with a piece of paper and will say paste after a couple of seconds when you put the cursor on the tool. Now for the magic! Press the left mouse button and your text will appear one letter at a time in PC99's BASIC command mode screen just as if you were typing it yourself into BASIC command mode. Proofread this newly "typed" code to make sure there are no errors (sometimes there are errors). Then press Enter to have this input accepted by the TI BASIC interpreter. Now move the mouse cursor to your windows text application, highlight the next line number of program code or command mode command, and move that into the PC99 BASIC editor. You can enter an entire BASIC program, one line number of code at a time, using this method.

PC99 makes a few errors when accepting input via copy and paste, so you should compare the PC99 BASIC editor screen to your original. Double colons don't always come out spaced correctly, and sometimes a character is dropped. Copy and paste isn't perfect, but it sure is an easy way to enter a BASIC program into PC99.

TEXTLOADER by Curtis Alan Provence

You need no special hardware beyond a basic 99/4A disk system to use

Continued on page 46

MICROREVIEWS

Continued from page 45

Textloader. It will also work from GPL mode on a Geneve. Essentially, this is a 13-sector XB program with attached assembly code that lets you load in D/V80 text files into XB. Text can be loaded from a disk file, PIO, or RS232. The needed D/V80 files can be created by TI-Writer with no modification because Textloader ignores carriage return symbols and the tab record that is written at the end of most TI-Writer files. Once loaded from XB, perhaps automatically as DSK1.LOAD, Textloader gives you access to several CALL LINKs either from within a running program or from XB command mode. These links remain available even after NEW is used to erase the program in memory.

CALL LINK("BATCH") lets you use a D/V80 file as a command mode batch file. Each line in the text file is considered a separate batch command, with these commands being limited to 80 characters. A text batch file example is provided that does a CALL FILES(1), NEW, and then runs a large memory image XB program all automatically just by loading a slightly modified version of Textloader. This application is how I was first introduced to Textloader. A corespondent sent me a disk full of music programs too large to normally run out of XB without CALL FILES(1). With Textloader the whole disk plays automatically directly from DSK1.LOAD.

CALL LINK("OLD") lets you load a program into memory. Although this can also be done with BATCH, OLD will let you input long XB program line numbers that exceed 80 characters in length and use several lines of your text file. This lets you use indentations in your XB text source file to make your XB code very easy to understand, similar to the indentations used in C source code. Each element of a complex FOR-NEXT loop or each element of a multi-command line of XB code separated by double colons can be written on a separate line and indented in the D/V80 source file. OLD will erase any XB program already in memory.

CALL LINK("MERGE") does the same thing as OLD but does not erase XB line numbers already in memory.

CALL LINK("HELP") brings up a help screen. The standard version of Textloader is set up to automatically bring up the help screen when Textloader first loads and then returns you to XB command mode. You can bring up the help screen later at any time with this call link.

This is the best software of this kind for those lacking a GRAM device or a Geneve. If you have this hardware then you should consider RXB v1003 or higher, which is an enhanced series of Extended BASIC GRAM files that includes "load text into XB" capabilities. The only significant limitation to Textloader is that it is picky about which version of Extended BASIC you are using. It works with TI Extended BASIC v110 but may not work with enhanced XBs. For example, it does not work with the Mechatronic Extended BASIC cartridge.

Textloader is feeeware. No donation is required, although the author says he will accept anything you want to send him. The DS/SD disk comes with commented source code, on-disk documentation, and a nice generic assembly loader that will let you boot E/A5 software from Extended BASIC. Send me $1 and I will send you the disk.

BASIC BUILDER by Paolo Bagnaresi

Like Textloader, this software has been around for several years. In fact the December 1987 v1.1 of Basic Builder probably predates Textloader. Essentially what you do is load Basic Builder into XB and then from either command mode or within a program enter CALL LINK("BUILD","DSKx.FILENAME"). Your XB program written as a D/V80 text file is then loaded into XB and can be immediately run. Only programs can be loaded, not command mode commands such as CALL FILES(1).

Like Textloader, your source text file can have one line of XB code spread over several lines of text with indentations to make the source file easy to understand. When Basic Builder encounters a number at the beginning of a line of text that is integer, positive, and less than 32768 it assumes that this is a new XB program line number. A text line beginning with anything else is added to the current XB program line number, with one interesting exception.

The exception is text lines beginning with the digit zero. Such lines of text are ignored and not pumped into the XB editor. You can use these "zero" lines to heavily comment your XB programs and not have these comments take up valuable XB memory. Normally, XB comments using REM or ! take up XB memory.

Basic Builder comes on a DS/SD disk complete with assembly source code, on-disk documentation, and text source code for an XB version of the game Frogger. The software is shareware. The author requests $5. You might send the author a letter of appreciation first to be followed later by cash if your letter is not returned by the post office because the 1987 Italian address may now be outdated. If you send me $1, I will mail you the Basic Builder disk.

SCSI CAT by Bruce Harrison

This public domain offering does something no other software will do. From within the Extended BASIC environment SCSI CAT gives you a catalog of any directory or subdirectory on any device, including SCSI and HFDC hard drives, RAMdisks, and floppies.

Lack of a good cataloging program has to date been a major problem for 99/4A systems that have SCSI hard drives. Run SCSI CAT, enter the device path to catalog, and see the catalog displayed on screen. If the display is long you can page up/down through the list of programs. The usual information is provided —

Continued on page 48

Continued from page 47

program type, program length in sectors, and protection status. The number of sectors used and free on your hard drive or floppy is correctly displayed. Subdirectory names are also indicated. When you exit SCSI CAT on a 99/4A, you are back to XB command mode. The program also works from XB on a Geneve but will not return cleanly to XB command mode.

This is not a disk manager. It just displays program and subdirectory names. You can't run programs from this display and you can't automatically bring up the list of files in a subdirectory. You have to type in the full path of the subdirectory to do this, but at least SCSI CAT will tell you what path name to type.

If you have a 99/4A system with any type of hard drive, you should have SCSI CAT. No other software does what SCSI CAT does from within the Extended BASIC environment.

EXTENDED BASIC V2.5 AND V2.6 by Tony Knerr

And speaking of unique features from within the Extended BASIC environment, here are the latest enhanced Extended BASIC offerings from Tony Knerr. Version 2.5 is for those with a 99/4A system that includes a GRAM device. Version 2.6 is for use on a Geneve and has some nice Geneve-specific features. I have reviewed Tony's enhanced Extended BASICs before, and all of the features of previous versions are included in these updates. What is new is disk formatting from XB command mode.

From command mode in XB type CALL SSSD or CALL DSSD or CALL DSDD to format a disk to the specified density. You can't use CALL DSDD with a TI controller. If you have the hardware to format in quad-density, Tony offers to send you a special version of his Extended BASIC that includes a CALL DSQD.

In each case you are prompted to put a disk in DSK1 and press Enter. The disk is then formatted without verification except for sectors 0 and 1. When the formatting is complete, you are presented with a disk catalog showing the number of sectors available on your newly formatted disk. You are then returned to XB command mode.

I know of no other product that formats a disk from within Extended BASIC. These enhanced Extended BASICs are public domain. Each comes on a DS/SD disk with plenty of on-disk documentation. I'll send you v2.5 for the 99/4A or v2.6 for the Geneve for $1 each.

ACCESS

Charles Good (source of all the software reviewed here except PC99)
P.O. Box 647, Venedocia OH 45894; 419-667-3131; email good.6@osu.edu
CaDD Electronics (source for PC99); 45 Centerville Dr.; Salem NH 03079; 603-895-0119; email mjmw@xyvision.com

Using Windows 95 to put V9T9 in its place

BY BRIAN TRISTAM WILLIAMS

This article originally appeared in BugBytes, the newsletter of the TI Brisbane User Group (TIBUG). The author can be reached via email at POLAR@globalco.za — Ed.

WIN95:'MS-DOS PROMPT' CONTEXT-MENU ADDITION

You load up Windows Explorer, and look at your V9T9 directory. You have the directory visible in Explorer, but you'd like to be there to do something in DOS, like run one of the PC programs in the UTILS directory. The default way would be to click on Start -> Programs -> MS-DOS prompt.

You'd then be thrown into Windows' default directory. To get to V9T9, you'd have to remember where V9T9 was (on your drive), then type something like: CD \APPS\UTILS\V9T9 to get there.

OK, easy enough, but this can get really tedious to do over and over again. The solution: How would you like to be able to right-click on any directory in Windows Explorer's lefthand-pane, then click on MS-DOS prompt, and be dropped off in the directory of your choice?

Here's how to do it:

In Windows Explorer, go to the menu and click on 'View' -> 'Options.' Then go to the File Types tab.

Scroll down to the registered File Type named 'Folder,' click on it, then click the 'Edit' button.

You will see the 'Edit File Type' dialogue. Click on the 'New' button. The 'New Action' dialogue pops up.

In the 'Action' field, enter "MS-DOS Prompt."

In the 'Application' field, you type "C:\Win95\command.com" (Note, however, that Win95 is the name of MY Windows 95 directory — you will need to replace this with your own directory's name — usually "WINDOWS".)

Click on the 'OK' button to close the 'New Action' dialogue, close the 'Edit File Type' dialogue using the 'Close' button, then close the 'Options' dialogue, using the 'Close' button.

From now on, you should be able to go to the directory of your choice by right clicking on it (in Windows Explorer's left-panel) and clicking on 'MS-DOS Prompt'.

You can switch this prompt, then start up Notepad, load up the file, read it, then close Notepad and go back and delete the file. And all you wanted to do was take a look at it! Wouldn't it be nice if you could look at such a file with Notepad, with the correct formatting, after two double-clicks?

Continued on page 50

Continued from page 49

First, you need to create a file named DV8OREAD.BAT, which looks like this:

```
C:\APPS\UTILS\V9T9\utils\ti2txt %l
NOTEPAD %1.txt
erase %1.txt
```

Make the following change: replace "C:\APPS\UTILS\V9T9" with the name of your V9T9 directory. Create this file by starting Notepad, and typing it in. After you've typed in this program, save it to a directory which is in your MS-DOS PATH, such as "C:\WINDOWS\COMMAND". Do this by going to Notepad's menu, then clicking on 'File' -> 'Save As...', and typing 'C:\WINMWS\COMMMD\DV80R@.BAT', then closing Notepad.

OK, you've done that, but how do you get it to run when you choose the D/V80 file you need to view?

Well, first you need to make DV8OREAD.BAT appear in your 'Open With' dialogue box. This is the dialogue that pops up when, in Windows Explorer, you double-click on a file with an extension that is not associated with any application.

In order to do this, open Notepad, type in a word or two, then save the file as "C:\1.!" This will put the file 1.!" in your root directory, for now.

Close Notepad, then go into Windows Explorer, and double-click on this file.

You will get the 'Open With' dialogue, most likely. Click the 'Other' button, then find the directory of DV8OREAD.BAT by double clicking on "Windows," then "Command," then "dv80read.bat".

This will select this file and close the 'Open With' file-selection dialogue box. Close the 'Open With' dialogue box by clicking the 'Close' button. You will then get an error message. Click the 'No' button, and close Notepad.

You can now delete the "1.!" file — it is no longer necessary.

Now you can go to a V9T9 FIAD directory such as DISK, double-click on a known D/V80 file, and it should pop up in Notepad. You can then save this file anywhere else, in DOS text file fornat.

There is one instance when this method won't work. If your D/V80 file has any characters which DOS wouldn't accept, such as a file named "FW/ DOCS", which could not contain the '/' character. V9T9 will change the '/' character to one the PC would accept, and this would confuse this little batch file. Also, if your filename is longer than eight characters, it won't work.

Chicago users slate 15th annual Faire

The 15th Annual Chicato TI International World Faire is scheduled from 9:30 a.m. to 4 p.m. Nov. 8 in the Evanston Public Library, at the corner of Church and Orrington streets in Evanston, Illinois.

The event is sponsored by the Chicago TI Users Group. For further information, contact Hal Shanafield, (847) 864-8644.

Fest West lines up hotels, TI facility

Plans for Fest West '98 in Lubbock are being finalized, according to event organizer Tom Wills of the Southwest 99ers User Group.

Wills says "at this time we do need to hear from vendors who are interested in setting up a booth. The room for the vendor activities is easily accessible for vendors to move equipment in and out of."

The vendor area will be open from 1 to 6 p.m. Feb. 14.

"Hopefully a shorter time period will be more productive and buyers won't need to be wandering around all day waiting until the end of the day for expected bargains," Wills said.

So far, two vendors have said they will attend.

TWO SITES

The fair will be held at two sites, one of them a TI production facility and the other a hotel. The TI facility was at one time used to produce the TI home computer.

Those who attend the activities at the TI facility, which will be offered at no charge, will have to be abide by a number of restrictions, including:

- No sales of any sort on TI grounds.
- Because of security policies for this type of facility, attendees will not be able to come and go at will. Everyone will have to register in the TI lobby between 8 and 8:30 a.m. An information card and a liability waiver will have to be filled out for each visitor. The activities will end at noon.
- No cameras will be allowed during any of the activities. Arrangements are being made to take "official" pictures which can be made available to those in attendence.
- Except for tours of the facility, those attending will have to remain in the area designated for the Fest West activities.

SCHEDULE OF EVENTS

The activities at the TI production facility will be conducted in the morning. These activities are expected to include tours, speakers, and a mini-museum. Speakers will include TI employees who were instrumental in the development of the TI99/4A. TI officials are contacting members of the development team to have them present at the activities.

From 1 to 6 p.m., the vendor portion of Fest West will take place at the Sheraton Four Points Hotel. As part of FW98, there will also be a hospitality room set up in the hotel which will be open from 1 to 10 p.m.

DRIVING DISTANCES

Some distances to Lubbock include:

City	Distance in miles
Tucson, AZ	668
Chicago, IL	1172
Atlanta, GA	1309
Los Angeles, CA	1168
Denver, CO	500
Cleveland, OH	1378
Seattle, WA	1767

Fest West will have two official

Continued on page 52

FEST WEST '98

Continued from page 51

hotels. The first is the Sheraton Four Points Hotel. There is a block of 100 rooms set aside for Fest West attendees. Prices are as follows:

Single/double room (1 or 2 people) $54 with an additional $10 for each extra person up to 4 persons per room. Taxes are extra. The rooms will be set aside for Fest West until 30 days before the event. Mention Texas Instruments Computer Fair for the special rates when making reservations.

The second hotel is the Koko Inn. A block of 100 rooms has been set aside, with another possible 100 rooms at the Koko Inn's sister hotel, the Villa Inn. Each room has either one king-sized bed or two queen-sized beds. The rates for the Koko Inn are $44 a night for up to four people per room. The block of rooms will be held until two weeks before Fest West. Call 800-448-3525 or 806-747-3525 to make reservations. Mention Fest West '98 for the special rates when making reservations.

For tourism information, call the Convention and Tourism Bureau of Lubbock at 1-800-692-4035.

TRANSPORTATION

Lubbock International Airport is served by five major airlines — American Eagle, ASA The Delta Connection, Continental, Southwest, and United Express.

Lubbock has 12 rental car companies to accommodate visitors. Rental agencies at the airport include: Advantage, Avis, Hertz, and National. Off-site rental agencies include: Advantage, Agency, Budget, Discount, Enterprise, Sears, Snappy, Thrifty, and Trusty.

Agencies offering van rental are Advantage, Discount, Thrifty, and Trusty.

All major convention hotels offer free airport shuttle service.

USER NOTES

Hidden powers of MIDI Master

There are hidden powers in MIDI-Master that we've just discovered. It started with a question from Richard Bell, who was testing our new version 2.5Z. He has a very new and advanced Casio keyboard, and discovered from its manual that it will accept and play MIDI notes a full octave below its own keys. The lowest key on the keyboard carresponds with the note 0C in MIDI-Master's SNF notation. He asked whether there could be a way to send that lower octave from MIDI-Master. There is!

According to its docs, MIDI-Master handles notes from 0C through 5C. By examining the source code in its compiler, we learned that it doesn't check the octave character for numeric values. It simply takes the ASCII value and performs integer math operations on it. It looked as though if we used the next lower ASCII character (/) in place of the zero, that we could compile a whole octave below 0C. This works!

USER NOTES

We ran a test with our Yamaha PSR 300, which also can play MIDI notes a full octave below its keys. We were able to play a scale starting with /C and running through 0C via MIDI-Master. We also discovered that we could put the octave up through 7C and our Yamaha would play that, too.

Not all keyboards have this capability, so it's best to be a bit careful using it. Our older Casio CT-650 will transpose any notes below its key range up into the range of its keys. This is probably true for other older model keyboards, but the newer

Continued on page 54

USER NOTES

Continued from page 53

ones can range all the way from one octave below 0C up through 7C when played by MIDI-Master.

PC-TI file transfers

The following was written by Bruce Rodenkirch and appeared in the newsletter of the Cleveland Area TI99/4A User Groups.

I got an old PC (for free at a Hamfest) with an 8088 CPU and have been spending some time with it. I bought a 20-megabyte hard drive and, after a lot of trial and error and help from my buddies, got it formatted and running. This seemed like an idea place to store the many files I have been accumulating. It isn't too hard to do, I discovered.

I have it connected to my TI (Geneve) through the serial port and have been experimenting with tranferring programs back and forth. I use a "straight through" cable from the RS-232 card out the front of the Peripheral Expansion Box, which just acts as an extension cord.

Then I bought two DB-25 sockets, one male and one female, and soldered short wires, about four inches, between them. These are also connected straight through with short (one-quarter inch) gaps in the insulation, staggered to minimize the chance of a short circuit. You don't need to use 25 wires, only about 10 or so to cover the pins you need to connect the pins you plan to use.

I then plug my modem cable to this and, since it is wired for TI parallel use, the modem thinks it is connected directly to the RS-232 card.

This is handy if you need to experiment with changes in the RS-232-to-modem wiring, such as with Term 80 or the Port terminal program for the Geneve. It also makes it easy to hook up a clone to the TI with a "Y" cable, which can be easily constructed. Get another DB-25 plug of the gender needed for the clone cable. Connect pins 1, 2, and 3 to the same wires in the short cord described above. Number 1 is ground, and 2 and 3 are receive and transmit. Because the the clone and TI pins 2 and 3 are the opposite, the TI will transmit to the receive pin of the clone.

Using appropriate terminal programs for the two computers (Telco and Procomm, for example), ASCII text files and XMODEM file transfers can be made. Also, whatever is typed on one screen will appear on the other screen. Reversing pins 2 and 3 to the clone will send the downloaded info from the modem to both computers.

Speeding up BASIC

The following was written by John Hale and has appeared in several user group newsletters.

Remember, BASIC reads every program line and parts of lines in its path. Unnecesasry comments take time to read.

• If you must have your program description first in your program, make your first line read GOTO XXX. BASIC will then skip these

USER NOTES

lines (REMs) while executing and go to the line referenced in GOTO XXX.

• REM may be used after a line branch has been placed. But it won't be seen by BASIC.

• Use OPTION 1 if you have no use for a zero being scanned on each reading of an array.

• Do not use DEFine. Functions are the worse time users than GOSUBs. Use DEFine only when you have a very complicated operation requiring repeated use with a variety of variables.

• Avoid using GOSUB or GOTO to reach short routines. Replace them with in-line solutions, even if they are needed again elsewhere in the program.

• Never use an array variable if you can use a simple variable instead.

• Use one or two character variables. Habitually use the same variables for all loops.

• Throw away all variables which are used only once. Replace them with a transient variable that will handle all of the single use variables.

• Do not use a variable when not absolutely needed. Use a literal constant instead. Variables are kept in a table and require time to locate.

• Do not write loops for short, repetitive sequences.

• Keep your programs linear.

• Do not use LET.

DISKS, ETC.

❑ Back Issues,$3.50 each to March 1996, later $6 each. List issues on separate sheet.
No price breaks on sets of back issues. Free shipping USA. Add $1, single issues to Canada/ Mexico. Other foreign shipping 75 cents single issue surface, $2.80 airmail. Write for foreign shipping on multiple copies.
OUT OF STOCK: V1#1-2; V2#1

❑ MICROpendium Index (2 SSSD disks, 1984-1992), XBASIC required $6

❑ MICROpendium Index II (9 SSSD disks, 1984-1992), XB req. $30

❑ MICROpendium Index II with MICROdex 99 (11 SSSD disks), XB required $35

❑ MICROdex 99 (for use with MP Index II, 2 SSSD disks), XB required $10

❑ Index II annual disks ordered separately (1 disk per year, 1984-1992); each $6

❑ MICROdex 99, by Bill Gaskill, is a collection of programs that allow users of MP Index II to modify their index entries, as well as add entries. MICROdex 99 supports many other functions, including file merging, deletion of purged records, record counting and file browsing.

GENEVE DISKS (SSSD unless specified)

❑ MDOS 2.21 (req. DSSD or larger (for floppy & hard drive systems) $4

❑ GPL 1.5 .... $4

❑ Myarc Disk Manager 1.50 .... $4

❑ Myarc BASIC 3.0 .... $4

❑ MY-Word V1.21 .... $4

❑ Menu 80 (specify floppy or HD version; includes SETCOLR, SHOW-COLOR, FIND, XUTILS, REMIND $4

GENEVE PUBLIC DOMAIN DISKS

These disks consists of public domain programs available from bulletin boards. If ordering DSDD, specify whether Myarc or CorComp.

	SSSD	DSSD	DSDD
❑ Series 1	$9	$7	$5
❑ Series 2	$9	$7	$5
❑ Series 3	$9	$7	$5
❑ Series 4	$9	$7	$5
❑ Series 5	$9	$7	$5
❑ Series 6	$9	$7	$5