TRS-80® Microcomputer News

Volume 6, Issue 2 February, 1984

$1.50

Fort Worth Scene

UTILITIES

Well, here it is Utilities time again. My how the months speed by. At this time last year the Model 12 was the big news. There's been a lot of water (or new products) under the bridge since then. In fact we have introduced so much that's new this year we were concerned about neglecting some of the interesting and useful programs you've sent to us. I think this issue will alleviate some of the guilt.

You'll find a lot of information packed in these few pages. Reader submitted programs contain several useful ideas. There's a slightly different backup scheme from Computer Customer Services, including procedures for equipment that wasn't even available this time last year. Backups—or the lack thereof—is the sort of stuff that nightmares are made of, so anything that will encourage you to develop and sustain a reliable system of backups is all to the good.

13 GHOSTS AND ORCHESTRA 90

One look at the graphics for this game should tell you that "13 Ghosts" is a game you'll want to see even if you don't own a Model III or 4. Though the art work provided is great, you still can't imagine how impressive the animation is when seen in action. "13 Ghosts" is lively, full of visual impact and sound.

Bryan also provides us with " . . . a way to load, compile and play more than a few files with one command" in his Musical Notes column this month.

COMPUTER CUSTOMER SERVICE REORGANIZATION CORRECTION

In December we informed you of changes being made in the organization of Computer Customer Services. There are corrections that need to be made in the descriptions of Group Number 1 and Group Number 2.

Group Number 1 will cover those Operating Systems, Languages, and Compilers used by Models I, II, III, 4, 12, 16, and 16B. Group Number 2 will support Operating Systems, Languages, and Compilers for the Color Computer, Pocket Computers, and our popular Model 100.

If you didn't notice last month, there are new numbers to go with all these changes. They are listed at the end of the Customer Service article.

MAGAZINES

Below are nine magazines of special interest to TRS-80 owners that we believe have editorial content of high quality and will be of use to our customers.

Basic Computing—The TRS-80
User Journal
3838 South Warner Street
Tacoma, WA 98409
(206)475-2219

Color Computer Magazine
Highland Hill
Camden, ME 04843
(207)236-9621

Color Micro Journal
5900 Cassandra Smith Road
Hixson, TN 37343
(615) 842-4600

80 Micro
P.O. Box 981
Farmingdale, NY 11737

Hot CoCo
P.O. Box 975
Farmingdale, NY 11737

Portable 100—The Magazine for
Model 100 Users
P.O. Box 468
Hasbrouck Heights, NJ 07604

PCM—The Portable Computing Magazine
9529 U.S. Highway 42
P.O. Box 209
Prospect, KY 40059

Rainbow (Covers the TRS-80 Color Computer)
P.O. Box 209
Prospect KY 40059
(502)228-4492

two/sixteen magazine
P.O. Box 1216
Lancaster, PA 17603
(717)397-3364

Cover art "The Utilities Factory" by Bill Semand, artist, Radio Shack Advertising Department.

TRS-80® Microcomputer News

TRS-80 Microcomputer News
Volume 6 Issue 2
FEBRUARY 1984

TRS-80 Microcomputer News is published monthly by Radio Shack, a division of Tandy Corporation, One Tandy Center, Fort Worth, Texas U.S.A. 76102. Copyright 1984 by Tandy Corporation, One Tandy Center, Fort Worth, Texas U.S.A. 76102. All rights reserved.

Reproduction or use, without express written permission from Tandy Corporation, of any portion of the Microcomputer News is prohibited. Permission is specifically granted to individuals to use or reproduce material for their personal, non-commercial use. Reprint permission for all material (other than Ivan Sygoda's Profile article), with notice of source, is also specifically granted to non-profit clubs, organizations, educational institutions, and newsletters.

TRS-80 Microcomputer News is published monthly by Radio Shack, a division of Tandy Corporation. A single six month subscription is available free to purchasers of new full size TRS-80 Microcomputer systems with addresses in the United States, Puerto Rico, Canada and APO or FPO addresses. Certain smaller TRS-80 Microcomputers will not include this free subscription. Subscriptions to other addresses are not available.

The subscription rate for renewals and other interested persons with U.S., APO or FPO addresses is twelve dollars ($12.00) per year, check or money order. Single copies of the Microcomputer News may be purchased from Radio Shack Computer Centers or Computer Departments for $1.50 suggested retail each.

The subscription rate for renewals and other interested persons with Canadian addresses is Fifteen dollars ($15.00) per year, check or money order in U.S. funds. All correspondence related to subscriptions should be sent to: Microcomputer News, P.O. Box 2910, Fort Worth, Texas 76113-2910.

Retail Prices in this newsletter may vary at individual stores and dealers. The company cannot be liable for pictorial and typographical inaccuracies.

Back issues of Microcomputer News prior to January, 1981 are available through your local Radio Shack store as stock number 26-2115 (Suggested Retail Price $4.95 for the set). Back issues of 1981 copies are available as stock number 26-2240 (Suggested Retail Price $9.95 for the set). The 1982 back issues copies are available as stock number 26-2241 (Suggested Retail Price $12.95 for the set).

The TRS-80 Newsletter welcomes the receipt of computer programs, or other material which you would like to make available to users of TRS-80 Microcomputer systems. In order for us to reprint your submission, you must specifically request that your material be considered for reprinting in the newsletter and provide no notice that you retain copyrights or other exclusive rights in the material. This assures that our readers may be permitted to recopy and use your material without creating any legal hassles.

Material for publication should be submitted on magnetic media (tape, disk, or CompuServe). If you submit material on tape or disk, and it is accepted for publication, we will send you two cassettes or diskettes for each one you sent us. Cassettes will come from our box of mixed blank cassettes. If you submit material on CompuServe, and we think we may use the material, we will extend your Microcomputer News subscription by six months for each article accepted. If you are submitting material over CompuServe, please include your name and address or your subscription number so we can find you. If the material is very short, send it to us in E-Mail. If you have more than a few lines, you need to place the material in the ACCESS area of CompuServe and then let us know it is there by leaving a message on E-Mail.

Material may be submitted by mail to P.O. Box 2910, Fort Worth, Texas 76113-2910, or through CompuServe. The Microcomputer News' CompuServe user ID number is 70007,535.

Programs published in the Microcomputer News are provided as is, for your information. While we make reasonable efforts to ensure that the programs we publish here work as specified, Radio Shack can not assume any liability for the accuracy either of the programs themselves or of the results provided by the programs.

Further, while Microcomputer News is a product of Radio Shack, the programs and much of the information published here are not Radio Shack products, and as such can not be supported by our Computer Customer Service group. If you have questions about a program in the Microcomputer News, your first option is to write directly to the author of the program. When possible, we are now including author's addresses to facilitate communications. If the address is not published, or if you are not happy with the response you get, please write us here at Microcomputer News. We will try (given the limited size of our staff) to find an answer to your question and, in many cases, will publish the answer in an up-coming issue of Microcomputer News.

Trademark Credits

CompuServe™	CompuServe, Inc.
CP/M®	Digital Research
Dow Jones™ NEWS/RETRIEVAL Service®	Dow Jones & Co., Inc.
LDOS™	Logical Systems, Inc.
VisiCalc®	VisiCorp, Inc.
XENIX™	Microsoft
Program Pak™	Tandy Corporation
SCRIPSIT™	Tandy Corporation
TRSDOS™	Tandy Corporation
TRS-80®	Tandy Corporation

Contents:

Prices shown in TRS-80 MICROCOMPUTER NEWS are in U.S. Funds.

Backup—Our Favorite Utility

There have been countless computer users who have had the misfortune of having their computer system "crash" and have had to start over with their accounting systems, rewrite numerous letters for their word processors, or reenter massive amounts of data into data base management systems. Most of these situations would not have occurred if the user had only taken advantage of the "BACKUP" or "SAVE" function of their system. In the past three years, there have been two very good articles on backup in the *TRS-80 Microcomputer News*, July 1981 page 22, and February 1983 page 9. However, since that time we have added new computer systems and storage peripherals, and the backup requirements have changed.

WHEN TO MAKE BACKUPS

How many backups should you have and how often should you make a backup? A good rule of thumb is to make a backup anytime the information you have stored would take longer to reconstruct or reenter than it would to make a backup. There will be times, of course, when you write a short note or memo, and you won't want to make a backup immediately. It would be faster to rewrite the memo if the information were lost. If you have spent thirty minutes carefully wording a memo, it might be worth the time to make a backup copy or at least copy the document to another disk if you do not wish to make a full backup.

This brings up another point about operating systems and the backup utility. Older, less sophisticated systems required the user to make complete disk backups or what is called "mirror image" backups. Many of our newer operating systems now have options to only make backups of those files, or only those files that have been modified within a range of dates. These options make backing up important information much faster, but the user must be organized well enough to know where to find the most current backup in case disaster should strike. A simple backup schedule can keep even the novice computer user well organized.

A SIMPLE BACKUP SCHEDULE

One of the simplest backup schedules is to make a set of disks, one for each day of the week. On Monday use the disk marked "Monday," and at the end of the day simply make a backup onto the disk marked "Tuesday." If a problem is encountered while using the "Tuesday" disk, another copy can be made from the "Monday" disk. Assuming that Tuesday's work goes without a hitch, make a backup at the end of the day onto the disk labelled "Wednesday." Follow the same procedure throughout the week. On Friday, make a weekly set of backups on a set of disks labelled "Week 1" and put them aside in a safe storage place. (Hopefully, they will never be needed.) Also make a backup of Friday's work onto the disk labelled "Monday," and you are ready to start the next week's work. On Friday of the second week, make a second set of weekly backups on a set of disks marked "Week 2," and also make a backup onto the disk marked "Monday," and you are ready to start work the following week. Follow the same procedure throughout the month, making a weekly set of backups. At the end of the month you will have a set of five daily disks, and a weekly set for each week of the month. Additional backups should be made at the end of the month, once before doing any "end of month processing" and another set after end of month processing. Keep both of these sets in the event a problem is encountered during end of month processing the following month, and an unprocessed set of data is required to reconstruct a good working set of data files. Repeat the cycle the following month, adding another set of backups of your data before you do "end of month processing." With all of the previous sets of backups, we have backed up over the set of disks with the same label from the previous month. This is the one exception. If you backed up the second month's pre end of month processing data over the first month's data, you would be destroying the previous month's unprocessed data. Each month you will follow these same procedures. At the end of the year you should also make two sets of backups—one before "end of year" processing and one after the end of year processing has been completed.

WHAT TO DO WHEN DISASTER STRIKES

One of Murphy's Laws of Computers certainly must be "computers will fail at the most inopportune time," translated to mean that right in the middle of a statement run your computer returns an error, and no amount of coaxing will get the computer to complete the statements. Many computer operators will blithely insert the backup copy of their data and try the operation again. Is there anything wrong with this procedure? After all, why have we been making all of these backup copies? NEVER use the previous day's backup to complete your processing!! Always make another backup copy first. This will keep your original copy intact. It will also help detect a possible hardware problem. If your computer system will not produce a backup for you, there is a high degree of probability that you have some type of hardware problem, even if it is as simple as dirty disk drive heads.

BUT I HAVE A HARD DRIVE

Many computer users feel that because of the reliability of hard disk drives they no longer need to make as many backups or "saves" as they are referred to with hard drives. This is simply not true. Good data processing procedure dictates that a rigid and complete schedule of "saves" be followed no matter what type of storage device is used. Backups become even more important when using a hard drive because of the potential to lose much larger quantities of data. The "SAVE" utility allows the user to make backups of only those files that have been modified during the most recent processing, within a range of dates or a specified list of data files. Caution should be taken when using a backup scheme that only includes parts of the total data files.

TRS-XENIX SAVES

Do the same rules apply to TRS-Xenix user? Yes. The Xenix operating system, however, gives the user at least three options to use when making saves of their systems—Xenix Save, the Tsh Save, and the Sysadmin Save. However, because of the inability to do a restore after doing a Sysadmin Save, we recommend the use of either the Xenix Save or the Tsh Save because both of these options will allow you to restore your data.

CONCLUSION

We hope that the backup schedule presented here will help you in your computer operations. We've tried to make it as simple as possible. We realize that occasionally your computer may fail and destroy important data. We hope that you will be properly protected by using the backup schedule we have presented.

Computer Customer Service Address and Phone Numbers

8AM to 5PM Central Time
Computer Customer Services
400 Atrium, One Tandy Center
Fort Worth, Texas 76102

Productivity/Special Applications	(817)338-2390
Accounting Software	(817)338-2391
O/S and Languages, Group No. 1	(817)338-2392
O/S and Languages, Group No. 2	(817)338-2393
Hardware and Communications	(817)338-2394
Home Software	(817)338-2395
Educational Software	(817)338-2396
Newsletter Subscription Problems	(817)870-0407

NOTES PREVIOUS

JUNE 1982

Machine Language Sort, Part II

James E. Haher
303 Chestnut Street
Oneonta NY 13820

William Barden's Machine Language program sorts from A to Z (or from 0 on to higher numbers). A very simple change permits it to sort from high to low which is useful for the printout of averages such as for bowling teams.

At line 490 of his June 1982 program he uses BLS, branch if lower or same. I substituted BHS, branch if higher or same for reverse sort.

BLS is Hex 23 while BHS is Hex 24 and POKES from the BASIC program permits options to sort from low to high or high to low.

You're probably way ahead of me on this, but I thought I'd pass on my thought. I have a feeling many more CC owners have use for Bill Barden's sort. It is excellent.

AUGUST 1983

The New PC-3!

Bill Wrigley
4931 Rebel Trail, NW.
Atlanta GA 30327

I would like to call your attention to several errors or omissions in the August 1983 issue of TRS-80 Microcomputer news article on "The New PC-3!".

1) The Verbs and Commands tabulation on pages 7 and 8 includes MERGE for the PC-1 which is not available on the PC-1.
2) PEEK and POKE are absent from this tabulation but are accepted by both the PC-3 and the PC-2.
3) The RND Numeric Function as shown on page 8 is not available in the PC-1.

Incidentally, items 1) and 2) above also appear in error in the appendix included in the PC-3 Operating Manual.

OCTOBER 1983

Sieve of Eratosthenes

Norman Eaddy
10613 Montrose Ave. #201
Bethesda MD 20814

In the October 1983 TRS-80 Microcomputer News, Ken Weiss gave timings for the Sieve of Eratosthenes, originally published in April 1983. With his enhancements the program requires over 4 minutes on a TRS-80 Model II to find all prime numbers from 0 to 1800. The following program requires only 25 seconds on a Model I.

```
10 CLS
     : K0 = 0
     : K1 = 1
     : K2 = 2
     : INPUT "PRIMES UP TO "; N
     : PRINT K2;
     : FM = INT((N - K1)/K2)
     : DIM F(FM)
20 FOR F = K1 TO (INT(SQR(N)) - K1)K2
30 IF F(F) = K0 THEN K = F * K2 + K1
     : PRINT K;
     : FOR I = INT(K * K/K2) TO FM STEP K
     : F(I) = K1
     : NEXT I
40 NEXT F
50 FOR F = F TO FM
     : IF F(F) = K0 THEN PRINT F * K2 + K1;
60 NEXT F
70 END
```

This program uses Ken's enhancement and two others. Only odd numbers are processed (except 2); and when multiples of a prime are being crossed out, the first multiple crossed out is the square of the prime (all smaller multiples will have been already crossed out).

NOVEMBER 1983

PC-2 Diagnostic Tests Revisited

The 16K System ROM Check program for the PC-2 (November 1983, page 20) frequently returns an error message. There have been several versions of the PC-2 ROM, each with a different checksum. Therefore if you do not have the one version of the ROM with a checksum of 2011609, you will get a CHECKSUM BAD message when you run the program. We were unaware of the different checksums at the time the program was published and apologize for any inconvenience that you may have experienced.

FORTRAN Input

Chip Kraft
5 East Broadway
P.O. Box 293
Union Bridge, MD 21791

INTRODUCTION

I am a Civil Engineer and extensively use FORTRAN in my work. I applaud Radio Shack's choice of Microsoft FORTRAN. I have used Microsoft FORTRAN on several other computers as well as on my TRS-80.

Unfortunately, Microsoft FORTRAN lacks one important capability: interactive, unformatted keyboard input. In BASIC, the INPUT statement provides this capability. One can enter a series of numbers, separated only by blanks or commas, and BASIC will place the values into the correct variables. In Microsoft FORTRAN one must use formatted input which requires precise placement of data into the correct columns. This is difficult to accomplish on a computer keyboard because it requires counting spaces. It is easy to lose count of the number of spaces and shift the data left or right. This may result in the wrong numbers being input by the computer.

This weakness is common to all the versions of Microsoft FORTRAN I have seen. Until Microsoft does provide the capability in the language, however, the subroutines URREAD, UIREAD and U4READ can be used.

This documentation is divided into two sections. The first section is practical user's documentation: it describes how to put URREAD, UIREAD and U4READ onto your computer system, how to structure subroutine calls from within a FORTRAN program, and how to access the routines using the linkage editor. The second section is for those who are interested in the details of how the program works.

SECTION 1—USER'S DOCUMENTATION

Subroutines UIREAD, U4READ and URREAD are placed into your computer by typing them into the editor and compiling them into /REL files. Then, they may be called by other FORTRAN programs when needed.

```
      SUBROUTINE UIREAD(NVAR,IARRAY)
      INTEGER IARRAY(NVAR)
      REAL ARRAY(15)
      CALL URREAD(NVAR,ARRAY)
      DO 6ØØ J=1,NVAR
6ØØ   IARRAY(J)= ARRAY(J)
      RETURN
      END
      SUBROUTINE U4READ(NVAR,IARRAY)
      INTEGER*4 IARRAY(NVAR)
      REAL ARRAY(15)
      CALL URREAD(NVAR,ARRAY)
      DO 6ØØ J=1,NVAR
6ØØ   IARRAY(J)= ARRAY(J)
      RETURN
      END
      SUBROUTINE URREAD(NVAR,ARRAY)
      BYTE IALPHA(8Ø)
      REAL ARRAY(NVAR)
C
C     URREAD READS INPUT AS
      ALPHANUMERIC CHARACTERS
C     AND RETURNS A REAL ARRAY OF NVAR
      NUMBERS
C
C     **********************************
C
C     INITIALIZATION
C
      ISIGN= 1
      LAST= Ø
      NUMBER= 1
      DO 7ØØ J=1,NVAR
7ØØ   ARRAY(J)= Ø.
C
C     **********************************
C
C
C     WRITE OUT "?" PROMPT ON SCREEN
C
1     WRITE(1,2Ø1)
2Ø1   FORMAT(' ? ')
C
C     READ IN ARRAY OF 8Ø ALPHANUMERIC
C     CHARACTERS
C
      READ(1,1ØØ) IALPHA
1ØØ   FORMAT(8ØA1)
C
C     **********************************
C
C     INITIALIZE SHIFT=Ø = ASSUME LEFT
C     OF DECIMAL POINT
C
      SHIFT= Ø.
      DO 6ØØ J=1,8Ø
C
C     SUBTRACT 48 FROM THE ASCII CODE
      TO CONVERT
C     TO THE ACTUAL DIGIT
C
      K= IALPHA(J)- 48
C
C     IF THE VALUE IS ZERO THRU NINE,
      THEN THE
C     CODE WAS A DIGIT.  IF NOT, IT
      COULD BE A
C     MINUS SIGN, A DECIMAL POINT, A
      SPACE OR
C     AN ILLEGAL CHARACTER
C
C     IF A DIGIT, GO TO 4, THE SECTION
      THAT BUILDS
C     NUMBERS
C
      IF(K .GE. Ø .AND. K .LE. 9) GO TO
      4
C
C     **********************************
C
C     TEST FOR MINUS SIGN
C     45= ASCII CODE FOR -
C
      IF( IALPHA(J) .NE. 45) GO TO 2
      ISIGN= -1
      GO TO 6ØØ
C
C     TEST FOR DECIMAL POINT
C     46= ASCII CODE FOR .
C
2     IF( IALPHA(J) .NE. 46) GO TO 6
C
C     ADJUST SHIFT PARAMETER TO MEAN
      RIGHT
C     SIDE OF DECIMAL POINT
C
      SHIFT= 1Ø.
      LAST= 1
      GO TO 6ØØ
C
```

```
C       **********************************
C
C       NOW BUILD NUMBER
C
4       IF(SHIFT .NE. 0.) GO TO 5
C
C       THIS NUMBER IS LEFT OF DEC PLACE
C
        ARRAY(NUMBER)= ARRAY(NUMBER)*10+K
        LAST= 1
        GO TO 600
C
C       THIS NUMBER IS RIGHT OF DEC PLACE
C
5       ARRAY(NUMBER)= ARRAY(NUMBER)+
        K/SHIFT
        SHIFT= SHIFT*10.
        GO TO 600
C
C       **********************************
C
C       NO MATCH FOUND, NEW NUMBER
C
6       IF( LAST .NE. 1) GO TO 7
        ARRAY(NUMBER)= ARRAY(NUMBER)*
        ISIGN
        ISIGN= 1
        NUMBER= NUMBER+1
        LAST= 0
        IF(NUMBER .GT. NVAR) RETURN
7       SHIFT= 0.
C
C       32 IS ASCII CODE FOR BLANK
C       44 IS ASCII CODE FOR COMMA
C
        IF( IALPHA(J) .EQ. 32 .OR.
        IALPHA(J)
      * .EQ. 44) GO TO 600
        WRITE(1,200) IALPHA(J)
200     FORMAT(' UNKNOWN CHARACTER ',A1)
600     CONTINUE
        IF(NUMBER .GT. NVAR) RETURN
C
C       IF NOT ENOUGH NUMBERS HAVE BEEN
        READ
C       IN YET, GO TO 1 AND GET ANOTHER
        LINE
C       OF INPUT FROM THE KEYBOARD
C
        GO TO 1
        END
```

Calls to these subroutines have the following format:

```
CALL UIREAD(NVAR,IARRAY)  For regular integers
CALL U4READ(NVAR,IARRAY)  For INTEGER*4
                          variables
CALL URREAD(NVAR,ARRAY)   For real variables
```

where NVAR = the number of variables to be read, and ARRAY or IARRAY = an array of the variables to be read.

Suppose one wishes to read four real variables A, X, Y, and Z. An appropriate FORTRAN statement would be:

```
CALL URREAD(4,A,X,Y,Z)
```

This type of input is used by the sample program TREAD.

```
        PROGRAM TREAD
C
C       DEMONSTRATES USE OF URREAD
        SUBROUTINE
C
        WRITE(1,100)
100     FORMAT(' ENTER A, B, AND C')
        CALL URREAD(3,A,B,C)
        WRITE(1,101) A,B,C
101     FORMAT(3F8.2)
        STOP
        END
```

Arrays of variables can also be read by these subroutines:

```
INTEGER*4 ICOUNT(10)
   .
   .
   .
CALL U4READ(10,ICOUNT)
```

This type of input is used by the sample program TREAD1.

```
        PROGRAM TREAD1
C
C       DEMONSTRATES USE OF URREAD
        SUBROUTINE
C
        REAL ARRAY(3)
        WRITE(1,100)
100     FORMAT(' ENTER 3 VALUES OF
        ARRAY')
        CALL URREAD(3,ARRAY)
        WRITE(1,101) (ARRAY(J),J=1,3)
101     FORMAT(3F8.2)
        STOP
        END
```

URREAD can read in as many variables as desired. UIREAD and U4READ are limited to a maximum of 15 variables on each call, due to the size of their internal dimension statements. It is extremely important to use the proper subroutine depending on the type of variable to be read in since each variable type has a different internal representation. Variable types cannot be mixed in one statement; separate calls are required to each subroutine for each variable type. One alternative is to read everything in as real values using URREAD and then convert selectively those which are to be stored as integers, using replacement statements or the IFIX subroutine.

When called, these routines return with a "?" prompt, indicating they are ready for data input. Then the routine requires NVAR separate keyboard inputs. These can be entered on the same line separated by commas or spaces, or entered on successive lines. The subroutines will continue asking for more data until the requirement for NVAR values has been satisfied.

To access the subroutines, the module URREAD/REL must be searched by the linkage editor. This is accomplished by including the module's name in the command list when the FORTRAN program is being linked. For example, to link sample program TREAD:

```
TRSDOS Ready
L80

*TREAD,URREAD-S,FORLIB-S,TREAD-N-E
```

The above sequence would load the main program, TREAD. Then the subroutine libraries URREAD and FORLIB would be searched for any subroutine references. Finally, TREAD/CMD would be saved on the disk.

SECTION 2—PROGRAMMING DOCUMENTATION

The two subroutines UIREAD and U4READ both call URREAD to get keyboard input. Once the data has been read in and stored in as real numbers, UIREAD and U4READ convert these to the appropriate INTEGER and INTEGER*4 formats before returning to the calling program. The conversion is made by truncating the decimal portion. For example, 5.3 rounds down to 5; and 5.9 also rounds down to 5. If you desire to round instead of truncate, lines 00600 and 01400 can be modified as follows:

```
ØØ6ØØ 6ØØ   IARRAY(J)= ARRAY(J)+ .5
```

Subroutine URREAD performs the processing of keyboard input. Rather than reading in data as numerical values, URREAD handles all keyboard input as alphanumeric characters and then converts to numbers as appropriate.

To understand this process, one must first understand how FORTRAN stores and processes alphanumeric characters. Keyboard characters cannot be stored directly in computer memory. Instead, the ASCII code for each character is stored. These are in fact integers which can be added, subtracted, multiplied or divided as any other numbers.

The program ASCII demonstrates this. It creates an ASCII code table by printing out the contents of storage location I in both integer and alphanumeric format. To prove that I is considered an integer storage location by the computer, note that it is also a DO loop counter!

```
ØØ1ØØ            PROGRAM ASCII
ØØ2ØØ  C
ØØ3ØØ  C         GENERATES AN ASCII CODE TABLE
ØØ4ØØ  C
ØØ5ØØ            BYTE I
ØØ6ØØ            WRITE(2,1ØØ)
ØØ7ØØ  1ØØ       FORMAT(' ASCII CODE   CHARACTER')
ØØ8ØØ            DO 6ØØ I=48,9Ø
ØØ9ØØ            WRITE(2,1Ø1) I, I
Ø1ØØØ  1Ø1       FORMAT(5X,I2,9X,A1)
Ø11ØØ  6ØØ       CONTINUE
Ø12ØØ            STOP
Ø13ØØ            END
```

In other words, there is nothing special about a variable or array containing alphanumeric information. The numbers in the array can be processed just like numbers from any other source.

Now suppose an alphanumeric array containing the following three numbers, 12.35, −.7 and 8 were read in from the keyboard:

1 2 . 3 5 − . 78

When stored, the array would actually contain the following integers which are the ASCII codes for the above characters:

49 50 46 51 53 32 45 46 55 32 56

First, note that ASCII codes 48 through 57 correspond directly to the numbers 0 through 9. By subtracting 48 from the ASCII code stored in the array, the actual numerical value of the digit is obtained in integer format. In URREAD, this converted value is stored in variable K. In addition, there are the special codes 32 = blank, 44 = comma, 46 = decimal point and 45 = minus sign which must receive special handling. Any other codes are illegal and are flagged.

The numbers are built a digit at a time. SHIFT is initially set equal to zero for each number, indicating that the number is left of the decimal point. When a decimal point is encountered, SHIFT is set equal to 10., and a different shifting scheme is employed. The following example demonstrates how the procedure works:

Step 1: J = 1 SHIFT = 0.
ASCII code = 49
K = 1 = 49-48
Use shifting procedure of line 10000
Build ARRAY(1) = 1 = 0 × 10 + 1

Step 2: J = 2 SHIFT = 0.
ASCII code = 50
K = 2 = 50-48
Use shifting procedure of line 10000
Build ARRAY(1) = 12 = 1 × 10 + 2

Step 3: J = 3
ASCII code = 46
Recognize special code for decimal point
set SHIFT = 10.

Step 4: J = 4 SHIFT = 10.
ASCII code = 51
K = 3 = 51-48
Use shifting procedure of line 10600
Build ARRAY(1) = 12.3 = 12 + 3/10
Multiply SHIFT by 10 : SHIFT = 100.

Step 5: J = 5 SHIFT = 100.
ASCII code = 53
K = 5 = 53-48
Use shifting procedure of line 10600
Build ARRAY(1) = 12.34 = 12.3 + 4/100
Multiply SHIFT by 10 : SHIFT = 1000.

Step 6: J = 6 SHIFT = 1000.
ASCII code = 32
Recognize blank- finish number, get set
for next number by incrementing NUMBER = 2, and
resetting ISIGN, LAST and SHIFT.

```
ASCII CODE   CHARACTER      ASCII CODE   CHARACTER
    48           Ø              7Ø           F
    49           1              71           G
    5Ø           2              72           H
    51           3              73           I
    52           4              74           J
    53           5              75           K
    54           6              76           L
    55           7              77           M
    56           8              78           N
    57           9              79           O
    58           :              8Ø           P
    59           ;              81           Q
    6Ø           <              82           R
    61           =              83           S
    62           >              84           T
    63           ?              85           U
    64           @              86           V
    65           A              87           W
    66           B              88           X
    67           C              89           Y
    68           D              9Ø           Z
    69           E
```

Introducing Multi-User Profile

The Small Computer Company
P.O. Box 2910
Fort Worth, TX 76113-2910
By Ivan Sygoda, Director, Pentacle

It all happened at once. Thanks to a generous grant from the Robert Sterling Clark Foundation, Pentacle was able to upgrade its computer installation ahead of schedule. Our two-drive Model 12 was transformed, like Cinderella, into a 512K Model 16B with a 12MB hard disk and a DT-1 data terminal. To boot (ha!), we also acquired the TRS-XENIX operating system and Multi-User Profile (26-6412, $499).

THE ONLY THING WE HAVE TO FEAR

Since I'm the official Pentacle computer nut, it was my job to get all this up and running as efficiently as possible. The hardware installation was done on a Friday. The Radio Shack service technician left me with a formatted hard disk, a pile of manuals and best wishes for a pleasant weekend. (It's the user's job to install TRS-XENIX and the applications programs.)

Was I nervous? Keyed-up is closer to the mark — like a runner before a big race. Here I was with computing power at my fingertips that would have cost a million dollars a decade ago, if it could be had at all. The pile of manuals began to look awesome.

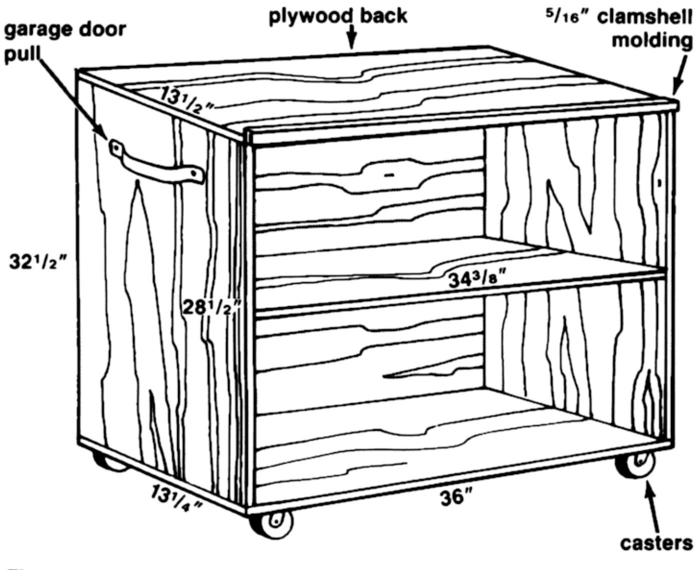


Figure 1: The Pentacle rolling computer manual cart

MANUAL LABOR

What did I do? I didn't go into the office at all on Saturday. Instead, I stayed home and built a rolling cart that could hold the manuals and be wheeled between terminals. It was great therapy, only taking a few hours and about $50 worth of lumber and hardware, and it's come in very handy. Similar carts from commercial office furniture houses begin at $200. (See figure 1.)

Sunday, then, was installation day. It turned out that I made a mountain out of a molehill. Installing TRS-XENIX was a breeze, and it took exactly one hour. All I had to do was to follow the instructions in the Operations Guide that came with the operating system. Connecting our DT-1 data terminal and setting passwords were equally thrilling operations for being uneventful. By mid-afternoon, I was ready to install Multi-User Profile.

WHERE TO BEGIN?

There's so much to say about Multi-User Profile that I don't know where to begin. "Begin at the beginning," says a little voice, and so I shall. Multi-User Profile is a data base management system for the Radio Shack Model 16 series computers (including upgraded Model 12s), which use the TRS-XENIX operating system. It takes the user-proven concepts and the modular design so successful in Profile Plus and Profile III Plus, and extends them (with enhancements that would take a book to describe) to a multi-user environment.

I could go on for months (and probably will) about its ease of use, its extraordinary power and flexibility, and its clever design. Multi-User Profile can extend, in myriad ways, your ability to understand and control the data with which you work. It's both simple and complex, like a superb stereo system that one can either turn on and listen to or fine-tune to hairline tolerances.

Just look at these "specs": Your data base can hold up to 16,000,000 records per file, spread over four floppy or hard disk drives. Each record can contain as many as 4,608 characters in up to 999 fields. The key segment can contain up to 512 of these characters. The optional data segment can contain up to 4,096 characters. A key field can be up to 512 bytes long. A data field can contain as many as 999 characters — over 12 lines of text! Any field can be accessed in sorting and selecting records, though it's more efficient to group frequently-used fields in the key segment. Field headings can be as long as 30 characters.

HIGHER MATHEMATICS

Profile Plus and Profile III Plus users have powerful math capabilities that enable them to add, subtract, multiply and divide the contents of fields. Multi-User Profile offers processing power a quantum leap above that. Not only can you perform the usual mathematical operations, you can perform them conditionally: that is, if some user-defined criteria are met. All the Boolean relationships are available to define these criteria (equals, does not equal, is greater than, etc.). This means, for instance, that you can have a sliding scale of discount or commission rates. Multi-User Profile will automatically consult the relevant totals and then perform the appropriate math. As if that weren't enough, Profile can also perform these operations on text strings! Here's a simple example: Profile can be instructed to read the two-character state code field and then multiply an amount by the correct sales tax.

NO STRINGS ATTACHED

Profile's ability to process character strings offers limitless opportunities for automating and manipulating text entries. Here's a random sampling of possibilities: You can automatically add commas after salutations; you can strip trailing blanks from field entries and concatenate them, with or without intervening punctuation; you can do global edits, such as changing every "Mrs." to "Ms."; you can automate the filling in of code fields by instructing the program to search other text or number fields for specified words, amounts or ranges.

An easy-to-use system of supplied and user-defined edits lets you format entries automatically. For example, keying in "8173903935" gives you "(817) 390-3935." You can also limit entries to pre-defined choices — "Y" or "N" for yes/no; "M" or "F" for sex, etc. You can move data between fields, from "bill to" to "ship to" if the latter field is empty, for instance. You can perform math on date fields and also do periodic or "batch" processing.

GOOD RELATIONS

One of the most powerful attributes of Multi-User Profile is its ability to perform "lookups." This means you can access, or look up, fields in up to ten other data bases at the same time. One obvious use of this would be to keep addresses in a mailing list data base or a contributions tracking file or whatever. Why store such information in more than one place? Retrieving information in this way lets you transfer it, change it, or simply use it in conditions for other processes. Thus, Multi-User Profile is a "relational" data base.

TURNING THE TABLES

Most of these various operations, which can be as sophisticated as you want, are defined on processing tables according to your needs. There are three such tables: (1) an "automatic" processing table which performs desired operations before each record is brought to the screen; (2) an "input" table that processes data as each new or revised record is recorded; and (3) an "output" table which is accessed as each record is used to print a report, label or form.

At first glance, these processing tables look a bit forbidding. Relax; it's not difficult to get the hang of them. First of all, you don't need to use them at all if your data base does not involve any math operations. If it does, or if you're transferring a data base involving addition, subtraction, multiplication and/or division from your original Model II/12 system, all you have to do is copy your formulas from your original math table. There's nothing new to learn. It's only when and as you become familiar with Multi-User Profile, and as your needs as a user develop, that you will want to become familiar with the intricacies of the processing tables.

Needless to say, I became instantly fascinated with the possibilities. Each of the three types of tables can be up to 200 elements long. Filling them in is somewhat like writing a program in BASIC, but easier. There are only a few key words to learn and some syntax rules to follow. What you get for your efforts is total control over all input and manipulation of your data. The operator can be directed from screen to screen during data entry, prompted for this, prevented from doing that, spoon-fed the third.

The screen creation program even lets you define the path the cursor will take during data entry and updating. In the coming months, we'll study the use of these processing tables in some detail. They constitute one of the keys to Multi-User Profile's great flexibility. They also enable you to make profoundly elegant data base operations.

BACK TO SQUARE ONE

But I'm getting ahead of myself. The first step is to get Profile itself up and running. Multi-User Profile comes as a set of eight inch floppy diskettes containing programs to be transferred to your hard disk under TRS-XENIX. The process is quite simple. Boot XENIX and log in as root or superuser. When the root prompt ("#") appears, type "install" and then (ENTER). The installation menu appears. Answer the prompts and change disks when asked to do so. The screen tells you when the process is complete, at which point you can either install another applications program or return to XENIX. That's all there is to it.

If you want to create a data base, log off root by typing (CTRL) (D) and log in under your usual ID. It is not a good idea to create Profile data bases in superuser mode for a variety of reasons.

Many users will move up to Multi-User Profile from Profile Plus installations. Existing data bases can be converted to Multi-User at any time after the installation process. Conversion is also done from root. Type "u" and (ENTER) at the "#" prompt. This calls up the Profile utility menu, whose choices include not only the conversion program but also ones for backing up your data bases onto floppies and restoring them to the hard disk drive. Choice one is the conversion program.

You are first prompted to answer a few standard questions (the name of the Profile Plus data base, for example, the name of the new data base, and which floppy drive you will use). The program then transfers certain files from TRSDOS to XENIX, prompting you to change disks when appropriate.

When the relevant files are transferred, the conversion process begins. The original key segment becomes the new key segment. Any and all Profile Plus data segments are combined into Multi-User Profile's single data segment. The map file, which keeps track of the fields and respective lengths you initially defined, is converted. Since Multi-User Profile has a different system of defining field types, your original specifications (alpha field, number-only field, etc.) are converted to the "*" default. The conversion program also converts all your screen formats, which is a nice time- and effort-saving feature. They'll look exactly the same on the Model 16B as they did on your Model II or 12. Report, label and forms formats cannot be converted, however. These will have to be re-created under Multi-User Profile.

ON YOUR MARK, GET SET, GO!

That's all there is to it. In less than an hour, your Profile Plus data base is ready for you and up to two other users to access, consult, update. You can build up to the full power of Multi-User Profile at your own pace. I think it's incredible that records I originally keyed into my Profile data base two years ago on my floppy-based Model III are now nestled in my 12MB hard disk TRS-XENIX 16-bit multi-user, multi-tasking, fawn white 16B, without my ever having to re-enter data! And each step along the way was logical, manageable and affordable.

I'll be telling you more about Multi-User Profile in the coming months, but we won't forget the many users of Profile III/4 and II/12 either. You're still first-class citizens.

PROFILE Editor's Note: This is Mr. Sygoda's sixteenth article in a series of 'how-to' Profile articles. Other articles in the series will be published over the next few issues in this column. We hope that you enjoy this feature, and we look forward to your comments and questions on Profile.

Pentacle is a New York City-based non-profit service organization specializing in administrative services for performing art groups.

Hearts for Valentine's Day

Janice and Dianna Timmann
P.O. Box 680
Phillipsburg, NJ 08865

My fourteen year old daughter, Dianna, wrote this program called HEARTS. I hope your readers enjoy it.

```
1Ø '       HEARTS FOR VALENTINE'S DAY
2Ø '           BY DIANNA TIMMANN
3Ø '       PHILLIPSBURG, N.J.  Ø8865
4Ø '        TRS-8Ø MODEL I, III, IV
5Ø CLS
     : CLEAR 5ØØ
6Ø PRINT "YOUR PICTURE CAN HAVE YOUR NAME, OR THE
     NAME OF A SPECIAL FRIEND"
7Ø INPUT "YOUR NAME PLEASE "; A$
     : L = LEN(A$)
     : DIM T$(114)
8Ø FOR D = Ø TO INT(57/L)
     : FOR I = 1 TO L
     : T$(D * L + I) = MID$(A$,I,1)
     : NEXT I
     : NEXT D
9Ø  ' FOR A VIDEO DISPLAY
1ØØ ' CHANGE LINES 9Ø, 11Ø, & 12Ø
11Ø ' CHANGE LPRINT TO PRINT
12Ø C = Ø
13Ø A1 = 1
     : P = 1
     : C = C + 1
     : IF C = 42 THEN 32Ø
14Ø LPRINT " "
15Ø READ A
     : A1 = A1 + A
     : IF P = 1 THEN 17Ø
16Ø FOR I = 1 TO A
     : LPRINT " ";
     : NEXT I
     : P = 1
     : GOTO 18Ø
17Ø FOR I = A1 - A TO A1 - 1
     : LPRINT T$(I);
     : NEXT I
     : P = Ø
```

(Continued on page 42)

Auto Log On to Dow Jones on the Model 100

Donald Parson
4 Driftwood Landing
Delray Beach, FL 33441

I have rewritten the Auto Log-on program which appears on page 199 of the Model 100 Owner's Manual to enable the user to input the requests directly.

You will recall that in the original version the requests had to be written into the program before it was run. This revision will allow up to five requests to be entered from the keyboard, but that number could be increased.

```
5 MAXFILES=3
1Ø INPUT "QUOTE:  ";A$
2Ø INPUT "LAST ITEM  ";M$
3Ø IF M$="Y" THEN 15Ø
4Ø INPUT "QUOTE:  ";B$
5Ø INPUT "LAST ITEM  ";N$
6Ø IF N$="Y" THEN 15Ø
7Ø INPUT "QUOTE:  ";C$
8Ø INPUT "LAST ITEM  ";O$
9Ø IF O$="Y" THEN 15Ø
1ØØ INPUT "QUOTE:  ";D$
11Ø INPUT "LAST ITEM  ";P$
12Ø IF P$="Y" THEN 15Ø
13Ø INPUT "QUOTE (final item) ";E$
15Ø Q$=","
155 ST$=CHR$(19)
16Ø PH$="(telephone#)<?:A=DOW1;;?WDJNS^M?
    @(password)^M>"
17Ø M=VARPTR(PH$)
18Ø AD=PEEK(M+1)+(PEEK(M+2)*256)
19Ø CALL 212ØØ
2ØØ CALL 21293,Ø,AD
21Ø CLS
22Ø OPEN"MDM:7I1D" FOR INPUT AS 1
23Ø OPEN"MDM:7I1D" FOR OUTPUT AS 2
24Ø OPEN"QUOTE.DO" FOR APPEND AS 3
25Ø Z$=INPUT$(1,1)
26Ø IF Z$<>ST$ THEN 25Ø
27Ø PRINT#3,DATE$;" ";TIME$
28Ø PRINT "STARTING QUOTES REQUEST"
32Ø PRINT#2,Q$;A$
33Ø GOSUB 4ØØØ
34Ø IF M$="Y" THEN 5ØØ
35Ø PRINT#2,Q$;B$
36Ø GOSUB 4ØØØ
37Ø IF N$="Y" THEN 5ØØ
38Ø PRINT#2,Q$;C$
39Ø GOSUB 4ØØØ
4ØØ IF O$="Y" THEN 5ØØ
41Ø PRINT#2,Q$;D$
42Ø GOSUB 4ØØØ
5ØØ PRINT "SIGNING OFF"
51Ø ST$=CHR$(7)
52Ø PRINT#2,"DISC"
53Ø GOSUB 4ØØØ
54Ø CLOSE
55Ø CALL 21179
56Ø END
4ØØØ Z$=INPUT$(1,1)
4Ø1Ø IF Z$=ST$ THEN RETURN
4Ø2Ø PRINT#3,Z$;
4Ø3Ø GOTO 4ØØØ
```

Communications Corner

By Al and Dru Simon

Welcome back to our corner! The topic under discussion in this month's issue is utilities so we thought we would discuss using a word processor as a utility for a terminal program, as well as give you some utilities you can use with almost every data service or bulletin board.

A word processor used for editing an ASCII file can save the user a lot of time and trouble when downloading or uploading programs to or from a data service. The main function of the word processor would be to input the ASCII file (all current versions of SCRIPSIT permit this in one form or another), make changes or corrections, and then resave the altered file.

For example, if you are visiting a BBS which caters to a different type of BASIC than the one you are using, you may have to make global changes (that is, replace some character or amount of characters in the program at all its occurrences at once) in order to make it compatible with your own equipment.

For another example, you might download a print file which is too wide for your printer. With a word processor you could truncate each line very readily. There are endless editing jobs that SCRIPSIT and other word processors can accomplish for you.

With this in mind we have included two small programs in this month's column just for you. They are listed below, and when used in conjunction with a word processor they will allow you to capture or send machine language programs in an ASCII format.

The first program is called CMDCONV/BAS. It will take a machine language file, (also known as /CMD file) or any similar file and convert it to an ASCII representation of that file which can then be transferred with complete ease since all online sources accept ASCII files.

```
Ø    REM CMDCONV/BAS
1Ø   REM Courtesy of Commnet-8Ø Riverside, Ca
2Ø   REM 714 359 3189
3Ø   'USE THIS PROGRAM TO CONVERT MACHINE LANGUAGE
      FILES INTO
4Ø   'ASCII FILES FOR TRANSMISSION
5Ø   '
6Ø   CLEAR 2ØØØ
     : DEFINT A-Z
     : DIM R$(8)
7Ø   CLS
     : PRINT128, "PLEASE ENTER 'CMD' FILE NAME ";
     : INPUT I$
8Ø   PRINT@256, "PLEASE TYPE OUTPUT FILE NAME ";
     : INPUT O$
9Ø   OPEN "R",1,I$
     : OPEN "O",2,O$
1ØØ FIELD 1,32 AS R$(1), 32 AS R$(2), 32 AS R$(3), 32
      AS R$(4), 32 AS R$(5), 32 AS R$(6), 32 AS R$(7),
      32 AS R$(8)
11Ø CLS
12Ø FOR I=1 TO LOF(1)
13Ø PRINT@256, "CONVERTING RECORD NUMBER";I
14Ø GET 1,I
15Ø FOR J=1 TO 8
16Ø L$=""
17Ø FOR M=1 TO 32
18Ø N=ASC(MID$(R$(J),M,1))
19Ø N1=N/16
2ØØ PRINT@6Ø," *"
21Ø GOSUB 34Ø
22Ø N$=C$
23Ø N1=N-16*N1
24Ø GOSUB 34Ø
25Ø N$=N$+C$
26Ø L$=L$+N$
27Ø NEXT M
28Ø PRINT#2,L$
29Ø NEXT J
3ØØ NEXT I
31Ø CLOSE
32Ø CLS
     : PRINT CHR$(23)
     : PRINT "CONVERSION COMPLETE "
33Ø GOTO 33Ø
34Ø IF N1>9 THEN C$=CHR$(55+1)
     : PRINT@6Ø,"  "
     : RETURN
35Ø C$=CHR$(48+1)
     : PRINT@6Ø,"  "
     : RETURN
```

ASCONV/BAS is the second program, and this will take a machine language file which has been turned into an ASCII file (for example, one you might have downloaded from a Bulletin board or CompuServe or the like) and convert it back into a machine language program or /CMD file.

```
1    REM ** ASCONV/BAS                              **
1Ø   REM ** COURTESY OF COMMNET-8Ø RIVERSIDE, CA **
2Ø   REM ** A/C 714 359 3189                        **
3Ø   REM ** THIS PROGRAM IS A PERFECT COMPLEMENT **
4Ø   REM ** TO CMDCONV/BAS AND IS USED TO CONVERT**
5Ø   REM ** ASCII ENCODED MACHINE LANGUAGE FILES **
6Ø   REM ** BACK INTO /CMD FILES (HEX FILES)      **
7Ø   CLS
      : CLEAR 1ØØØ
      : PRINT@256,;
8Ø   LINEINPUT "NAME OF 'ASCII' FILE TO BE CONVERTED
      >";I$
9Ø   PRINT
      : LINEINPUT "NAME OF MACHINE LANGUAGE OUTPUT
      FILE  >";O$
1ØØ OPEN "I",1,I$
      : OPEN "O",2,O$
11Ø LINEINPUT#1,A$
      : IF A$="" GOTO11Ø
      : A$=RIGHT$(A$,LEN(A$)-INSTR(A$,"'"))
12Ø D$=""
      : PRINT@512,A$
      : FOR X=1 TO LEN(A$) STEP 2
13Ø C$=MID$(A$,X,2)
      : D=ASC(C$)
      : E=ASC(RIGHT$(C$,1))
      : D=(D-48)+(D>57)*7
```

```
        : E=(E-48)+(E>57)*7
        : M=D*16+E
        : D$=D$+CHR$(M)
        : PRINT CHR$(-M*(M>31 AND
        M<97-46*(M<32ORM>96));" ";
        : NEXT X
140 PRINT"2,D$;
        : IF NOT EOF(1) THEN 110 ELSE CLOSE
150 PRINT "FINISHED"
```

WHY ARE BASICS DIFFERENT?

For your reference we'd like to explain why BASIC for one machine and system will not work for another machine and system unless transferred in ASCII.

Your BASIC programs are passed through an interpreter, either ROM or RAM resident, so that each reserved word (for example the word PRINT) is encoded into a number called a TOKEN. That encoding scheme (or table) tends to vary from BASIC to BASIC; therefore, if you try to transmit a BASIC file which is not in ASCII, you will be sending the TOKEN number. In other words, while N may be the TOKEN number for PRINT in TRS BASIC, it may well represent the word RUN in IBM's BASIC.

THE NEWS FROM PLUMB

Here's some warm stories from our friends at PLUMB . . .

>General Electric has joined the growing number of companies trying to entice potential employees through their home computers. GE invites job-seekers with technical or engineering skills to file their resumes directly into the company's GENSTAR computer. To access the system call 301-340-4800 for 300 baud or 301-340-5096 for 1200 baud. Type "H" several times, then enter "GGV44101,GE" for the user code and hit ENTER. According to GE, the resume will be stored at the company's Valley Forge, Pa. headquarters and electronically transmitted to GE Space Systems facilities across the country.

>The story of TIMECOR—The International Modem Exchange Corporation—may well be the ultimate bulletin board success story. "It all started out as a lark," said Bud Napier, the man who started the system a couple of years ago . . . in his Boston apartment.

As the board's popularity increased, callers found it harder to log on. Eventually Napier installed a multiplex system that lets five callers at a time share access to the board, but that still didn't satisfy demand so Napier took the next logical step—he went commercial. He loaded an expanded version of the Boston board on a mainframe computer in McLean, Va. and plugged it into the Telenet system, which offers local-number access to commercial databases in the larger U.S. cities.

TIMECOR itself is a fruit salad of computer goodies stocked on some 30 boards. Callers to the Boston board 617-720-3600 get an extensive tour of the premises and access to some of the general interest boards. Telenet callers can check out the mainframe version of TIMECOR by typing C 202141 at the prompt. Enter "guest.timecor" for user name and "guest" for the password. There is a charge for the service.

As usual our thanks and greetings to Ric Manning at Riverside Data Inc for his permission to reprint these items. PLUMB can be contacted by writing to P.O. Box 300, Harrods Creek, KY 40027 (502) 228 3820 or by leaving messages for CompuServe account 72715,210 or account STQ007 on The Source.

THE CORNER MAILBOX

Dear Al & Dru Simon,

I read your article in . . . <the> September issue with keen interest. I went out and bought the Art Gallery program and the video selector and cable. I put everything together and couldn't get titles to transfer to video tape.

Would GREATLY APPRECIATE your telling me what I did wrong.

Respectfully,
Philip Schneider
Brooklyn, NY

Dear Philip:

As you will find on page 9 of your manual, when you want to save your pictures to videotape, have the picture loaded onto the screen of the Color Computer by pressing 3 at the main menu which will load your stored picture. Here's what you do next:

1. *If you have not included MOVING text with your picture then turn on your VTR, putting it in RECORD/PAUSE position (or VIDEO INSERT/PAUSE if you have that option).*
2. *Next, turn the selector so that the VTR recording is to Auxiliary 2, and TV is to VTR.*
3. *Make sure the VTR/TV switch on the VTR itself is in the VTR position.*
4. *Check the picture, release the PAUSE for the duration you wish the title to last, and VOILA!*

If you have mixed MOVING text with your picture after loading as above, go to RECORD/PAUSE on your VTR, release the PAUSE, press number 5 on the main menu to start the text moving, press the space bar when you wish to stop the text, and continue as above.

That method works just perfectly for us. Good luck!

Dear Al,

I recently purchased an Extended BASIC Color Computer, mainly as an inexpensive introduction to computers for my children and myself.

I happened to mention to my wife that it would be nice to get a print-out of the programs I was trying to write. She bought me a printer with a serial interface board and that was the start of my problems.

I am not at all knowledgeable about electronics or computers so I was confused by the fact that the Radio Shack cable #26-3014 had four (4) pins and the printer connector had seven (7) pins.

The man working on my problem kept plugging and unplugging the cable with the power to both the computer and the printer switched on (contrary to the instruction manuals). Maybe this is not a problem, but I hate to take chances.

In addition to the cable problem I have no idea what the word "handshake" means in relation to a computer.

I feel competent enough to modify the cable connector after having read your articles.

Yours very truly,
Don Love
Houston, Tx

Dear Don;

The printer you bought is using handshaking protocols which are not supported by only four pins. The 4 pins you receive are READ DATA, SEND DATA, SIGNAL GROUND, and CHASSIS GROUND.

Handshaking means that the printer is looking for a signal to be sent to it to tell it to GET READY; then it's sending a signal back saying "I'M READY"; and then the text is transmitted to the printer until it sends a message which says "MY BUFFER IS FULL, WAIT A MINUTE" or until the end of file is reached (whichever comes first). Most of these signals are not available with the four pin cable that you have, such as the RTS and DTR and therefore I would suggest that you contact the manufacturer of the printer and tell him that you have just 2,3,7, and 8 available and ask his advice as to what pins he would suggest that you tie together on the other end.

Also, your feeling is correct; it is NOT a good idea to connect and disconnect equipment with the power on, on either side!

Bugs, Errors, and Fixes

LETTER CORRECTIONS TO SOFTWARE

Following are brief descriptions of problems to be fixed in specific software packages and the dates of letters that were sent to all registered owners containing the corrections to the problem.

If you are a registered owner of a software package described below and have not received the letter detailing the software problem and its correction, then contact your nearest Radio Shack Computer Center or Computer Customer Services. If you have not registered and are a legal owner of the software, you need to register by sending in the card that came with the package.

Model I/III/4

SUPERSCRIPSIT (26-1590 Vers. 01.02.01)

These corrections, to Model III SuperScripsit only, prevent the printing of multiple copies of the same lines in very long documents.

Letter dated: September 8, 1983

SUPERSCRIPSIT (26-1590 Vers. 01.02.02)

A brief power outage or an improper exit from the program can cause a screen to show "garbage" or to lock up. This happens particularly when block action commands are attempted. The corrections reduce the chance of the situation happening.

Letter dated: September 20, 1983

Model II/12/16

TIME ACCOUNTING (26-4520 Vers. 01.00.00)

Updates programs dealing with deletion of clients, billing worksheets, statements, and simulations, and the index handling of the client file.

Letter Dated: August 4, 1983

GENERAL LEDGER (26-4601 Vers. 01.00.00)

Changes programs concerned with transaction processing and statement print.

Letter Dated: August 29, 1983

ACCOUNTS RECEIVABLE (26-4604 Vers. 01.00.00)

Modifies programs dealing with invoices and salesmen reports.

Letter Dated: August 22, 1983

ACCOUNTS PAYABLE (26-4605 Vers. 01.00.00)

Corrects inaccurate posting and distribution of Accounts Payable transaction processing.

Letter Dated: August 22, 1983

ORDER ENTRY/ICS (26-4607 Vers. 01.00.00)

Changes concerned with deletion of orders, order processing, rounding errors, and price selection with the F1 key.

Letter Dated: August 4, 1983

ICS-HARD DRIVE (26-4802 Vers. 01.00.00)

Problems with transaction processing have been corrected.

Letter Dated: September 7, 1983

OTHER CORRECTIONS

The following changes and corrections are optional and provided for your information. If you have an applications program which is working correctly, you should probably NOT make any changes to it. If you feel that changes should be made, but you do not feel qualified to make the change yourself, contact your local Radio Shack Computer Center or Expanded Computer Department for assistance. If you do not have access to one of these stores, then you may want to call Computer Customer Service in Fort Worth for assistance.

Color Computer

MC-10 OPERATION & LANGUAGE REFERENCE MANUAL (26-3011)

Page 18, Part 3

Reads: Graphics characters generated from the keyboard can be very useful when a graphics printer or graphics pad (such as the TRS-80 Model GT-16 x-pad) is connected to the MC-10.

Change: Graphics characters generated from the keyboard can be very useful when a graphics printer is connected to the MC-10.

Model I/III/4

MODEL 4 TRSDOS AND BASIC INTERPRETER (26-0313)

In MAILLIST 01.01.00 on TRSDOS 06.01.00 or 06.01.01, change statement 1670 and 180 to read:

```
1670 GOSUB 2820
     : IF NOT REPLY% THEN 1760
180 PRINT@(12,31), "Version 01.01.01"  'Print Version
```

Peripherals

DMP-2100 OPERATIONS MANUAL (26-1256)

Change line 200 on page 53 to read:

```
200 DATA -5,0,-5,1,-5,2,-5,4,-5,8,-5,16,-5,
     32,-5,39,64,127,32,32
```

Administrative Series for TRS-80 Model 12/II

A new series of administrative software products from Radio Shack is designed to help schools collect, store, retrieve, and print out basic student information, attendance data, student grades, and scheduling information.

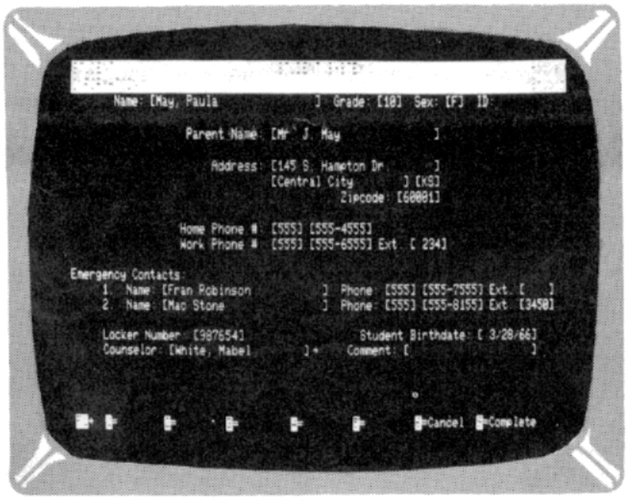
The Radio Shack School Administrative Software Products series is a comprehensive administrative tool in four modules: Student Information System, Attendance System, Grade Reporting System, and Individual Scheduling System. Together, these systems will interact to meet most of the student records needs of schools. The Student Information System was released fourth quarter 1983. The Attendance System and the Grade Reporting and Individual Scheduling systems will be available in early 1984.

Programs can be used with a TRS-80 Model 12 computer with one floppy disk drive and a hard disk, a Model 12 computer with two floppy disk drives, or a Model II computer with one floppy disk drive and one hard disk. A line printer is also needed to produce reports. For all packages except the Student Information System, a Radio Shack CR-510 Card Reader is needed to read attendance, grades, and schedule cards.

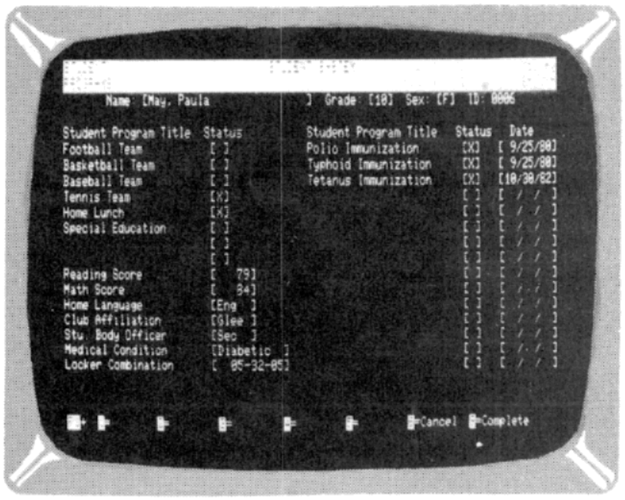
THE STUDENT INFORMATION SYSTEM

The Student Information System (Radio Shack Cat. No. 26-2729) is the foundation of the data base. The Student System is a collection of integrated computer programs that allow you to collect, organize, retrieve, and print out information on the student population at your school.

The heart of the Student Information System is a list of student profiles that you construct as students enter or leave your school. Each profile includes basic information in which most schools are interested—for example, parent names and addresses, and emergency contacts (see Screen 1).


```
      Name: [May, Paula                 ]  Grade: [10]  Sex: [F]  ID:

                   Parent Name: [Mr. J. May             ]

                       Address: [145 S. Hampton Dr.     ]
                                [Central City       ] [KS]
                                          Zipcode: [60001]

                  Home Phone #: [555] [555-4555]
                  Work Phone #: [555] [555-6555] Ext. [ 234]

Emergency Contacts:
     1.  Name: [Fran Robinson           ]  Phone: [555] [555-7555] Ext. [    ]
     2.  Name: [Mac Stone               ]  Phone: [555] [555-8155] Ext. [3450]

     Locker Number: [9876543]               Student Birthdate: [ 3/28/66]
     Counselor: [White, Mabel        ] *   Comment: [                    ]
```

A profile can also contain information that you choose to enter based on your school's unique needs. Such information might include items like an individual's extracurricular activities, health and immunization status, or scores on standardized tests. Screen 2 shows some of the special information categories that a school might set up and how this information might be recorded for one student:


```
      Name: [May, Paula                 ]  Grade: [10]  Sex: [F]  ID: 0006

Student Program Title  Status          Student Program Title   Status   Date
Football Team          [ ]             Polio Immunization       [X]   [ 9/25/80]
Basketball Team        [ ]             Typhoid Immunization     [X]   [ 9/25/80]
Baseball Team          [ ]             Tetanus Immunization     [X]   [10/30/82]
Tennis Team            [X]                                      [ ]   [  /  /  ]
Home Lunch             [X]                                      [ ]   [  /  /  ]
Special Education      [ ]                                      [ ]   [  /  /  ]
                       [ ]                                      [ ]   [  /  /  ]
                       [ ]                                      [ ]   [  /  /  ]
Reading Score          [    79]                                 [ ]   [  /  /  ]
Math Score             [    84]                                 [ ]   [  /  /  ]
Home Language          [Eng   ]                                 [ ]   [  /  /  ]
Club Affiliation       [Glee  ]                                 [ ]   [  /  /  ]
Stu. Body Officer      [Sec   ]                                 [ ]   [  /  /  ]
Medical Condition      [Diabetic  ]                             [ ]   [  /  /  ]
Locker Combination     [  05-32-05]                             [ ]   [  /  /  ]
```

Once student information has been entered, many options are available. Standard print-outs such as mailing labels, folder labels, and lists of enrolled and dropped students, are provided for in the system. In addition, a "special inquiry" feature lets you construct special reports. For example, you might want to print a list of the names and complete addresses of all students in the marching band (if "marching band" is one of the information categories you added to the system).

Records in the system can be updated at any time, and a special "promotion and graduation" feature allows you to move all students in the system ahead one grade level (exceptions can be made for students who will not be promoted).

Since the Student Information System stores the basic enrollment information for students in your school, you need this system in order to use the Attendance, Grade Reporting, or Scheduling systems.

THE ATTENDANCE SYSTEM

The Attendance System (Cat. No. 26-2730) helps you keep track of attendance at your school for those students whose names have been entered into the Student Information System. The teacher records absences on special cards; then school personnel can quickly feed the information on these cards into the computer using a CR-510 Card Reader attached to the computer. The Attendance System gives you the option of taking attendance on either a period-by-period or daily basis, and it calculates monthly attendance statistics. The system can also keep track of how absences are resolved.

THE GRADE REPORTING SYSTEM

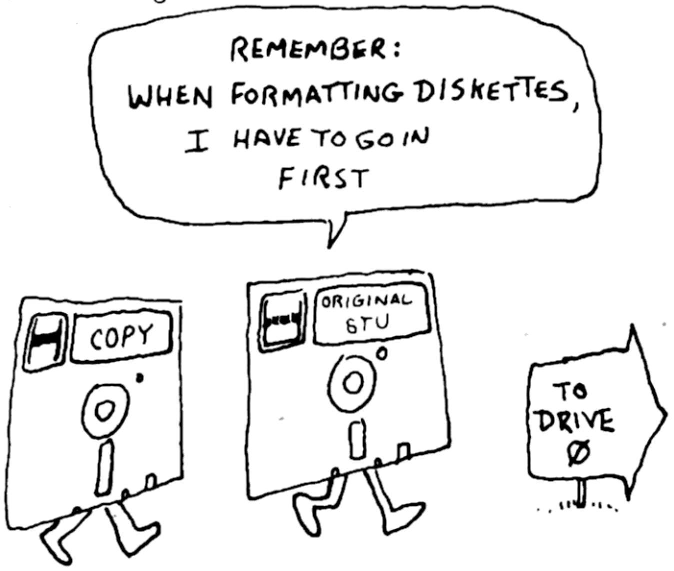
The Grade Reporting System (Cat. No. 26-2731) helps schools produce grade reports. Grades for each class roster can be recorded on special machine-readable cards, then printed out on report cards. Each student's grade can include an academic mark, a citizenship mark, absences, and the number of credits earned in the course. The evaluation can also be supplemented by up to three "teachers' comments" chosen from a list of 36 comments which the school's faculty can itself make up.

THE INDIVIDUAL SCHEDULING SYSTEM

The Individual Scheduling System (Cat. No. 26-2732) helps you plan your master schedule, schedule students, produce student schedule cards, and continue to change individual student schedules during the term whenever necessary—all with a minimum of paperwork.

HOW YOU CAN USE THE RADIO SHACK SCHOOL ADMINISTRATIVE SOFTWARE PRODUCTS

The four modules in this series can be used by schools of any size. To make it easy for any teacher or administrator to use the programs, each package contains a training manual with a demonstration section. Helpful drawings and a friendly tone in the manual help create an experience that is non-threatening.



To learn how to obtain the Student System, the Attendance System, or the other packages as they are released, check with your local Radio Shack store or Computer Center. Radio Shack also has 25 Regional Educational Coordinators (listed below) located around the country to help local schools and districts meet their computing needs.

RADIO SHACK REGIONAL EDUCATIONAL COORDINATORS

Western Educational Marketing Manager

Scott Bowers
Radio Shack
1400 One Tandy Center
Fort Worth, TX 76102
817/390-3910

Chicago Region
Donna Comber
Radio Shack
679 W. North Avenue, Suite 204
Elmhurst, IL 60126
312/833-1010

Columbus Region
Sonny O. Compton
Radio Shack
4343 Williams Road
Groveport, OH 43125
614/836-3980 X252

Dallas Region
Sid Agent
Radio Shack
2588 Royal Lane
Dallas, TX 75229
214-258-1530

Denver Region
Rosemary Shiels
Radio Shack
5890 W. 44th Avenue
Denver, CO 80212
303-424-4467

Detroit Region
Celia Magro
Radio Shack
29548 Southfield Rd., Suite 200
Southfield, MI 48076
313/552-9290

Houston Region
Jim Savoie
Radio Shack
7119 San Pedro
San Antonio, TX 78216
512/341-2622

Kansas City Region
Pat Nickens
Radio Shack
9590 Quivera St.
Linexas KS, 66215
913/541-1088

Los Angeles Region
Terry Kramer
Radio Shack
14126 E. Firestone
Santa Fe Springs, CA 90670
213/921-2659

Louisville Region
Penny Shattuck
Radio Shack
553 N. Court, Suite 175
Palatine, IL 60067
312/991-2275

Sacramento Region
Steve Terhune
Radio Shack
1337 Howe Avenue, Suite 201
Sacramento, CA 95826
916/920-1298

San Diego Region
Bob Norman
Radio Shack
12821 Knott Street
Garden Grove, CA 92641
714/894-1371 X255

San Francisco Region
Pat McDonald
Radio Shack
1291 E. Hillsdale, Suite 211
Foster City, CA 94404
415/574-1708

Seattle Region
Annie Gillvan Green
Radio Shack
5030 Roosevelt Way, N.E.
Seattle, WA 98105
296/527-8607

Eastern Educational Marketing Manager

Ron Moore
Radio Shack
1400 One Tandy Center
Fort Worth, TX 76102
817/390-3527

Albany Region
Don Francolino
Radio Shack
39 S. Main Street
West Hartford, CT 06017
203/232-4529

Atlanta Region
Art Williams
Radio Shack
241 W. Wieuca Road, N.E.
Atlanta, GA 30342
404/255-9438

Boston Region
Dick Callahan
Radio Shack Education Division
250 Granite Street
Braintree, MA 02368
617/848-0780

Charlotte Region
Jerry Proffitt
Radio Shack
5701 W. Park Drive, Suite 205
Charlotte, NC 28210
704/527-7442

Miami Region
Pete Lenkway
Radio Shack
4360 N. Federal Highway
Ft. Lauderdale, FL 33308
305/493-5021

New York Region
Mick Nuspl
Radio Shack
211 North Avenue
New Rochelle, NY 10801
914/576-7065

New York Region
Bob Sochor
Radio Shack
6 East 39th Street, Room 401
New York, NY 10016
212/696-9800

Philadelphia Region
Maddie Lesnick
Radio Shack
Crest Plaza
Cedar Crest Blvd. & U.S. 22
Allentown, PA 18104
215/395-5755

Tampa Region
Paul Hoagland
Radio Shack
5729 N. Hoover Blvd., Suite 230
Tampa, FL 33614-5373
813/886-2974

Philadelphia Region
Don Wallick
Radio Shack
Rt. 130 & Cinnaminson Avenue
Cinnaminson, NJ 08077
609/829-6911

Pittsburgh Region
Winston Ferrell
Radio Shack
1679 Washington Road
Pittsburgh, PA 15228
412/833-1918

Washington D.C. Region
Kevin Hogans
Radio Shack
Computer Center #7267
48 W. Ridgley Road
Lutherville, MD 21093
301/561-2472

Disk Sorting on the TRS-80

Richard Earp, Ph.D.
Associate Professor of System Science.
University of West Florida
12/128
Pensacola FL 32514

INTRODUCTION

There have been several programs and articles published that deal with sorting. (1,2) Unfortunately, these sort articles deal mostly with array sorting and not with a major problem of the small computers. Array sorts are fine for in-core (in memory) sorts of strings and numbers, but difficulties with disk I-O speed and with memory arise when one wants to sort long files. This article deals with the sorting of files of various lengths and record sizes on disk. Three approaches to sorting files will be presented—the addition sort/merge, key sorts, and partitioned sorts. The observations made here spring from the experience of writing software to efficiently sort files (3). In the examples that follow, the direct access (random) file mode is used. Sorting sequential files could be done similarly, but an intermediate, random file would be necessary for the partitioned and key sorts.

BACKGROUND

Before attacking the real problems of disk sorting (disk I-O and memory), it must be understood that some algorithm for in-core sorting is necessary. A couple of examples of file sorting situations which are inefficient, albeit instructional, will show the problem of disk sorting and the necessity for the sort algorithm.

To sort a file, one could use the in-core algorithm and substitute disk I-O for array indexing but this is inefficient because of slow disk I-O. For example, instead of . . .

```
500 REM -- PART OF SORT ALGORITHM
510 IF A$(I) < A$(J) THEN 540
520 REM SWAP
530 H$ = A$(I)
     : A$(I) = A$(J)
     : A$(J) = H$
540 REM -- REST OF THE SORT ALGORITHM
```

One might substitute (e.g., FIELD 1,20 AS R$) . . .

```
500 REM -- DISK SORT
502 GET 1,I
     : REM -- THE Ith RECORD FROM FILE 1
504 RI$ = R$
506 GET 1,J
     : REM -- THE Jth RECORD
508 RJ$ = R$
510 IF RI$ < RJ$ THEN 540
520 REM SWAP
530 LSET R$ = RI$
     :PUT 1,J
535 LSET R$ = RJ$
     :PUT 1,I
540 REM -- REST OF SORT
```

Using a sort algorithm to read, compare and re-write records two-at-a-time is very slow because disk I-O is very slow and the less disk I-O, the better. Therefore, more efficient sorts use some technique of bringing data into memory, sorting, and then returning the data to disk. For example, for a small file, the following segments would be appropriate . . .

```
390 REM early housekeeping is omitted
400 FIELD 1,20 AS R$
410 L1 = LOF(1)
420 FOR J = 1 TO L1
430 GET 1,J
440 A$(J) = R$
450 NEXT J
```

500 . . . 540 contains a sort for A$.

```
600 FOR J = 1 TO L1
610 LSET R$ = A$(J)
620 PUT 1,J
630 NEXT J
```

This program will only work for small files which have the sort key beginning in the first byte of the record. If the sort key were elsewhere, A$ would be a concatenated string of MID$s which, after sorting, could be rearranged to re-file using MID$s again. This program gives the idea of what the ideal sort contains—one pass through the file to gather data, sorting, then one pass to re-write the file.

There are various sorting mechanisms which can be written in BASIC which might be used in lines 500-540. However, because interpretive BASIC is much slower than machine language, it is preferable to use an assembled algorithm if the size of the array to be sorted is significant (if there are only 5 or 10 entities, it might be easier to use a BASIC subroutine). In the Model III, one has the CMD "O" sort built in. With the Model II, 12 and 4 a bit more work is required, because the user has to load his own sort routine. The sort routine for array sorts as described in the *TRS-80 Microcomputer News,* July 80, works quite well for non-Model III users (4).

SORTING DISK FILES

To discuss disk sorting, there are two givens. One, a fast machine language sort is best for the in-core part of the sort, if the length of the array to be sorted is significant. Two, minimal disk I-O is always sought because disk I-O is a relatively slow process. To sort disk files, three disk sort mechanisms will be presented: (a) a limited addition-merge sort, (b) a key only sort, and (c) a partitioned sort. Program segments will accompany the latter two mechanisms. All of the techniques require sufficient disk space for a scratch (temporary) file.

THE LIMITED ADDITION-MERGE SORT

One way to build a sorted file is to limit the additions to the file to a small number of records. This filing technique illustrates the minimal disk I-O virtue. It is useful when only a small fraction of a file is out of order. The small number of records that are added to the file are kept in memory before they are added to the file. (In fact, the file need not be opened until the records to be added are accumulated and sorted.) After 10 or so records are accumulated in an array, they are sorted in memory and then merged into the file itself. It is easier to merge than to sort, because the position of the records being merged can be determined by one pass through the sorted array of additions and one read-pass through the file. The safest way to add records and to recreate the file is to create a sorted "scratch" file and then either to recopy the scratch file onto the original file space or to keep the scratch file as the new version of the sorted file. The programming sequence is summarized as follows..

1. Read the data to be added into an array A$.
2. Sort the array A$.
3. Open the sorted file.
4. Read the sorted file and the sorted array A$ one record at a time and file the lesser in a SCRATCH file.
5. Copy the SCRATCH file over the original file or rename the SCRATCH file and keep it as the newest version of the data.

This technique results in slow data entry, but it keeps the user from having to face the problem of sorting an entire file. To build the file, one might limit the number of additions to say 10 records or less. Even if the file was 250 bytes long, all 10 records could be kept in memory and sorted. When the first 10 records had been used to create the original file, the next 10 would be merged with the first 10. Each time, the new data is read into memory, sorted, and merged with the old data to form a longer file.

If desired, generations of files could be kept. For example, suppose that 10 records were used to create the first file, and it was named FILE1 (10 records). Then, the second, merged file might be named FILE2. FILE2 would contain 20 sorted records. When the next 10 records were ready to be added, FILE2 could be renamed as FILE1 (wiping out FILE1) and a new FILE2 created. Note that FILE2 will always be one generation later than FILE1 and contain the latest addition of records. (Generation data sets can consume a great deal of disk space.)

The advantage of this "limited addition" system for maintaining large, sorted files is that it is easy to implement. The program segments to do the sorting can even be written in BASIC with little sacrifice in sort time. The disadvantages of this system are that one has to enter a small number of records and wait, and there is ever a need to sort the entire file, this technique won't help much.

THE KEY ONLY SORT

When an entire file has to be sorted, the key only sort could prove to be useful (if enough memory is available). Suppose that one has a file containing 800 records, each record with a length of 200 bytes. To do an in-core sort of all records would require 160K. However, suppose that the file contained names, addresses, phone numbers, etc. Suppose further that what was desired was to sort the file on the name field. Using Model III terminology, consider the following segment:

```
10 CLEAR 20000
15 DEFINT I-N
20 DIM NA$(800)
30 OPEN "R",1,"FILEIN",200
40 FIELD 1,20 AS N$, 180 AS R$
    : REM N$ = name - 1st 20 bytes
50 L1 = LOF(1)
60 FOR J = 1 TO L1
70 GET 1,J
80 NA$(J) = N$
    : REM Each name is put in the array NA$
90 NEXT J
100 CMD"O",L1,NA$(1)
    : REM The array NA$ is sorted
```

To produce a list of names, one could print out NA$; however, the ultimate goal of this study is to produce a sorted file. To write a new, sorted file requires that (a) a second file be defined and (b) that some mechanism be defined to tell where to get the records after NA$ is sorted.

The following lines are added to the program:

```
52 FOR J = 1 TO L1
54 NA$(J) = STRING$ (22," ")
    : REM 22 = length of N$ + 2 bytes
56 NEXT J
```

These lines predefine the NA$ array so that MID$ may be used as a pseudovariable (on the left hand side of the =) to minimize garbage collection (garbage collection is a slow process). It is necessary to define only L1 array entries (see statement 50).

Next, some lines must be modified and added:

```
80 MID$ (NA$(J),1,20) = N$
82 MID$ (NA$(J),21,2) = MKI$(J)
```

These two lines prevent garbage collection because they operate on a predefined string. Statement 82 stores the binary equivalent of the record number in bytes 21 and 22 of the NA$ array.

Now, when CMD "O" is executed, the position of the names will change to sorted order, but the 21-22 bytes of the array contain a pointer to record the position that the name came from. How is the sorted file produced? After statement 100, the following would generate the sorted, new file:

```
110 OPEN "R",2,"FILEOUT",200
120 FIELD 2,20 AS NX$,180 AS RX$
130 FOR J = 1 TO L1
140 NR = CVI (MID$(NA$(J),21,2))
150 GET 1,NR
160 LSET NX$ = N$
    : LSET RX$ = R$
170 PUT 2,J
180 NEXT J
190 CLOSE1
    : CLOSE2
```

The key to the finding of the original record is statement 140. Array NA$ was sorted in statement 100. NA$(1) contains the lowest name, and byte 21-22 tells where the record with that lowest name is in FILEIN. This record is fetched (statement 150), the buffer for FILEOUT is loaded (160), and record 1 of FILEOUT is written (170).

The advantage to this system is that it allows the user to sort larger files than in system 1. There is no need to presort the file and all 800 records could be sorted in two passes (one sequential and one random). The disadvantage, of course, lies with the limitation on memory. What if there are 2000 records or what if the key is 60 bytes long or both? This situation must be handled using a partitioned file sort as described in the next section.

THE PARTITIONED SORT

Although the file sort scheme in the previous section is efficient for files of limited size, larger files require more work. The technique will be to divide the file into small digestible parts, sort the parts and then to merge the sorted parts into one sorted file. A scratch file will be used for the intermediate, sorted, partitioned file. An example of this technique might be illustrated as follows. There exists a file of 8000 records, each record 250 bytes long. Suppose that this file again contained names, addresses, etc., and that is was desired to sort it by name. The file could be broken up (partitioned) as 10 files of 800 records each. Each 800 record segment could be sorted and the final file could be created by merging the lowest record from each partition.

Programs similar to the registration and records sorting system of Jefferson Davis Junior College are shown below. The sort of the file is accomplished by using two programs. In the testing phase of this work, it was easier to use two programs for debugging; these two programs could easily be combined.

Program 1, SSS1, generates SCRATCH, the partition-sorted file. Statement 30 limits the partition size to 450 records. Statement 100 uses NN as an end-of-file marker. Statement 100 could be replaced with some other means of finding EOF, but LOF is convenient. The statement in 120 (NH = 400) defines the partition size. The array A$ is predefined in order to use MID$ as a pseudovariable and limit garbage collection. Variable JJ points to the next record to be read in the input file. Within each partition, as it is being created, K defines the position in A$. Note that the sort system is Model II, using the July 80 technique for machine-language sorting and that it is re-loaded every time in statement 320.

Partition Sort Program 1—SSS1

```
2 REM FIRST PART OF SORT "SSS1" APR 2Ø 83
1Ø CLEAR 2ØØØØ
2Ø GOSUB 3ØØØ
     : REM load sort routine
3Ø DIM A$(45Ø)
4Ø DEFINT I - N
5Ø LA% = 18
6Ø OPEN "R",1,"FILEIN",LA%
7Ø FIELD 1,LA% AS R$
8Ø OPEN "R",2,"SCRATCH",LA%
     : REM the output file
9Ø FIELD 2,LA% AS RA$
1ØØ NN = LOF(1)
11Ø K = Ø
12Ø NH = 4ØØ
     : REM this is the number of records in a
     partition
13Ø FOR J = 1 TO NH
     : A$(J) = STRING$(LA% + 2," ")
     : NEXT
14Ø JJ = Ø
15Ø JJ = JJ + 1
16Ø GET 1,JJ
17Ø K = K + 1
18Ø MID$(A$(K),1,LA%) = R$
19Ø MID$(A$(K),LA% + 1,2) = MKI$(JJ)
2ØØ IF K = NH OR JJ = NN THEN 22Ø
21Ø IF JJ < NN THEN 15Ø
22Ø IF K <> NH THEN NH = K
23Ø GOSUB 2ØØØ
     : REM perform sort routine
24Ø REM what follows creates the next partition in
     order
25Ø FOR J = 1 TO NH
26Ø NP = CVI(MID$(A$(J),LA% + 1,2))
27Ø GET 1,NP
28Ø LSET RA$ = LEFT$(A$(J),LA%)
29Ø NQ = JJ - (NH - J)
3ØØ PUT 2,NQ
31Ø NEXT
32Ø GOSUB 3ØØØ
     : REM reload sort (it self-destructs)
33Ø IF JJ < NN THEN K = Ø
     : GOTO 15Ø
34Ø CLOSE 1
35Ø CLOSE 2
36Ø RUN"SSS3"
2ØØØ IF NH < 2 THEN RETURN
2Ø1Ø Z = Ø
2Ø2Ø N1(Ø) = NH
     : N1(1) = VARPTR(A$(1))
2Ø3Ø Z = USRØ(VARPTR(N1(Ø))
2Ø4Ø RETURN
3ØØØ REM FROM JULY 8Ø, TRS 8Ø MICROCOMPUTER NEWS
3Ø1Ø DEFUSRØ = &HFØDØ
3Ø2Ø SYSTEM"LOAD SORT/CIM"
3Ø3Ø REM sort stored on disk and reloaded each time
3Ø4Ø RETURN
```

Program 2, SSS3, reads the SCRATCH file created in SSS1. The trick here is keep track of where you are in each partition. NG defines the number of partitions (line 120). The array NS keeps track of where you are in each partition, and array NP gives the upper limit of each partition as a record number. For example, suppose that there were 1000 records to be merged and partition sizes were 400. There would have to be 3 partitions, NG = 3.

Following the logic in lines 110-200,

NS(1) = 1, NP(0) = 0
NS(2) = 401, NP(1) = 400
NS(3) = 801, NP(2) = 800
NS(4) = 1000, NP(3) = 1000

A$(i) contains the record in the first, NS(i), position of each partition (unless the partition is exhausted, where A$ is set = "ZZZ"). When the A$ array is defined, it is sorted and the smallest A$ is used to file the next record in FILEOUT. As the smallest entity of the sorted A$ array is filed, the next member of the partition from which A$ came replaces A$(1). Note that when A$ is defined (line 240), the partition number is appended. After sorting, the partition number is still in bytes LA% + 1 for 2 bytes. The partition number is the basis for incrementing the counter NS in line 330. To keep the program simple, garbage collection was ignored; however, for more efficiency, the use of predefined strings as in SSS1 would be better. Also, a common BASIC sort for the A$ array was used. For only 10 or so partitions, the loss in sort efficiency due to BASIC is not too bad. Also, if using a non-Model III, one would have to re-load the sort L1 times. If sort speed was a factor, then the machine language sort could be substituted (but it would have to be loaded with POKEs).

Partition Sort Program 2—SSS3

```
2 REM PROGRAM "SSS3" MAR 24 83
1Ø CLEAR 2ØØØØ
2Ø DIM A$(2Ø)
   : REM up to 2Ø partitions
3Ø DEFINT I - N
4Ø LA% = 18
   : REM matches statement 5Ø in SSS1
5Ø OPEN"R",1,"SCRATCH",LA%
   : REM now to be used as input
6Ø FIELD 1,LA% AS R$
7Ø OPEN"R",2,"FILEOUT",LA%
8Ø FIELD 2,LA% AS RA$
9Ø NH = 4ØØ
1ØØ NN = LOF(1)
11Ø K = Ø
12Ø NG = (NN/NH) + 1
   : REM number of partitions
13Ø IF NH * (NG - 1) = NN THEN NG = NG + 1
14Ø FOR J = 1 TO NN STEP NH
15Ø K = K + 1
16Ø NS(K) = J
17Ø NP(K - 1) = J - 1
18Ø NEXT
19Ø NS(K + 1) = NN
2ØØ NP(K) = NN
21Ø NO = Ø
22Ø FOR J = 1 TO NG
23Ø GET 1,NS(J)
24Ø A$(J) = R$ + MKI$(J)
25Ø NEXT
26Ø GOSUB 2ØØØ
   : REM sort A$ (for simplicity BASIC is used)
27Ø IF A$(1) = "ZZZ" THEN 37Ø
28Ø NO = NO + 1
29Ø LSET RA$ = A$(1)
3ØØ PUT 2,NO
31Ø IF NO = NN THEN 37Ø
32Ø NQ = CVI(MID$(A$(1),LA% + 1,2))
33Ø NS(NQ) = NS(NQ) + 1
34Ø IF NS(NQ) > NP(NQ) THEN A$(1) = "ZZZ"
35Ø IF A$(1) <> "ZZZ" THEN GET 1,NS(NQ)
   : A$(1) = R$ + MKI$(NQ)
36Ø GOTO 26Ø
37Ø CLOSE1
   : CLOSE2
38Ø RUN"MENU"
2ØØØ REM SORT FOR PARTITION SAMPLES
2Ø1Ø FOR L = 1 TO NG - 1
2Ø2Ø X$ = A$(L)
2Ø3Ø KA = L
2Ø4Ø FOR K = L + 1 TO NG
2Ø5Ø IF A$(K) < X$ THEN X$ = A$(K)
   : KA = K
2Ø6Ø NEXT K
2Ø7Ø IF KA = L THEN 211Ø
2Ø8Ø XA$ = A$(L)
2Ø9Ø A$(L) = A$(KA)
21ØØ A$(KA) = XA$
211Ø NEXT L
212Ø RETURN
```

CONCLUSION

Techniques exist for sorting arrays. These array sorts are of varying efficiency, often depending on the data. The problem exists that there is no convenient or fixed way to sort large files on micro-personal computers. For files with limited volatility, a limited addition-merge sort might be practical. For a small file where keys will fit in memory, a key sort is fast and efficient. For a sort problem involving an entire large file, the partitioned approach to file sorting was presented. By partitioning a file into smaller files of manageable size, large files can be partition-sorted and re-merged into the sorted, desired product.

ENDNOTES

1. Barron, T., Diehr, G., "Sorting Algorithms for Microcomputers," *BYTE*, Vol. 8, No. 6, May 1983, 482
2. Lorin, H., *Sorting and Sort Systems*, Addison Wesley, Reading, PA, 1975
3. These sort problems were encountered and solved when developing the registration and records system of Jefferson Davis Junior College, Brewton, AL, using a TRS-80, Model II, and a hard disk (5 Meg).
4. There are two peculiarities that should be noted when using the July 80 sort. First, if more than one sort is required in a program (as might be necessary for sorting disk files), the July 80 sort program must be reloaded. This is not really an inconvenience, but it was surprising to find that the sort self-destructed on the Model II. Second, the sort program has what looks like useless statements (like Z = 0 which is later redefined) that are absolutely necessary.

Computer Clubs

80 SOONERS
University of Oklahoma
c/o Brad Quinn
660 Parrington Oval
Norman, OK 73019

MONTREAL TRS-80 USERS GROUP
1176 Phillips Place, Suite 201
Montreal, Quebec
Canada H3B 3C8

COLOR SPECTRUM COMPUTE
c/o Charles Ankenbaum
132 19th Street S.E.
Largo, FL 33540

MID-CITIES TRS-80 USERS GROUP
Mailing address:
c/o D. D. Freeman
334 Fieldside Drive
Garland, TX 75043
Meetings held at:
2901 Abrams
Arlington, TX

MICRO-PRO
c/o John Terfehr
56320 Bonanza Drive
Yucca Valley, CA 92284
1-(619)-365-4349 BBS
1-(619)-365-5757

DENVER AREA TRS-80 ASSOCIATION (DATA)
c/o Ann Crawford
2318 Albion Street
Denver, CO 80207

Extended Play with Orchestra-90

By Bryan Eggers
Software Affair

Anyone who has owned an Orchestra-90™ music synthesizer for more than a few days will probably have written or collected several music files. Did I say "several"? Maybe I should have said "Several HUNDRED," especially if you have access to our Orchestra-90 SIG on CompuServe!

Ever wonder if there is a way to load, compile and play more than a few files with one command? Perhaps you'd like ORCH-90 to play a two-hour concert! Tape users of ORCH-90 can do this with an existing command, "GET @", which plays every file on one side of a tape cassette. We couldn't think of any way to improve on that!

But, the disk version of Orchestra-90 handles multiple files a bit differently. The GET command allows several music files to be entered and played in "jukebox" fashion. This command automatically loads, compiles and plays a small number of music files. Here's an example:

GET TUNE1 TUNE2 TUNE3 TUNE4 TUNE5 TUNE6

You can fill up one entire line with filenames, but suppose you want to play more files than you can fit on one line? One way would be to rename the files with shorter filenames, possibly even with single letter filenames, like A, B and C, but a week later you'd never remember which file is which.

A better way to play a large number of ORCH-90 files on a disk system would be to create an "Automatic Command Input File," more commonly referred to as a "job file," "chain file," or "BUILD file." This file would contain the names of all the music files on your disk(s). Theoretically, you would use the "DO" command from TRSDOS READY to execute the file, which in turn would run ORCH-90 and feed it your long list of filenames.

However, if you've already tried this, you know it doesn't work! ORCH-90 completely ignores your job file. No commands are ever passed to ORCH-90's command line. That's because ORCH-90 takes over the normal TRS-80 keyboard routine and won't allow any inputs from a disk job file. Rather than spend time explaining the technical problems, let's move directly to the solution—a simple patch that takes only a few minutes!

This patch will allow ORCH-90 to execute job files when run on TRSDOS 1.3, allowing you to play as many files as your system will accommodate on line simultaneously! It's really a very small patch, but you should NOT patch your master copy of ORCH90/CMD! This patch affects some of the Edit commands, so we'll apply the patches to a backup copy and use it only for playing job files.

To avoid confusing the two versions, create a backup copy with a different name. This backup will be made from the master ORCH-90 program, VERSION 01.00.00, not a version already configured for a specific number of voices. The new file will be called PLAY90/CMD. At TRSDOS READY, type:

```
COPY ORCH9Ø/CMD to PLAY9Ø/CMD <ENTER>
```

Then, carefully apply the following patches:

```
PATCH PLAY9Ø/CMD (ADD=62EA,FIND=D75C,CHG=ØØØØ)
PATCH PLAY9Ø/CMD (ADD=62F7,FIND=ØE,CHG=16)
PATCH PLAY9Ø/CMD (ADD=6317,FIND=CC243Ø,CHG=CD2BØØ)
PATCH PLAY9Ø/CMD (ADD=6AD8,FIND=C8,CHG=C9)
```

The result is PLAY90/CMD, a version of ORCH-90 patched to accept input from job files. You can now create a job file that contains any number of music filenames in any order, and you can even use the same job file to Quit ORCH-90 and reconfigure it for a different number of voices before it plays the next song!

PLAY90/CMD should only be used for playing job files, not for editing music files. This change in ORCH-90's keyboard routine affects certain characteristics of the Edit function. Continue using your regular ORCH90/CMD program for editing purposes.

Each parameter actually passed to PLAY90 must be preceded by a SPACE (or any dummy character). Let's examine a short job file. This syntax is for ORCH-90 used on TRS-80 Model 4 computer (in Model III mode on TRSDOS 1.3).

```
PLAY90/CMD
 4
 N
 GET TUNE1
 GET TUNE2
 GET TUNE3
 GET TUNE4
 GET TUNE5
 GET TUNE6
 Q
```

I saved this file to disk as MEDLEY/BLD. Create one just like it using SCRIPSIT, the BUILD command in TRSDOS, or any ASCII editor. Use your own music filenames instead of "TUNE1", "TUNE2", etc. Execute this job file from TRSDOS READY by typing:

```
DO MEDLEY <ENTER>
```

The first command in the job file is "PLAY90/CMD". This command invokes the actual music synthesizer and is sent directly to TRSDOS, so no leading space is required.

Next, an initialization prompt displayed by PLAY90 asks for the NUMBER OF VOICES that ORCH-90 is to be configured for. The "4" in our job file supplies the answer to that question. This could be 3, 4 or 5 voices. The last prompt asks if we want to save the program, and we answer "N" to that one. These commands are accepted and executed so fast that you'll miss them if you blink!

Next comes the command "GET TUNE1". This command appears on PLAY90's command line and will be executed just as if you had typed it in yourself. PLAY90 will load, compile and play the music file TUNE1/ORC, then request the next command in our job file, "GET TUNE2"

After TUNE6 (the last song in our file) has been played, our job file sends a "Q" to PLAY90. This tells ORCH-90 to Quit and return to TRSDOS READY.

If PLAY90 can't find a music file on your system, or if a music file won't compile properly, the job won't "bomb out." PLAY90 will execute the next job file command. This means that your music will never stop until the last file is played, but it also means that a song will be skipped if you spelled the filename wrong or if it won't compile properly. Check your job file carefully. With a few dozen files on line, you might not notice that one is being skipped each time!

If you press the O key while a song is playing, the next command in the job file will be executed immediately.

Suppose that you have some 4 voice arrangements and some 5 voice arrangements. As you know, ORCH-90 always sounds best when configured for the correct number of voices. Here's a job file that will load PLAY90, configure it for 4 voices, play several 4 voice songs, then go back to TRSDOS READY, load PLAY90, configure for 5 voices, and play several 5 voice arrangements!

```
PLAY90/CMD
  4
  N
  GET TUNE1
  GET TUNE2
  GET TUNE3
  GET TUNE4
  GET TUNE5
  GET TUNE6
  Q
PLAY90/CMD
  5
  N
  GET TUNE7
  GET TUNE8
  GET TUNE9
  GET TUNE10
  GET TUNE11
  GET TUNE12
  Q
```

NOTE TO MODEL III USERS

Your job file needs to answer the "FAST CLOCK" prompt before the "VOICES 3/4/5" prompt appears. Your file should contain an "N" immediately after the "PLAY90/CMD" command like this:

```
FAST CLOCK
PLAY90/CMD
  N
  4
  N
  GET TUNE1
  GET TUNE2
  GET TUNE3
  GET TUNE4
  GET TUNE5
  GET TUNE6
  Q
```

ORCHUTIL/CMD

One of these days someone will ask you to transfer all the files on your disk to their cassette. When that happens, you will thank me for this next patch! A disk-to-cassette transfer like this requires the ORCHUTIL/CMD program that you received with your ORCH-90, plus a considerable amount of time and patience, especially if two or three dozen files are involved.

To keep your friendship intact, we'll make ORCHUTIL accept job file commands, too. You'll be able to insert any number of filenames into a job file then and sit back while ORCHUTIL does all the disk-to-cassette file transfers for you.

To avoid patching the original copy of ORCHUTIL/CMD, we'll create a backup copy and name it UTIL/CMD. At TRSDOS READY type:

```
COPY ORCHUTIL/CMD to UTIL/CMD//(ENTER//)
```

Then, carefully apply the following patches:

```
PATCH UTIL/CMD (ADD=5A9D,FIND=9554,CHG=ØØØØ)
PATCH UTIL/CMD (ADD=5AA8,FIND=Ø6,CHG=ØC)
PATCH UTIL/CMD (ADD=5AC1,FIND=243Ø,CHG=2BØØ)
```

We now have a new copy of ORCHUTIL called UTIL/CMD that will accept job commands from a disk file. The job file syntax will be a bit different because UTIL/CMD accepts single keystroke commands using the INKEY$ function. In other words, ENTER is not required after most commands. Also, no leading spaces are required. I'll give you a sample job file and you can create your own by inserting the actual music filenames. This is the format:

```
UTIL/CMD
OH
RTUNE1
WTUNE1
RTUNE2
WTUNE2
RTUNE3
WTUNE3
Q
```

Let's call this one TRANSFER/BLD. Make sure your disks are inserted and that your cassette player has a blank cassette inserted and is in RECORD mode. Then, execute your file from TRSDOS READY by typing:

```
DO TRANSFER <ENTER>
```

The first line in the job file runs our UTIL/CMD program. The next line contains the characters "OH". Two commands are actually being sent in this one line because of the INKEY$ function. The "O" tells UTIL/CMD that you want to change the Output option of program. The "H" selects High Speed (1500 baud tape cassette). UTIL/CMD is now configured to Read music files from disk and Write them to cassette at 1500 baud.

Our next step is to Read a music file from disk. We use the command "RTUNE1" in our job file. Again, two commands are passed in a single line. The "R" tells UTIL/CMD that you want to Read a file into memory, and "TUNE1" supplies the filename.

The next line in our job file is "WTUNE1" which contains a "W" to Write the file to cassette, followed by the tape filename "TUNE1". A word of caution here—don't forget that disk files can contain up to 8 characters while tape filenames can contain only 6 characters. You must follow the rules for standard tape filenames.

The job file then Reads and Writes the TUNE2 and TUNE3 music files, and ends the session by sending a "Q". This tells PLAY90 to (Q)uit and return to TRSDOS READY.

Orchestra-90 is a trademark of Software Affair, Ltd.

Line Printer V Print Codes

Mary Orum
Lago Vista Business Services, Inc.
13 Poe Cove, Unit #1-E
Lago Vista, TX 78641

I am not a proficient programmer. My company uses numerous packaged Radio Shack software programs, but other than copying some simple fun programs from your magazine, my expertise in programming is quite limited.

Last spring we bought a DMP-500 printer. We already had two Daisy Wheel printers, and we were accustomed to changing the pitch of the printing by flipping a switch or changing the style by changing the print wheel. However, changing the print on the DMP-500 was not quite so simple. It was my understanding that while the DMP-500 was on, the style and pitch of the printing had to be changed via software print codes. And since none of the programs we were using provided for sending print codes to the printer from software, my employees and I were forever going back to the manual to look up the print codes for the style we wished to use. PRNTCODE is a simple program I wrote which allows multiple choice selection of print code options available on the DMP-500. It first changes the CRT print to large size. It then reminds the operator to be sure that the printer is ready to be programmed. Next it asks the operator to select one of five print styles and allows her to print a sample of the style she has chosen. It then asks if the operator would like to have the printing elongated or bold printed and again allows for a sample. Finally, when "Q" is pressed, it returns the print to normal, rolls the paper up to the beginning of the next page, and returns to TRSDOS. We have found this little program to be quite a time saver for switching from 12 to 10 to 16 characters per inch in the course of changing from one program to another. I think others might find it to be of some benefit.

```
10 REM *** ================================= ***
20 REM             *** "PRNTCODE" ***
30 REM *** A PROGRAM WRITTEN BY MARY ORUM FOR ***
40 REM *** LAGO VISTA BUSINESS SERVICES, INC. ***
50 REM *** FOR SELECTING PRINT OPTIONS ON THE ***
60 REM ***     RADIO SHACK DMP 500 PRINTER    ***
70 REM *** ================================= ***
80 CLS
     : PRINT CHR$(31)
90 PRINT "BE SURE THE PRINTER IS LOADED WITH PAPER"
100 PRINT "AND THAT THE LINE SWITCH IS 'ON'."
110 PRINT
     : PRINT
     : PRINT
     : PRINT
     : PRINT
120 PRINT "BE SURE TO PRESS THE RESTART BUTTON."
130 PRINT
     : PRINT
     : PRINT
     : PRINT
     : PRINT
140 PRINT "PRESS 'Y' WHEN READY."
150 PRINT
     : PRINT
     : PRINT
     : PRINT
     : PRINT
160 INPUT "NOW"; X$
170 IF X$ = "Y" THEN 180
     : ELSE 160
180 LPRINT CHR$(27) CHR$(15)
     : LPRINT CHR$(27) CHR$(32)
190 CLS
200 PRINT "CHOOSE ONE OF THE FOLLOWING OPTIONS:"
     : PRINT
     : PRINT
     : PRINT
     : PRINT
210 PRINT "A.  NORMAL -- 10 CHARACTERS PER INCH
     "PRINT
220 PRINT "B.  COMPRESSED -- 12 CHARACTERS PER INCH"
     : PRINT
230 PRINT "C.  CONDENSED -- 16 CHARACTERS PER INCH "
     : PRINT
240 PRINT "D.  CORRESPONDENCE -- LETTER QUALITY    "
     : PRINT
250 PRINT "E.  PROPORTIONAL SPACING     "
     : PRINT
     : PRINT
     : PRINT
     : PRINT
260 INPUT "WHICH OPTION"; N$
270 CLS
280 PRINT "DO YOU WANT A SAMPLE OF THE"
     : PRINT
     : PRINT
290 INPUT "PRINT YOU HAVE CHOSEN -- Y OR N"; A$
300 IF N$ = "A" THEN 350
310 IF N$ = "B" THEN 390
320 IF N$ = "C" THEN 430
330 IF N$ = "D" THEN 470
340 IF N$ = "E" THEN 510
350 LPRINT CHR$(27) CHR$(19)
360 IF A$ = "Y", THEN 370 ELSE 550
370 LPRINT "This is an example of normal or default
     printing."
380 LPRINT "This printing has ten (10) characters per
     inch."
     : LPRINT
     : LPRINT
     : LPRINT
     : GOTO 550
390 LPRINT CHR$(27) CHR$(23)
400 IF A$ = "Y" THEN 410 ELSE 550
410 LPRINT "This is an example of compressed
     printing."
420 LPRINT "This printing has twelve (12) characters
     per inch."
     : LPRINT
     : LPRINT
     : LPRINT
     : GOTO 550
430 LPRINT CHR$(27) CHR$(20)
440 IF A$ = "Y" THEN 450
     : ELSE 550
450 LPRINT "This is an example of condensed
     printing."
```

```
460 LPRINT "This printing has sixteen (16) characters
     per inch."
     : LPRINT
     : LPRINT
     : LPRINT
     : GOTO 550
470 LPRINT CHR$(27) CHR$(18)
480 IF A$ = "Y" THEN 490 ELSE 550
490 LPRINT "This is an example of correspondence
     quality printing."
500 LPRINT "This printing has ten (10) characters per
     inch."
     : LPRINT
     : LPRINT
     : LPRINT
     : GOTO 550
510 LPRINT CHR$(27) CHR$(17)
520 IF A$ = "Y" THEN 530 ELSE 550
530 LPRINT "This is an example of proportional
     printing."
540 LPRINT "This printing has different spacing
     allotted for each letter."
     : LPRINT
     : LPRINT
     : LPRINT
     : GOTO 550
550 CLS
560 PRINT "WHAT DO YOU WANT TO DO WITH THE"
     : PRINT
570 PRINT "PRINTING?"
     : PRINT
     : PRINT
     : PRINT
     : PRINT
580 PRINT "A.  ELONGATION"
     : PRINT
590 PRINT "B.  BOLD PRINTING"
     : PRINT
600 PRINT "C.  NOTHING"
     : PRINT
     : PRINT
     : PRINT
     : PRINT
610 INPUT "WHICH OPTION"; B$
620 CLS
630 IF B$ = "C" THEN 730
640 INPUT "DO YOU WANT A SAMPLE -- Y OR N"; C$
650 IF B$ = "A" THEN 670
660 IF B$ = "B" THEN 700
670 LPRINT CHR$(27) CHR$(14)
680 IF C$ = "Y" THEN 690 ELSE 730
690 LPRINT "This is an example of elongated
     printing."
     : LPRINT
     : LPRINT
     : LPRINT
     : GOTO 730
700 LPRINT CHR$(27) CHR$(31)
710 IF C$ = "Y" THEN 720 ELSE 730
720 LPRINT "This is an example of bold printing."
     : LPRINT
     : LPRINT
     : LPRINT
     : GOTO 730
730 CLS
740 PRINT "PRESS 'Q' TO RETURN TO TRSDOS."
     : PRINT
     : PRINT
     : PRINT
     : PRINT
750 INPUT "ALL DONE"; Y$
760 IF Y$ = "Q" THEN PRINT CHR$(30)
     : SYSTEM "FORMS T"
     : SYSTEM
     : ELSE 750
```

Is There a Name in Your Phone Number?

by Randy Rife

This program for the Model 12 takes a seven-digit phone number and converts it to its alphabetic equivalents.

This all started as an easy way of remembering phone extensions in our office. (We have BUDI, BETI, CUDL, and even a BRAT on our floor.) Soon I was asked to make the program work on complete phone numbers. Converting the program from four to seven digits was easy, but the possible combinations went from 81 to 2,187! Because of this, I added the ability to print out the results on a printer.

Running the program is easy. Start by entering the phone number (use no spaces or dashes) and press S (for screen) or P (for Printer). If you select Printer, press Y (for yes) or N (for no) at "Pause between pages?" When the program is finished listing all of the combinations, it will return to where you enter the phone number. Press BREAK to exit or enter a new phone number to rerun the program.

There is a problem with ones and zeroes since there are no letters on the telephone dial that correspond to them. Therefore, this program will display (or print) a one or a zero instead of a letter.

Although written on a Model 12, this program should work without modifications on a Model II or Model 16. Changing SYSTEM"T" in lines 620 and 650 to LPRINT CHR$(12) should be the only changes needed for Models I/III/4. Change the **LPRINT**s to **PRINT#-2**s for the Color Computer.

```
100 REM ***  CONVERT PHONE NUMBERS TO TEXT  ***
110 REM *** WRITTEN SEPT 1983 BY RANDY RIFE ***
120 DATA 0, 1, ABC, DEF, GHI, JKL, MNO, PRS, TUV, WXY
130 FOR X=0 TO 9
     : READ N$(X)
     : NEXT X
     : CLS
140 INPUT "Enter your phone number (7 digits, no
     dash) "; PH$
150 IF LEN(PH$) <> 7 THEN 140
160 P1=VAL(MID$(PH$,1,1))
     : P2=VAL(MID$(PH$,2,1))
     : P3=VAL(MID$(PH$,3,1))
170 P4=VAL(MID$(PH$,4,1))
     : P5=VAL(MID$(PH$,5,1))
     : P6=VAL(MID$(PH$,6,1))
180 P7=VAL(MID$(PH$,7,1))
190 PRINT "Print on the Screen or Printer (S/P)? ";
200 N$=INKEY$
     : IF N$="" THEN 200
210 IF N$="P" OR N$="p" THEN PRINT "PRINTER"
     : GOTO 510
220 IF N$="S" OR N$="s" THEN PRINT "SCREEN"
     : GOTO 310
230 GOTO 200
300 REM *** PRINT TO THE SCREEN ***
310 FOR X1=1 TO LEN(N$(P1))
     : FOR X2=1 TO LEN(N$(P2))
     : FOR X3=1 TO LEN(N$(P3))
320 FOR X4=1 TO LEN(N$(P4))
     : FOR X5=1 TO LEN(N$(P5))
     : FOR X6=1 TO LEN(N$(P6))
330 FOR X7=1 TO LEN(N$(P7))
```

```
340 PRINT MID$(N$(P1),X1,1); MID$(N$(P2),X2,1);
     MID$(N$(P3),X3,1);
350 PRINT MID$(N$(P4),X4,1); MID$(N$(P5),X5,1);
     MID$(N$(P6),X6,1);
360 PRINT MID$(N$(P7),X7,1),
370 NEXT X7,X6,X5,X4,X3,X2,X1
380 PRINT
     : PRINT
     : GOTO 140
500 REM *** PRINT ON THE PRINTER ***
510 PRINT "Pause between pages (Y/N)? ";
520 PS$=INKEY$
     : IF PS$="" THEN 520
530 IF PS$="Y" OR PS$="y" THEN PRINT "YES"
540 IF PS$="N" OR PS$="n" THEN PRINT "NO"
550 PG=0
     : L=0
     : GOSUB 720
560 FOR X1=1 TO LEN(N$(P1))
     : FOR X2=1 TO LEN(N$(P2))
     : FOR X3=1 TO LEN(N$(P3))
     : FOR X4=1 TO LEN(N$(P4))
570 FOR X5=1 TO LEN(N$(P5))
     : FOR X6=1 TO LEN(N$(P6))
     : FOR X7=1 TO LEN(N$(P7))
580 LPRINT MID$(N$(P1),X1,1); MID$(N$(P2),X2,1);
     MID$(N$(P3),X3,1);
590 LPRINT MID$(N$(P4),X4,1); MID$(N$(P5),X5,1);
     MID$(N$(P6),X6,1);
600 LPRINT MID$(N$(P7),X7,1),
610 C=C+1
     : IF C=5 THEN LPRINT
     : C=0
     : L=L+1
620 IF L=50 THEN L=0
     : SYSTEM"T"
     : GOSUB 710
630 NEXT X7, X6, X5, X4, X3, X2, X1
640 LPRINT
     : LPRINT
     : LPRINT "END OF LISTING"
650 SYSTEM"T"
     : PRINT
     : GOTO 140
700 REM *** PRINT HEADER ***
710 IF PS$="Y" OR PS$="y" THEN GOSUB 810
720 PG=PG+1
730 LPRINT "PHONE NUMBER = "LEFT$(PH$,3)" -
     "RIGHT$(PH$,4);TAB(70)"PAGE"PG
740 LPRINT
     : RETURN
800 REM *** PAUSE BETWEEN PAGES ***
810 PRINT "Press <SPACEBAR> for next page or <Q> to
     quit"
820 N$=INKEY$
     : IF N$="" THEN 820
830 IF N$=" " THEN RETURN
840 IF N$="Q" OR N$="q" THEN RUN
850 GOTO 820
```

PATCH Files for the Model II

Alexander B. Spencer
7708 Turnberry Lane
Dallas, TX 75248

I use this short program for maintaining my file of patches. The patches are built into DO files and executed by the DO command. Each file begins with a CLS command to clear the screen followed by a PAUSE statement that identifies the patch. All patches that I might wish to reverse from time to time are appended with the reverse patch preceded by CLS and PAUSE statements to clear the screen and identify them. To use, the DO file is executed down through the desired patch and execution terminated by (BREAK).

All "fix" type patches for a given program are combined into a single file for convenience in transferring them from disk to disk and to conserve directory space. I put a copy of each patch applied on the disk and thereby have a record of patching to programs on that disk. (These files are short and use little space.)

I have reserved a disk solely for my patch files and the patch file maintenance program. This disk is, of course, appropriately backed up as good practice demands.

I have included in the program menu a command that I have found very convenient when using the Daisy Wheel II printer with tractor feed. It is a form feed command that quickly allows feeding printed pages past the tractor mechanism of the DW II for easy removal. I now include it in the menu of most of my programs including the user menus of Profile II and Profile+. It can easily be added to menus of BASIC programs such as General Ledger.

```
10 REM              ********** PROGRAM 'PFILE'
     **********
20 REM
30 REM   * BY ALEXANDER B SPENCER
40 '
50 REM   * PATCH FILE DIRECTORY
60 REM   * FILES CREATED BY 'BUILD'
70 REM   * FILES EXECUTED BY 'DO'
80 REM   * FILE NAMES OF FORM 'PATCHnn'
90 REM     WHERE 'nn' IS TWO DIGIT NUMBER
100 REM  * FIRST 'PAUSE' USED AS TITLE
110 REM  * REQUIRES -F:1
120 REM  * VERSION 1.2 (07/15/82)
130 '
140 CLEAR 500
150 GOTO 240
160 '
170 REM ***** MISC SEQUENCES *****
180 LPRINT CHR$(12);
     : RETURN
190 PRINT TAB(30)CHR$(26);A$;CHR$(25);B$
     : PRINT
     : RETURN
200 PRINT@ (23,32),STRING$(30," ");
210 PRINT@ (23,32),"KEY TO CONTINUE ";CHR$(01);
220 Z$=INKEY$
     : IF Z$="" THEN 220 ELSE RETURN
230 '
240 REM ***** MENU *****
250 CLS
     : PRINT CHR$(02)
260 PRINT TAB(28)"-PATCH FILE MAINTENANCE-"
     : PRINT
     : PRINT
270 PRINT TAB(38)"MENU"
     : PRINT
280 A$="1"
     : B$=" DISPLAY DIRECTORY"
     : GOSUB 190
290 A$="2"
     : B$=" PRINT DIRECTORY"
     : GOSUB 190
300 A$="3"
     : B$=" DISPLAY FILES"
     : GOSUB 190
310 A$="4"
     : B$=" PRINT FILES"
     : GOSUB 190
```

```
320 A$="5"
     : B$=" DISPLAY FILE"
     : GOSUB 190
330 A$="6"
     : B$=" PRINT FILE"
     : GOSUB 190
340 A$="7"
     : B$=" FORM FEED"
     : GOSUB 190
350 A$="0"
     : B$=" EXIT"
     : GOSUB 190
360 GOSUB 200
370 Z=VAL(Z$)
380 IF Z$="0" THEN CLOSE
     : CLS
     : END
390 ON Z GOSUB 410,570,750,900,1060,1220,180
     : GOTO 240
400 '
410 REM ***** DISPLAY DIRECTORY *****
420 CLS
     : PRINT CHR$(02);
430 PRINT TAB(29)"-PATCH FILE DIRECTORY-"
440 ON ERROR GOTO 540
450 I=1
460 F$="PATCH" + RIGHT$("00" + RIGHT$(STR$(I),
     LEN(STR$(I))-1),2)
470 PRINT
     : OPEN "I",1, F$
480 LINE INPUT #1,IN$
490 IF MID$(IN$,2,5)="PAUSE" THEN X=7
     : GOTO 520
500 IF MID$(IN$,1,5)="PAUSE" THEN X=6
     : GOTO 520
510 GOTO 480
520 PRINT F$;"   ";RIGHT$(IN$,LEN(IN$)-X);
530 CLOSE
     : I=I+1
     : GOTO 460
540 CLOSE
     : RESUME 550
550 ON ERROR GOTO 0
     : GOSUB 200
     : RETURN
560 '
570 REM ***** PRINT DIRECTORY *****
580 CLS
     : PRINT CHR$(02);
590 PRINT TAB(31)"-PRINTING DIRECTORY-"
     : PRINT TAB(40)CHR$(01);
600 LPRINT TAB(29)"-PATCH FILE DIRECTORY-"
610 ON ERROR GOTO 720
620 I=1
630 F$="PATCH" + RIGHT$("00"+RIGHT$(STR$(I),
     LEN(STR$(I))-1),2)
640 LPRINT
     : OPEN "I",1, F$
650 IF EOF(1) THEN CLOSE
     : RESUME 730
660 LINE INPUT #1,IN$
670 IF MID$(IN$,2,5)="PAUSE" THEN X=7
     : GOTO 700
680 IF MID$(IN$,1,5)="PAUSE" THEN X=6
     : GOTO 700
690 GOTO 660
700 LPRINT F$;"   ";RIGHT$(IN$,LEN(IN$)-X);
710 CLOSE
     : I=I+1
     : GOTO 630
720 CLOSE
     : LPRINT CHR$(12)
     : RESUME 730
730 ON ERROR GOTO 0
     : RETURN
740 '
750 REM ***** DISPLAY PATCH FILES *****
760 CLS
     : PRINT CHR$(02);
770 ON ERROR GOTO 870
780 I=1
790 PRINT TAB(28)"-LISTING OF PATCH FILE-"
     : PRINT
     : PRINT
800 F$="PATCH" + RIGHT$("00"+RIGHT$(STR$(I),
     LEN(STR$(I))-1),2)
810 PRINT
     : PRINT
     : OPEN "I",1, F$
     : PRINT TAB(36)F$
     : PRINT
820 IF EOF(1) THEN CLOSE
     : I=I+1
     : GOSUB 200
     : CLS
     : PRINT CHR$(02)
     : GOTO 790
830 LINE INPUT #1,IN$
840 IF ASC(LEFT$(IN$,1))=208 THEN PRINT
     RIGHT$(IN$,LEN(IN$)-1)
     : J=1
     : GOTO 820
850 PRINT IN$
860 GOTO 820
870 CLOSE
     : RESUME 880
880 RETURN
890 '
900 REM ***** PRINT PATCH FILES *****
910 CLS
     : PRINT CHR$(02);
920 PRINT TAB(30) "-LISTING PATCH FILES-"
     : PRINT TAB(40)CHR$(01);
930 LPRINT TAB(28)"-LISTING OF PATCH FILES-"
940 ON ERROR GOTO 1030
950 I=1
960 F$="PATCH" +
     RIGHT$("00"+RIGHT$(STR$(I),LEN(STR$(I))-1),2)
970 LPRINT
     : LPRINT
     : OPEN "I",1, F$
     : LPRINT TAB(36)F$
     : LPRINT
980 IF EOF(1) THEN CLOSE
     : I=I+1
     : GOTO 960
990 LINE INPUT #1,IN$
1000 IF ASC(LEFT$(IN$,1))=208 THEN LPRINT
     RIGHT$(IN$,LEN(IN$)-1)
     : J=1
     : GOTO 980
1010 LPRINT IN$
1020 GOTO 980
1030 CLOSE
     : RESUME 1040
1040 LPRINT CHR$(12)
     : RETURN
1050 '
1060 REM ***** DISPLAY PATCH FILE *****
1070 CLS
1080 PRINT TAB(5);
     : LINE INPUT "DISPLAY FILE NUMBER ";I$
1090 CLS
     : PRINT CHR$(02)
1100 F$="PATCH"+RIGHT$("00"+I$,2)
1110 PRINT TAB(28)"-LISTING OF ";F$;"-"
     : PRINT
1120 ON ERROR GOTO 1190
1130 PRINT
     : PRINT
     : OPEN "I",1, F$
     : PRINT TAB(36)F$
     : PRINT
114Ø IF EOF(1) THEN 119Ø
115Ø LINE INPUT #1,IN$
116Ø IF ASC(LEFT$(IN$,1))=2Ø8 THEN PRINT
     RIGHT$(IN$,LEN(IN$)-1)
     : GOTO 114Ø
117Ø PRINT IN$
118Ø GOTO 114Ø
119Ø CLOSE
     : RESUME 12ØØ
12ØØ GOSUB 2ØØ
     : RETURN
121Ø '
122Ø REM ***** PRINT PATCH FILE *****
123Ø CLS
124Ø PRINT TAB(7);
     : LINE INPUT "LIST FILE NUMBER ";I$
125Ø CLS
     : PRINT CHR$(Ø2)
126Ø F$="PATCH"+RIGHT$("ØØ"+I$,2)
127Ø PRINT TAB(32)"-LISTING ";F$;"-"
     : PRINT TAB(4Ø)CHR$(Ø1);
128Ø LPRINT TAB(28)"-LISTING OF ";F$;"-"
129Ø ON ERROR GOTO 136Ø
13ØØ LPRINT
     : LPRINT
     : OPEN "I",1, F$
131Ø IF EOF(1) THEN 136Ø
132Ø LINE INPUT #1,IN$
133Ø IF ASC(LEFT$(IN$,1))=2Ø8 THEN LPRINT
     RIGHT$(IN$,LEN(IN$)-1)
     : J=1
     : GOTO 131Ø
134Ø LPRINT IN$
135Ø GOTO 131Ø
136Ø CLOSE
     : RESUME 137Ø
137Ø LPRINT CHR$(12)
     : RETURN
```

Cursive Printing with the Model II and the LP V

Pete Giovagnoli
4200 N.E. Birmingham Rd.
Kansas City, MO 64117

The START routine allows for entry and later change of data in a file called LETTER with a record length of ten. In line 60 the program asks you to input R. R is the consecutive record number entered (R = 1, R = 2, R = 3, . . . ,R = N).

Letters each take up ten records. Since the record length is ten, the letters are actually printed in a 10 × 10 grid. A starts at record 1, B starts at record 11, C starts at record 21, D starts at record 31 and so on. The punctuation characters (! . ? , space) follow the alpha characters. If you need to make a change to a letter, load and run START and at INPUT R do the following:

Find the record number of the character you want to change.
Count to the line of print in that letter starting at 0. Add this to the starting record number to get the record in the letter you want to change. After you put this in as R, you are ready to change the line so it is correct.
The data being entered is stored as L$ in the file.

START

```
1Ø '  ROUTINE TO CREATE, AND INPUT DATA TO, THE 'LETTER' FILE
2Ø '  PETE GIOVAGNOLI 1983
3Ø '  RUN THIS ROUTINE FIRST
4Ø OPEN"D",1,"LETTER",1Ø
5Ø FIELD 1,1Ø AS L$
6Ø INPUT "INPUT R";R
7Ø INPUT X$
8Ø LSET L$=X$
    : PUT 1,R
9Ø GOTO 6Ø
```

CURSIVE accesses the file named LETTER created by START and prints the cursive characters. First the program sets the printer to the graphics printing mode. Next, it asks for a message (M$). After you enter the message, the program checks the length of the message to see if it is too long. If the message is too long, the program asks for another message. Then it finds the length of your message again (T$). It sets up a loop (FOR T1 = 1 TO T) to find each letter of the message. Each letter is stored in N$.

Next, it looks through the list of letters to find the correct one. The starting record (X) for that letter is found and used in a GET 1,X statement later. First though it must add C to X to make sure it has the correct record. This procedure is like the one explained for changing letters in the LETTER file.

Then the program sets up a loop (FOR V = 1 TO 10) to analyze the 1$ in record X. In this loop, O$ = MID$(L$,V,1) will return a number 0 - 5. It prints a graphics character for each number.

0 = space
1 = ■
2 = ◤
3 = ◢
4 = ◥
5 = ◣

After it has completed ten loops, it will do a NEXT T1 to get the next letter. After all letters are completed, it does a CHR$ (10) which is a line feed, adds 1 to C, and checks to see what C is. If C is anything but 10, it will go back to 60 and print the next line of each letter in the message. In this manner, all first lines of letters are printed in a row, then all second lines, etc. until all lines are printed (C will equal 10.)

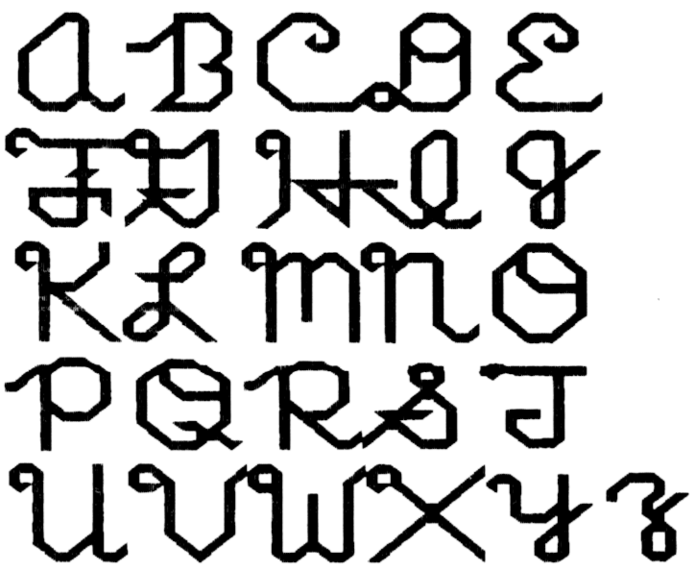
Finally, it will ask for ANOTHER LINE (Y/N)?. If you say N, it will ask for ANOTHER MESSAGE (Y/N)?. If you answered Y to ANOTHER LINE, it would have gone back to the beginning of the program and asked for the next message. If you answer Y to ANOTHER MESSAGE, the program will space the printer to the top of the next page, go back to the beginning, and ask for a new message. If you answer N, it will go back to regular printing, close the file LETTER, and end.

```
1Ø '  CURSIVE PRINTING ROUTINE FOR LP V AND MODEL II
2Ø '  PETE GIOVAGNOLI
3Ø '  RUN ONLY AFTER ALL DATA HAS BEEN ENTERED
4Ø CLS
5Ø OPEN"D",1,"LETTER",1Ø
6Ø FIELD 1,1Ø AS L$
7Ø LPRINT CHR$(27);CHR$(28)
8Ø PRINT@35Ø, "***CURSIVE***"
9Ø PRINT@96Ø, "THIS PROGRAM WILL PRINT OUT ANY
     MESSAGE OF UP TO 13 CHARACTERS (ON ONE LINE) YOU
     PUT IN. MORE LINES MAY BE ADDED LATER."
1ØØ INPUT "ENTER YOUR MESSAGE";M$
     : C=Ø
11Ø IF LEN(M$)=14 OR LEN(M$) > 14 THEN PRINT "ONLY 13
     CHARACTERS, GREEDY!"
     : GOTO 1ØØ
12Ø T=LEN(M$)
```

```
13Ø FOR T1=1 TO T
    : N$=MID$(M$,T1,1)
14Ø IF N$="A" THEN X=1
15Ø IF N$="B" THEN X=11
16Ø IF N$="C" THEN X=21
17Ø IF N$="D" THEN X=31
18Ø IF N$="E" THEN X=41
19Ø IF N$="F" THEN X=51
2ØØ IF N$="G" THEN X=61
21Ø IF N$="H" THEN X=71
22Ø IF N$="I" THEN X=81
23Ø IF N$="J" THEN X=91
24Ø IF N$="K" THEN X=1Ø1
25Ø IF N$="L" THEN X=111
26Ø IF N$="M" THEN X=121
27Ø IF N$="N" THEN X=131
28Ø IF N$="O" THEN X=141
29Ø IF N$="P" THEN X=151
3ØØ IF N$="Q" THEN X=161
31Ø IF N$="R" THEN X=171
32Ø IF N$="S" THEN X=181
33Ø IF N$="T" THEN X=191
34Ø IF N$="U" THEN X=2Ø1
35Ø IF N$="V" THEN X=211
36Ø IF N$="W" THEN X=221
37Ø IF N$="X" THEN X=231
38Ø IF N$="Y" THEN X=241
39Ø IF N$="Z" THEN X=251
4ØØ IF N$="?" THEN X=261
41Ø IF N$="." THEN X=271
42Ø IF N$="!" THEN X=281
43Ø IF N$="," THEN X=291
44Ø IF N$=" " THEN X=3Ø1
45Ø X=X+C
46Ø GET 1,X
47Ø FOR V=1 TO 1Ø
48Ø O$=MID$(L$,V,1)
49Ø IF O$="Ø" THEN LPRINT CHR$(224);
5ØØ IF O$="1" THEN LPRINT CHR$(239);
51Ø IF O$="2" THEN LPRINT CHR$(251);
52Ø IF O$="3" THEN LPRINT CHR$(252);
53Ø IF O$="4" THEN LPRINT CHR$(253);
54Ø IF O$="5" THEN LPRINT CHR$(254);
55Ø NEXT V
56Ø NEXT T1
    : LPRINT CHR$(1Ø)
57Ø C=C+1
58Ø IF C=1Ø THEN 6ØØ
59Ø GOTO 13Ø
6ØØ INPUT "WOULD YOU LIKE ANOTHER LINE"; LI$
61Ø IF LI$="Y" THEN 1ØØ
62Ø CLS
    : PRINT@96Ø,"WOULD YOU LIKE ANOTHER MESSAGE"
    : INPUT NME$
63Ø IF NME$="Y" THEN CLS
    : SYSTEM "FORMS T"
    : GOTO 8Ø
64Ø LPRINT CHR$(1Ø)
    : LPRINT CHR$(27);CHR$(54)
65Ø CLOSE
66Ø END
```

If you don't want to have to create your own character set you can run to program below to get Mr. Giovagnoli's characters. After running the program below then run the CURSIVE program.

```
1Ø T=Ø
2Ø OPEN "D",1,"LETTER",1Ø
3Ø FIELD 1, 1Ø AS L$
4Ø IF T=31Ø THEN END
5Ø READ L1$
6Ø LSET L$=L1$
7Ø T=T+1
8Ø PUT 1,T
9Ø GOTO 4Ø
1ØØ DATA ØØØØ315ØØØ, ØØØ32Ø1ØØØ, ØØ32ØØ1ØØØ,
     Ø32ØØØ1ØØØ, Ø1ØØØØ1ØØØ
11Ø DATA Ø1ØØØØ1ØØØ, Ø1ØØØØ1ØØØ, Ø1ØØØØ1ØØØ,
     Ø45ØØ315Ø3, ØØ41120412
12Ø DATA ØØØØ1115ØØ, ØØØ31ØØ45Ø, ØØ321ØØØ1Ø,
     112Ø1ØØØ1Ø, ØØØØ1ØØ32Ø
13Ø DATA ØØØØ1Ø415Ø, ØØØØ1ØØØ1Ø, ØØØØ1ØØØ1Ø,
     ØØØ32ØØ32Ø, ØØ311112ØØ
14Ø DATA ØØØ3115ØØØ, ØØ32ØØ45ØØ, Ø32ØØ3Ø1ØØ,
     Ø1ØØØ412ØØ, Ø1ØØØØØØØØ
15Ø DATA Ø1ØØØØØØØØ, Ø1ØØØØØØØØ, Ø45ØØØØØØØ,
     ØØ45ØØØØØ3, ØØØ4111112
16Ø DATA ØØØØ3115ØØ, ØØØ32ØØ45Ø, ØØØ1ØØØØ1Ø,
     ØØØ15ØØ31Ø, ØØØ141121Ø
17Ø DATA ØØØ1ØØØØ1Ø, ØØØ1ØØØØ1Ø, 3152ØØØØ1Ø,
     1Ø15ØØØ32Ø, 41241112ØØ
18Ø DATA ØØØ3115ØØØ, ØØ32ØØ45ØØ, ØØ1ØØ3Ø1ØØ,
     ØØ45Ø412ØØ, ØØØ45ØØØØØ
19Ø DATA ØØ312ØØØØØ, Ø32ØØØØØØØ, Ø1ØØØØØØØØ,
     Ø45ØØØØØØ3, ØØ41111112
2ØØ DATA 315ØØØØØØØ, 1Ø41111111, 42ØØØØ1ØØØ,
     ØØØØØØ1ØØØ, ØØØØØ312ØØ
21Ø DATA ØØØØØØ1ØØØ, ØØ1111111Ø, ØØ1ØØØ1Ø1Ø,
     ØØ45ØØ1Ø4Ø, ØØØ4112ØØØ
22Ø DATA 315ØØØ31ØØ, 1Ø1ØØ321ØØ, 411112Ø1ØØ,
     ØØ1ØØØØ1ØØ, ØØ1ØØØØ1ØØ
23Ø DATA ØØ1ØØØØ1ØØ, Ø41112Ø1ØØ, Ø315ØØØ1ØØ,
     32Ø45Ø32ØØ, 2ØØØ412ØØØ
24Ø DATA Ø315ØØØØ1Ø, Ø1Ø1ØØØØ1Ø, Ø421ØØØØ1Ø,
     ØØØ1ØØØØ1Ø, ØØØ1ØØØØ1Ø
25Ø DATA ØØØ1411111, ØØØ1Ø45Ø1Ø, ØØØ1ØØ451Ø,
     ØØ32ØØØ41Ø, Ø32ØØØØØ4Ø
26Ø DATA ØØØØ315ØØ, ØØØØ32Ø1ØØ, ØØØØ1ØØ1ØØ,
     ØØØØ1ØØ1ØØ, ØØØØ1ØØ1ØØ
27Ø DATA 111Ø1ØØ1ØØ, 45ØØ1ØØ1ØØ, Ø45Ø4532ØØ,
     ØØ45Ø11ØØ3, ØØØ4124112
28Ø DATA ØØØ3115ØØØ, ØØ32ØØ1ØØØ, ØØ1ØØØ1Ø32,
     ØØ1ØØØ132Ø, ØØ45ØØ12ØØ
29Ø DATA ØØØ4131ØØØ, ØØØØ321ØØØ, ØØØØ1Ø1ØØØ,
     ØØØØ1Ø1ØØØ, ØØØØ412ØØØ
3ØØ DATA Ø315ØØØØ1Ø, Ø1Ø1ØØØØ1Ø, Ø421ØØØ32Ø,
     ØØØ1ØØ32ØØ, ØØØ1112ØØØ
31Ø DATA ØØØ145ØØØØ, ØØØ1Ø45ØØØ, ØØØ1ØØ45ØØ,
     ØØØ1ØØØ45Ø, ØØØ1ØØØØ45
32Ø DATA ØØØØ315ØØØ, ØØØ32Ø1ØØØ, ØØØ1Ø32ØØØ,
     Ø41112ØØØØ, ØØØ1ØØØØØØ
33Ø DATA ØØØ1ØØØØØØ, Ø3115ØØØØØ, 32Ø145ØØØØ,
     1Ø32Ø45ØØØ, 412ØØØ45ØØ
34Ø DATA 315ØØØØØØØ, 1Ø1315315Ø, 4212Ø12Ø45,
     ØØ1ØØ1ØØØ1, ØØ1ØØ1ØØØ1
35Ø DATA ØØ1ØØ1ØØØ1, ØØ1ØØ1ØØØ1, ØØ1ØØ1ØØØ1,
     ØØ1ØØØØØØ1, ØØ1ØØØØØØ1
36Ø DATA 315ØØØØØØØ, 1Ø13115ØØØ, 4212ØØ1ØØØ,
     ØØ1ØØØ1ØØØ, ØØ1ØØØ1ØØØ
37Ø DATA ØØ1ØØØ1ØØØ, ØØ1ØØØ1ØØØ, ØØ1ØØØ1ØØØ,
     ØØ1ØØØ45Ø3, ØØ1ØØØØ412
38Ø DATA ØØØ3115ØØØ, ØØ31ØØ45ØØ, Ø321ØØØ45Ø,
     Ø1Ø45ØØØ1Ø, Ø1ØØ41111Ø
39Ø DATA Ø1ØØØØØØ1Ø, Ø1ØØØØØØ1Ø, Ø45ØØØØ32Ø,
     ØØ45ØØ32ØØ, ØØØ4112ØØØ
4ØØ DATA ØØØØ3115ØØ, ØØ312ØØ45Ø, Ø321ØØØØ1Ø,
     12Ø1ØØØØ1Ø, ØØØ1ØØØØ1Ø
41Ø DATA ØØØ15ØØ32Ø, ØØØ14112ØØ, ØØØ1ØØØØØØ,
     ØØØ1ØØØØØØ, ØØØ1ØØØØØØ
42Ø DATA ØØØ3115ØØØ, ØØ31ØØ45ØØ, Ø321ØØØ45Ø,
     Ø1Ø45ØØØ1Ø, Ø1ØØ41111Ø
43Ø DATA Ø1ØØØØØØ1Ø, Ø1ØØØØØØ1Ø, Ø45Ø41532Ø,
     ØØ45ØØ115Ø, ØØØ4112Ø42
44Ø DATA ØØØØ3115ØØ, ØØ312ØØ45Ø, Ø321ØØØØ1Ø,
     12Ø1ØØØØ1Ø, ØØØ1ØØØØ1Ø
45Ø DATA ØØØ15ØØ32Ø, ØØØ14112ØØ, ØØØ1Ø45ØØØ,
     ØØØ1ØØ45Ø3, ØØØ1ØØØ412
46Ø DATA ØØØØ315ØØØ, ØØØØ1Ø1ØØØ, ØØØØ1Ø1ØØØ,
     ØØØØ412ØØØ, ØØØØ315ØØØ
47Ø DATA ØØØ32Ø45ØØ, Ø41112Ø1ØØ, Ø315ØØØ1ØØ,
     32Ø45ØØ1ØØ, 2ØØØ4112ØØ
48Ø DATA 315ØØØØØØØ, 1Ø4111111Ø, 42ØØØØ1ØØØ,
     ØØØØØØ1ØØØ, ØØØØØØ1ØØØ
49Ø DATA ØØØØØØ1ØØØ, ØØ111Ø1ØØØ, ØØ1ØØØ1ØØØ,
     ØØ45ØØ1ØØØ, ØØØ4112ØØØ
5ØØ DATA 315ØØØØ1ØØ, 1Ø1ØØØØ1ØØ, 421ØØØØ1ØØ,
     ØØ1ØØØØ1ØØ, ØØ1ØØØØ1ØØ
51Ø DATA ØØ1ØØØØ1ØØ, ØØ1ØØØØ1ØØ, ØØ1ØØØ31ØØ,
     ØØ45Ø321Ø3, ØØØ412Ø412
52Ø DATA 315ØØØØØØ3, 1Ø45ØØØØ32, 4121ØØØØ1Ø,
     ØØØ1ØØØØ1Ø, ØØØ1ØØØØ1Ø
53Ø DATA ØØØ1ØØØØ1Ø, ØØØ1ØØØØ1Ø, ØØØ45ØØ32Ø,
     ØØØØ4532ØØ, ØØØØØ42ØØØ
54Ø DATA 315ØØØØØØ3, 1Ø1ØØØØØ32, 411ØØØØØ1Ø,
     ØØ1ØØ1ØØ1Ø, ØØ1ØØ1ØØ1Ø
55Ø DATA ØØ1ØØ1ØØ1Ø, ØØ1ØØ1ØØ1Ø, ØØ1ØØ1ØØ1Ø,
     ØØ1Ø315Ø1Ø, ØØ412Ø412Ø
56Ø DATA 315ØØØØØØ3, 1Ø1ØØØØØ32, 4245ØØØ32Ø,
     ØØØ45Ø32ØØ, ØØØØ412ØØØ
57Ø DATA ØØØØ315ØØØ, ØØØ32Ø45ØØ, ØØ32ØØØ45Ø,
     Ø32ØØØØØ45, 32ØØØØØØØ4
58Ø DATA ØØØØØØØØØØ, 315ØØØ1ØØØ, 4Ø1ØØØ1ØØØ,
     ØØ1ØØØ1ØØØ, ØØ1ØØ3132Ø
59Ø DATA ØØ411212ØØ, ØØØØ31ØØØ, ØØØ321ØØØ,
     ØØØØ1Ø1ØØØ, ØØØØ412ØØØ
6ØØ DATA ØØØØØØØØØØ, ØØ3115ØØØØ, Ø32ØØ1ØØØØ,
     Ø1ØØ3232ØØ, ØØØØ132ØØØ
61Ø DATA ØØØØ41ØØØØ, ØØØØ315ØØØ, ØØØØ1Ø1ØØØ,
     ØØØØ1Ø1ØØØ, ØØØØ412ØØØ
62Ø DATA ØØØ3115ØØØ, ØØ32ØØ45ØØ, ØØ1ØØØØ1ØØ,
     ØØØØØØ32ØØ, ØØØØ312ØØØ
63Ø DATA ØØØØ1ØØØØØ, ØØØØ1ØØØØØ, ØØØØ1ØØØØØ,
     ØØØØØØØØØØ, ØØØØ1ØØØØØ
64Ø DATA ØØØØØØØØØØ, ØØØØØØØØØØ, ØØØØØØØØØØ,
     ØØØØØØØØØØ, ØØØØØØØØØØ
65Ø DATA ØØØØØØØØØØ, ØØØØØØØØØØ, ØØØØØØØØØØ,
     ØØØØ11ØØØØ, ØØØØ11ØØØØ
66Ø DATA ØØØØØØØØØØ, ØØØØ111ØØØ, ØØØØ111ØØØ,
     ØØØØ111ØØØ, ØØØØ111ØØØ
67Ø DATA ØØØØ412ØØØ, ØØØØØ1ØØØØ, ØØØØØ1ØØØØ,
     ØØØØØØØØØØ, ØØØØØ1ØØØØ
68Ø DATA ØØØØØØØØØØ, ØØØØØØØØØØ, ØØØØØØØØØØ,
     ØØØØØØØØØØ, ØØØØØØØØØØ
69Ø DATA ØØØØØØØØØØ, ØØØØØØØØØØ, ØØØØ11ØØØØ,
     ØØØØ11ØØØØ, ØØØØ32ØØØØ
7ØØ DATA ØØØØØØØØØØ, ØØØØØØØØØØ, ØØØØØØØØØØ,
     ØØØØØØØØØØ, ØØØØØØØØØØ
71Ø DATA ØØØØØØØØØØ, ØØØØØØØØØØ, ØØØØØØØØØØ,
     ØØØØØØØØØØ, ØØØØØØØØØØ
```



Merge BASIC Programs on the MC-10

James P. Jones
496 Amboy Ave.
Perth Amboy, NJ 08861

This program will allow you to merge two or more BASIC programs on the MC-10.

```
1 REM MERGE BASIC PROGRAMS
2 REM FOR MC-1Ø
3 REM by JAMES P. JONES
4 REM 496 AMBOY AVE.
5 REM PERTH AMBOY, NJ Ø8861
6 REM [7Ø575,1273] COMPUSERVE NUMBER
7 REM make sure that program #2 has
8 REM larger line numbers than #1.
9 REM ENJOY!
1Ø B=Ø:FOR X = 17152 TO 17168
2Ø READ A
    : POKE X,A
    : B = B + A
3Ø NEXT
4Ø IF B <> 1895 THEN PRINT "DATA ERROR"
    : STOP
5Ø DATA 222,147,255,66,17,222,149,9,9
6Ø DATA 32,3,254,66,17,223,147,57
7Ø PRINT "1-EXEC 17152"
8Ø PRINT "2-CLOAD(NEW PROGRAM)"
9Ø PRINT "3-EXEC 17163
1ØØ PRINT "*LINE NOS. MUST BE HIGHER IN 2ND PROGRAM*"
```

For you assembly language programmers, I have enclosed an assembly listing of Merge.

```
                 ØØ93    ØØ1ØØ BASBEG   EQU    $ØØ93
                 ØØ95    ØØ11Ø BASEND   EQU    $ØØ95
                         ØØ12Ø *BASIC MERGE PROGRAM
                         ØØ13Ø *FOR THE MC-1Ø
                         ØØ14Ø *MICRO COLOR COMPUTER
                         ØØ15Ø *WRITTEN BY:
                         ØØ16Ø *JAMES P. JONES
                         ØØ17Ø *SEPT 1983
                         ØØ18Ø *
                         ØØ19Ø *
43ØØ                     ØØ2ØØ          ORG    $43ØØ
43ØØ   DE    93          ØØ21Ø          LDX    BASBEG
43Ø2   FF    4311        ØØ22Ø          STX    TEMPAD
43Ø5   DE    95          ØØ23Ø          LDX    BASEND
43Ø7   Ø9                ØØ24Ø          DEX
43Ø8   Ø9                ØØ25Ø          DEX
43Ø9   2Ø    Ø3          ØØ26Ø          BRA    GOBACK
43ØB   FE    4311        ØØ27Ø  MERGE   LDX    TEMPAD
43ØE   DF    93          ØØ28Ø  GOBACK  STX    BASBEG
431Ø   39                ØØ29Ø          RTS
4311                     ØØ3ØØ  TEMPAD  RMB    2
       43ØØ              ØØ31Ø          END    START
*
ØØØØ   TOTAL ERRORS
*
BASBEG  ØØ93
BASEND  ØØ95
GOBACK  43ØE
MERGE   43ØB
TEMPAD  4311
```

Variable Swapper

Doug Faulkner, Jr.
106 Baugh Avenue
Hogansville, GA 30230

Since variable swapping techniques have been at issue in the past few newsletters, I would like to add one of my own. This subroutine works with any two single precision, double precision, integer, or string variables. It will not work with arrays as it is presented here. Only two prerequisites are required:

1. DIMension an integer array for a maximum of eight—DIM A%(8)
2. Both variables must be of the same type.

This program was written for a Model III 48K computer.

The subroutine uses an integer array to hold the decimal code for the machine language version of a memory move. A%(1) holds the from address, A%(3) holds the to address, and A%(5) holds the number of bytes to move. This routine will work for any memory move you wish to perform.

Executing line 1000 will swap b and c, if they are of any type except array variables.

Note: &HE000 is an arbitrary memory location. Any other unused location is acceptable (including the video portion).

```
Ø '- - Sample variable swapping program - -
1 '- - By : Doug Faulkner, Jr.          - -
2 '- - No rights reserved               - -
1Ø CLS
     : CLEAR 1ØØØ
2Ø DEFINT A
25 PRINT "- - - - - - - - - Variable swapper sample
     program - - - - - - -"
26 PRINT STRING$(63,126)
3Ø INPUT "Which type of variable do you want to
     swap....
    1). Integer
    2). Single precision
    3). Double precision
    4). String (note: they can be of varying lengths)
    Enter choice ";TYPE
4Ø IF (TYPE < 1 OR TYPE > 4) THEN CLS
    : GOTO 3Ø
5Ø ON TYPE GOTO 6Ø,7Ø,8Ø,9Ø
6Ø DEFINT B - C
    : GOTO 1ØØ
7Ø DEFSNG B - C
    : GOTO 1ØØ
8Ø DEFDBL B - C
    : GOTO 1ØØ
9Ø DEFSTR B - C
    : GOTO 1ØØ
1ØØ INPUT "Enter the value for B ";B
11Ø INPUT "Enter the value for C ";C
12Ø PRINT "B = ";B
     : PRINT "C = ";C
125 PRINT STRING$(63,126)
13Ø GOSUB 1ØØØ
14Ø PRINT "B = ";B
     : PRINT "C = ";C
15Ø INPUT "Try another (Y/N) ";Z$
     : IF Z$ = "Y" THEN 1Ø ELSE STOP
1ØØØ A(Ø) = 8448
      : A(2) = 4352
      : A(4) = 256
      : A(6) = -2Ø243
      : A(7) = 2Ø1
      : A(1) = VARPTR(B)
      : A(3) = &HEØØØ
      : A(5) = PEEK (VARPTR(B) - 3)
      : DEFUSRØ = VARPTR(A(Ø))
      : A(8) = USR(Ø)
      : A(1) = VARPTR(C)
      : A(3) = VARPTR(B)
      : A(8) = USR(Ø)
      : A(1) = &HEØØØ
      : A(3) = VARPTR(C)
      : A(8) = USR(Ø)
1Ø1Ø RETURN
```

Decimal to Binary Conversion Program

Josh Bloomstone
17 Crestwood Avenue
Montreal, Que. Canada H4X IN3

I have been studying both Computer Science and Data Processing for four years. Three months ago, I endeavored to learn how to program in Z-80 Assembler Language. I went out to my local Radio Shack Computer Center and bought both the Series-I Editor Assembler and the book "How To Program The Z-80."

I found that in order to program in Assembly, I needed to further my knowledge in how to use the Binary Coded Decimal system.

Frustrated with always having to use pencil and paper to convert my decimals to binary, I sat down with my Model III and wrote a program to solve my conversion problem.

```
Ø  CLS
1  '!!!!!!!!!!!!!!!!!
2  'DECIMAL TO BINARY
3  '  CONVERSION
4  'JOSH BLOOMSTONE
5  '  MONTREAL
6  'JUNE 13, 1983
7  '!!!!!!!!!!!!!!!!!
8  PRINT CHR$(23)
     : PRINT TAB(1Ø) "DECIMAL TO BINARY"
     : PRINT TAB(11) "CONVERSION"
     : PRINT TAB(12) "CALCULATION"
9  FOR H4 = 1 TO 1ØØØ
     : NEXT
     : CLS
1Ø CLS
     : C = Ø
2Ø INPUT "Would you like a CALculation or CONversion
     or @ for end";A$
21 IF A$ = "@" THEN END
3Ø IF A$ = "CAL" THEN 1ØØ
31 IF A$ <> "CAL" AND A$ <> "CON" THEN 1Ø
4Ø INPUT "What is your number";NUM
5Ø PRINT NUM;
51 PRINT
6Ø FOR Y = 15 TO Ø STEP -1
7Ø M = INT (NUM/2[Y)
8Ø PRINT M;
9Ø L = NUM - M * 2[Y
91 NUM = L
92 NEXT Y
93 PRINT
     : INPUT "<ENTER> TO GO ON";LKJ
     : GOTO 1Ø
1ØØ CLS
     : INPUT "What will your calculation be Addition
     or Multiplication";B$
```

```
110 IF B$ = "A" THEN 200
111 IF B$ <> "A" AND B$ <> "M" THEN 100
120 GOSUB 500
121 C = 1
130 FOR X = 1TOA
131 PRINT "INPUT # "X
132 INPUT B(X)
140 C = C * B(X)
150 NEXT
151 NUM = C
160 GOTO 50
200 GOSUB 500
210 FOR X = 1 TO A
211 PRINT "INPUT # "X
212 INPUT B(X)
220 C = C+B(X)
230 NEXT
240 NUM = C
250 GOTO 50
500 INPUT "How many numbers are there";A
501 RETURN
```

Labelmaker

Bud Myers
2 Church Street
Washburn, ME 04786-0498

Unable to afford a disk system, I am forced to organize my cassettes in order to use them efficiently. This is my method; perhaps others will also find it to be of some use.

Several programs on one cassette are economical. Unless I know just where each tape is positioned, however, locating a given program can take enough time to offset any monetary savings. Therefore, I always include the recorder counter dial reading on my cassette labels. By marking the position of the tape on each cassette before returning it to storage, I have a starting point for the next use.

Suppose I need a program on cassette "A." The index tells me the program I want is at dial reading 075. The index mark is at 045. I look at the current dial reading on the recorder. If it is less than 045, I fast forward again to 075. I then load the program.

If the dial reading on the recorder is greater than the index mark on the cassette, I first reset the counter to zero, and then proceed as above. Since the dial does not change on rewind without a cassette in place, this is one of two alternatives. The other is to keep a useless cassette handy. When this seems best, I put it in the recorder, rewind to the index setting, and proceed as above. Old audio cassettes are fine for this use.

I store all my program tapes in hard cases. I buy only those blank cassettes which include such cases. For commercially obtained programs which do not, I buy empty cases in which to store them.

I use grease pencil (china marker) to place the Index mark on each cassette showing the dial reading of the tape. These marks are easily changed when the tapes inside are repositioned.

To produce neat labels for both the case and the cassette inside, I use the program in Listing 1. It works on either Model I or Model III systems with 16 K or larger memories. With a Lineprinter II, IV or VIII, it will index 24 programs per cassette—12 on each side. I use both sides of my cassettes, so this program indexes both sides and prints a label for each.

I use different colored construction paper for each category of programs: yellow for utilities, green for games, gray for graphics, purple for word processing, etc. This also helps prevent putting cassettes in the wrong case when I'm rushed. The paper I use is nine inches wide and does not fit between the sprocket pins of my printer, so I trim it to 8 1/2 inches before beginning.

Using the program is largely a matter of responding to prompts. There are six for each line—three for each side, A and B. The first, "Counter?", accepts either numeric or non-numeric responses. I use leading zeros for numbers of less than three digits. When I want to print nothing for all three fields of either half of a line, I answer this prompt with any character other than a number or the period.

The "File?" prompt accepts any single displayable character. Quotation marks should not be used; the print routine will provide them. I use the symbol under which the program was saved.

"Title?" may be any combination of upper- and lower-case symbols up to 20.

Following the final title on side B, I respond to the "Counter?" prompt with a period ("."). This stops the indexing and brings up the final prompt. "Label?" is for the title identifying the entire collection of programs on the cassette. It will appear on both the case label and those for the two sides of the cassette itself. It may be up to eighteen characters in length.

After printing one set of labels, I remove and invert the paper, then print another set on the same sheet. I cut them all out on the printed lines. The case label is folded and slipped inside the case. The cassette labels are attached with rubber cement. I have occasionally put the label reading "Side A" on the wrong side of the cassette. Rubber cement allows me to correct such mistakes fairly easily. I hope you will be more careful than I.

I can manage my two cassette-based systems and their ninety-odd cassettes fairly well with these procedures. I'm still saving up for disk drives in the future. I'll try to modify the program to produce labels for diskettes at that time.

```
0  GOTO 44 '      LABELMAKER BY BUD MYERS
1  CLEAR 1500
     : DEFINT I,L,Z
     : DEFSTR C,F,P - R,T - W
     : P = CHR$(27)
     : Q = CHR$(34)
     : R = P + CHR$(19)
     : TH = P + CHR$(28)
     : TE = P + CHR$(15)
     : V = CHR$(124)
     : W = P + CHR$(20)
     : TX = P + CHR$(14)
     : QB = CHR$(34) + " " + CHR$(34)
2  LPRINT W STRING$(62,45)TH; V
3  LPRINT TX "SIDE A:        SIDE B:         "TE; V
4  LPRINT STRING$(62,32) V
     : L = 2
5  INPUT "COUNTER"; CA
     : IF ASC(CA) > 47 AND ASC(CA) < 58 THEN 8
6  IF CA = "." THEN 19
7  CA = " "
     : FA = " "
     : TA = " "
     : GOTO 10
8  INPUT "FILE"; FF
     : IF FF = "'" THEN FA = QB ELSE FA = Q + FF + Q
9  INPUT "TITLE"; TA
```

```
10 INPUT "COUNTER"; CB
     : IF ASC(CB) > 47 AND ASC(CB) < 58 THEN 12
11 CB = " "
     : FB = " "
     : TB = " "
     : GOTO 14
12 INPUT "FILE"; FF
     : IF FF = "'" THEN FB = QB ELSE FB = Q + FF + Q
13 INPUT "TITLE"; TB
14 LPRINT CA" "FA" "TA;
15 LPRINT TAB(30)CB" "FB" "TB;
16 LPRINT TAB(62) V
17 L = L + 1
18 GOTO 5
19 X = 14 - L
20 FOR Z = 1 TO X
     : LPRINT STRING$(62,32) V
     : NEXT
21 LPRINT R STRING$(37,45)TH
22 INPUT "LABEL"; CL
23 L = LEN(CL)
     : IF L > 20 THEN CL = LEFT$(CL,20)
     : L = 20
24 S = L/2
     : IF S <> INT(S) THEN S = S + .5
25 IF S >  = 10 THEN X = 0 ELSE X = 10 - S
26 LPRINT TX STRING$(X - 1,32)CL
     : LPRINT TH STRING$(37,45)
27 FOR Z = 1 TO 2
     : LPRINT STRING$(37,32) V
     : NEXT
28 LPRINT "." STRING$(34,32)".
29 LPRINT "   ." STRING$(30,32)".
30 LPRINT "     ." STRING$(26,32)".
31 LPRINT "       " STRING$(24,46)
32 LPRINT W STRING$(5,138)
     : T = "A"
33 LPRINT STRING$(3,32) STRING$(53,95)TH
34 LPRINT V STRING$(57,32) V
35 LPRINT V TAB(23)TX"SIDE "T; TE TAB(56) V
     : LPRINT TH
36 LPRINT V STRING$(13,32) STRING$(32,45) TAB(58) V
37 LPRINT TH
     : FOR Z = 1 TO 2
38 LPRINT V STRING$(10,32) V STRING$(36,32) V TAB(58)
     V
39 NEXT
     : LPRINT TH; V STRING$(13,32) STRING$(32,45)
40 LPRINT V; TX TAB(18 - (LEN(CL)/2)); CL
41 LPRINT V STRING$(57,95) V
42 I = I + 1
     : IF I = 1 THEN T = "B"
     : GOTO 33
43 CLS
     : LPRINT R
     : END
44 CLS
     : PRINT
     : PRINT "A PROGRAM TO PRODUCE LABELS FOR HARD
     CASSETTE CASES ON RADIO
45 PRINT "SHACK LINEPRINTER IV. UP TO 12 PROGRAMS PER
46 PRINT "SIDE MAY BE LISTED. EACH LISTING CONSISTS
     OF COUNTER DIAL READ-
47 PRINT "ING, FILE REFERENCE SYMBOL AND TITLE. THE
     COMBINED LENGTH OF
48 PRINT "EACH LISTING, INCLUDING SPACES, IS 30.
     LABELS FOR THE CASSETTE
49 PRINT "ITSELF ARE ALSO PRINTED.
50 PRINT
     : PRINT "ANY KEY TO CONTINUE.
51 U$ = INKEY$
     : IF U$ = "" THEN 51
52 CLS
     : PRINT "BE SURE A PRINTER IS CONNECTED, TURNED
     ON AND CONTAINS PAPER.
53 PRINT
     : PRINT "ANY KEY TO BEGIN.
54 U$ = INKEY$
     : IF U$ = "" THEN 54
55 CLS
     : GOTO 1
```

Short Directory Program

Dr. Thomas R. W. Longstaff
39 Pleasant Street
Waterville, ME 04901

I've been a TRS-80 user since 1979 when I first took a Model I to the Upper Galilee (Israel) for use in building a data base for on-site processing of information at an archaeological site. I have recently "up-graded" to a Model III system with which I am very happy.

Although I have purchased some good software packages, I have not yet replaced the "Master Directory" program which I purchased for my Model I. A few days ago I wanted a list of the programs and files which I had saved to disk and it occurred to me that I could produce such a list quickly and easily with a simple program. It may surprise some new users to see just how easy the task was! The "one liner" below allows the user to insert disks sequentially into Drive 1 of a two drive system and produce hard copy directories for each diskette in turn. The program could easily be adapted for a single drive system.

```
10 FOR X = 1 TO 50
     : LPRINT
     : LPRINT "DIRECTORY FOR DISKETTE NO.  "; X
     : CMD "Z", "ON"
     : CMD "D:1"
     : CMD "Z", "OFF"
     : INPUT "CHANGE DISKETTE IN DRIVE 1; PRESS ENTER
     TO CONTINUE; BREAK TO END"; C
     : NEXT
```

DO File Directory

Mark Rife
5405 North Charles Street
Baltimore, MD 21210

I was introduced to the TRS-80 in 8th grade, I was impressed with the capabilities of this computer. About a year ago my family bought a basic 16K cassette Model III. We quickly outgrew the system, and, as I am now entering 10th grade, we are now the proud owners of a 48K two drive Model III! In this time, I have accumulated disks upon disks full of programs. I decided I would create a program that would print a directory, a free space map, and information about each diskette in one easy step. Monthly, I run the program; so I have up-to-date printed records.

This program will work on any system that uses TRSDOS 1.3 commands and has one or more disk drives and any printer. *(Editor's Note: We tested this program on a 48K two disk drive Model III with a DMP-2100.)* To set yourself up, type the disk BASIC program in and save it.

The main program is run from disk BASIC. After you tell how many drives you have, the diskette number, the code, the date, your name, and any comments, all the information is printed. Then using the CMD "I" command, a DO file is accessed from TRSDOS. This file asks for the diskette to be reported to be entered in the drive and for you to press ENTER. It then prints the directory and free space map. The file executes BASIC * and asks if you want to report any more diskettes.

```
1Ø ' **********************************
2Ø ' *  BY MARK RIFE                   *
3Ø ' *     54Ø5 NORTH CHARLES STREET *
4Ø ' *     BALTIMORE, MD  2121Ø       *
5Ø ' **********************************
6Ø CLEAR 1ØØØ
7Ø GOSUB 65Ø
8Ø I$ = INKEY$
9Ø PRINT@21Ø, "NUMBER OF DRIVES: <1> OR <2>"
     : FOR I = 1 TO 1ØØ
     : NEXT I
     : PRINT@229, " ";
     : PRINT@236, " ";
     : FOR I = 1 TO 5Ø
     : NEXT I
1ØØ IF I$ = "" THEN 8Ø
11Ø DR = VAL(I$)
12Ø IF DR > 2 OR DR < 1 THEN 8Ø
13Ø GOSUB 65Ø
14Ø C$ = CHR$(3Ø)
15Ø PRINT@192, "---> DISKETTE NUMBER";C$;
     : INPUT DN$
16Ø L$ = LEFT$(DN$,1)
17Ø IF L$ > "9" OR L$ < "Ø" THEN 15Ø
18Ø IF LEN(DN$) > 1Ø THEN 15Ø
19Ø PRINT@256, "---> CODE (UP TO 3 CHARACTERS)";C$;
     : INPUT CD$
2ØØ IF LEN(CD$) > 3 OR LEN(CD$) = Ø THEN 19Ø
21Ø PRINT@32Ø, "---> TODAY'S DATE ";C$;
     : LINEINPUT DA$
22Ø IF LEN(DA$) > 2Ø OR LEN(DA$) = Ø THEN 21Ø
23Ø PRINT@384, "---> YOUR NAME";C$;
     : INPUT NA$
24Ø IF LEN(NA$) > 2Ø OR LEN(NA$) = Ø THEN 23Ø
25Ø PRINT@448, "---> COMMENTS (IF NONE PRESS
     <ENTER>):";C$
26Ø LINEINPUT  CO$
27Ø IF LEN(CO$) = Ø THEN CO = 1
28Ø I$ = INKEY$
29Ø PRINT@976, "IS THIS INFORMATION CORRECT (Y/N)?";
     : FOR I = 1 TO 1ØØ
     : NEXT I : PRINT@976, STRING$(35," ");
     : FOR I = 1 TO 5Ø
     : NEXT I
3ØØ IF I$ <> "Y" AND I$ <> "N" THEN 28Ø
31Ø IF I$ = "Y" THEN 34Ø
32Ø IF I$ = "N" THEN PRINT@974, "TO REINPUT THE
     INFORMATION, HIT <ENTER>";
33Ø I$ = INKEY$
     : IF I$ = "" THEN 33Ø ELSE DN$ = ""
     : CD$ = ""
     : DA$ = ""
     : NA$ = ""
     : CO$ = ""
     : GOTO 13Ø
34Ø GOSUB 65Ø
35Ø I$ = INKEY$
36Ø PRINT@2Ø2, "PLEASE ALIGN THE PRINTER AND HIT
     <ENTER>";
     : FOR I = 1 TO 1ØØ
     : NEXT I
     : PRINT@2Ø2, STRING$(42," ")
     : FOR I = 1 TO 5Ø
     : NEXT I
37Ø IF I$ = "" THEN 35Ø
38Ø PRINT@192, "DISKETTE NUMBER:  "; DN$;C$
39Ø LPRINT "DISKETTE REPORTER"
4ØØ LPRINT "DISKETTE NUMBER:  "; TAB(2Ø);DN$
41Ø PRINT@256, "DISKETTE CODE:  ";CD$
42Ø LPRINT "DISKETTE CODE:  "; TAB(2Ø);CD$
43Ø PRINT@32Ø, "DATE OF RECORD:  ";DA$
44Ø LPRINT "DATE OF RECORD:  "; TAB(2Ø);DA$
45Ø PRINT@384, "NAME OF RECORDER:  ";NA$
46Ø LPRINT "NAME OF RECORDER:  "; TAB(2Ø);NA$
47Ø IF CO = 1 THEN 52Ø
48Ø PRINT@448, "COMMENTS:  "
     : PRINT CO$
49Ø LPRINT "COMMENTS:"
5ØØ LPRINT CO$
51Ø LPRINT
52Ø LPRINT "----------------------------------------
     -----------------------"
53Ø LPRINT
54Ø FOR I = 1 TO 5ØØ
     : NEXT I
55Ø IF DR = 1 THEN CMD"I", "DO DRIVE1/BLD"
56Ø IF DR = 2 THEN CMD"I", "DO DRIVE2/BLD"
59Ø GOSUB 65Ø
6ØØ I$ = INKEY$
61Ø PRINT@2Ø1, "WOULD YOU LIKE TO MAKE ANY MORE
     REPORTS (Y/N)?"
     : FOR I = 1 TO 1ØØ
     : NEXT I
     : PRINT@2Ø1, STRING$(5Ø," ")
     : FOR I = 1 TO 5Ø
     : NEXT I
62Ø IF I$ = "" THEN 6ØØ
63Ø IF I$ = "Y" THEN 13Ø
64Ø IF I$ = "N" THEN CLS
     : END
65Ø CLS
66Ø PRINT@23, "DISKETTE REPORTER"
67Ø PRINT@64, STRING$(64,95)
68Ø RETURN
```

After saving the program, type in the DO files, DRIVE1/BLD and DRIVE2/BLD, using the BUILD command.

```
BUILD DRIVE1 <ENTER>

Hit BREAK to exit
Type in up to 63 Characters
CLS <ENTER>
Type in up to 63 Characters
PAUSE ***ENTER DISKETTE TO BE REPORTED IN DRIVE #Ø***
     <ENTER>
Type in up to 63 Characters
DIR :Ø (PRT) <ENTER>
Type in up to 63 Characters
FREE :Ø (PRT) <ENTER>
Type in up to 63 Characters
BASIC * <ENTER>
Type in up to 63 Characters
GOTO 59Ø <ENTER>
Type in up to 63 Characters
<BREAK>

BUILD DRIVE2 <ENTER>

Hit BREAK to exit
Type in up to 63 Characters
CLS <ENTER>
Type in up to 63 Characters
PAUSE ***ENTER DISKETTE TO BE REPORTED IN DRIVE #1***
     <ENTER>
Type in up to 63 Characters
DIR :1 (PRT) <ENTER>
Type in up to 63 Characters
FREE :1 (PRT) <ENTER>
Type in up to 63 Characters
BASIC * <ENTER>
Type in up to 63 Characters
GOTO 59Ø <ENTER>
```

```
Type in up to 63 Characters
<BREAK>
```

Magazine Index

Case Larsen
115 Bixby Drive
Milpitas, CA 95035

The first program creates an index of magazine articles by Category, Title, Author, Code, Year, Page Number and Month of the article.

```
5 CLEAR 5ØØ
1Ø OPEN "O",1,"FILENAME/EXT"
2Ø INPUT "CATEGORY";C$
3Ø INPUT "TITLE";T$
4Ø INPUT "AUTHOR";A$
5Ø INPUT "CODE";CD$
55 INPUT "YEAR";YR
57 INPUT "PAGE NUMBER";P
58 INPUT "MONTH";MO$
6Ø INPUT "IS THIS RIGHT";ZZ$
7Ø IF ZZ$ = "Y" THEN 9Ø
8Ø IF ZZ$ = "N" THEN 2Ø
85 IF ZZ$ <> "Y" OR ZZ$ <> "N" THEN 6Ø
9Ø PRINT #1,C$; ","; T$; ","; A$; ","; CD$; ","; YR;
    ","; P; ","; MO$
1ØØ INPUT "AGAIN Y/N";ZZ$
11Ø IF ZZ$ = "Y" THEN 2Ø
12Ø IF ZZ$ = "N" THEN CLOSE
    : END
13Ø IF ZZ$ <> "Y" OR ZZ$ <> "N" THEN 1ØØ
```

If you wish to add more records to the file, change the file mode in Line 10 to extend (E) instead of sequential output (O).

```
1Ø OPEN "E",1,"FILENAME/EXT"
```

The following program accesses the index by Category, Title, Author, Code, Year, Page Number, or Month of the article.

```
5 CLEAR 5ØØØ
1Ø CLOSE
    : OPEN "I",1,"FILENAME/EXT"
2Ø INPUT "WHICH ONE C(ATEGORY), T(ITLE), A(UTHOR),
    CO(DE), Y(EA)R, P(AGE NUMBER), M(ONTH),
    E(ND)";Q$
4Ø IF Q$ = "C" THEN GOTO 1ØØ
5Ø IF Q$ = "T" THEN GOTO 2ØØ
6Ø IF Q$ = "A" THEN GOTO 3ØØ
7Ø IF Q$ = "CO" THEN GOTO 4ØØ
8Ø IF Q$ = "YR" THEN GOTO 5ØØ
9Ø IF Q$ = "P" THEN GOTO 6ØØ
95 IF Q$ = "M" THEN GOTO 7ØØ
97 IF Q$ = "E" THEN END
98 GOTO 2Ø
1ØØ INPUT "CATEGORY";W$
1Ø5 IF EOF(1) THEN PRINT "END OF INDEX FILE"
    : GOTO 1Ø
11Ø INPUT #1,C$,T$,A$,CD$,YR,P,MO$
12Ø IF W$ = C$ THEN PRINT "CATEGORY: ";C$
    : PRINT "TITLE: ";T$
    : PRINT "AUTHOR: ";A$
    : PRINT "CODE: ";CD$
    : PRINT "YEAR: ";YR
    : PRINT "PAGE NUMBER: ";P
    : PRINT "MONTH: ";MO$
13Ø GOTO 1Ø5
2ØØ INPUT "TITLE";X$
2Ø5 IF EOF(1) THEN PRINT "END OF INDEX FILE"
    : GOTO 1Ø
21Ø INPUT #1,C$,T$,A$,CD$,YR,P,MO$
22Ø IF X$ = T$ THEN PRINT "CATEGORY: ";C$
    : PRINT "TITLE: ";T$
    : PRINT "AUTHOR: ";A$
    : PRINT "CODE: ";CD$
    : PRINT "YEAR: ";YR
    : PRINT "PAGE NUMBER: ";P
    : PRINT "MONTH: ";MO$
23Ø GOTO 2Ø5
3ØØ INPUT "AUTHOR";Y$
3Ø5 IF EOF(1) THEN PRINT "END OF INDEX FILE"
    : GOTO 1Ø
31Ø INPUT #1,C$,T$,A$,CD$,YR,P,MO$
32Ø IF Y$ = A$ THEN GOSUB 1ØØØ
33Ø GOTO 3Ø5
4ØØ INPUT "CODE";Z$
4Ø5 IF EOF(1) THEN PRINT "END OF INDEX FILE"
    : GOTO 1Ø
41Ø INPUT #1,C$,T$,A$,CD$,YR,P,MO$
42Ø IF Z$ = CD$ THEN GOSUB 1ØØØ
43Ø GOTO 4Ø5
5ØØ INPUT "YEAR";U
5Ø5 IF EOF(1) THEN PRINT "END OF INDEX FILE"
    : GOTO 1Ø
51Ø INPUT #1,C$,T$,A$,CD$,YR,P,MO$
52Ø IF U = YR THEN GOSUB 1ØØØ
53Ø GOTO 5Ø5
6ØØ INPUT "PAGE NUMBER";V
6Ø5 IF EOF(1) THEN PRINT "END OF INDEX FILE"
    : GOTO 1Ø
61Ø INPUT #1,C$,T$,A$,CD$,YR,P,MO$
62Ø IF V = P THEN GOSUB 1ØØØ
63Ø GOTO 6Ø5
7ØØ INPUT "MONTH";L$
7Ø5 IF EOF(1) THEN PRINT "END OF INDEX FILE"
    : GOTO 1Ø
71Ø INPUT #1,C$,T$,A$,CD$,YR,P,MO$
72Ø IF L$ = MO$ THEN GOSUB 1ØØØ
73Ø GOTO 7Ø5
1ØØØ PRINT "CATEGORY: ";C$
    : PRINT "TITLE: ";T$
    : PRINT "AUTHOR: ";A$
    : PRINT "CODE: ";CD$
    : PRINT "YEAR: ";YR
    : PRINT "PAGE NUMBER: ";P
    : PRINT "MONTH: ";MO$
1Ø1Ø RETURN
```

Sort Integers Using CMD"0"

J. F. Atherley
35 Faraday Crescent
Deep River, Ontario
Canada K0J 1P0

I realize that many sorting programs have been prepared and discussed, and I hesitated to submit another.

This program is designed to demonstrate and test a subroutine that sorts a list of integers using the disk BASIC procedure for sorting strings, CMD"0". To test the subroutine, a timer was built into the program to time both the whole subroutine and the sorting procedure itself. During several runs of the program, I varied the number of integers sorted and the amount of memory CLEARed for string storage. The results tabulated below demonstrate how efficient the CMD"0" procedure is and the effect of "garbage collections" when processing strings.

SORT TIME IN SECONDS

NO.	CLEAR 10000		CLEAR 20000		CLEAR 25000	
	S/R	CMD"0"	S/R	CMD"0"	S/R	CMD"0"
250	17	1	17	1	16	1
500	88	2	32	2	32	2
750	104	5	48	5	49	4
1000	237	6	209	6	64	6

This program was written for a TRS-80 Model III with 48K and 2 disk drives.

```
1 'SRTINT/PRG
2 'Demonstrates
     1. Subroutine which uses the disk BASIC command,
     CMD"O", to sort a list of positive and/or
     negative integers.
3 '  2. A built-in timing procedure.
1Ø ' Programmed by : J.F.Atherley,
                    35 Faraday Crescent,
                    Deep River, Ontario  KØJ 1PØ
5Ø ' ALGORITHM.
51 ' The program uses the random number generator to
     obtain a list of positive and negative integers,
     the user indicating the number to be selected.
52 ' The unsorted list is displayed, sorted, and the
     the sorted list displayed. The program times the
     actual sorting procedure.
53 ' The timing procedure sets the system clock to
     ØØ:ØØ:ØØ at the start of the interval to be
     timed and converts the clock to seconds at the
     end of the interval.
1ØØ CLEAR 1ØØØØ
     : DEFINT I - Z
     : DIM LI(1ØØØ), LI$(1ØØØ)
     : DEF FNT(I) = PEEK(16919) + 6Ø * (PEEK(1692Ø) +
     6Ø * PEEK(16921))
11Ø CLS
     : INPUT "Number of integers in list "; N
12Ø FOR I = 1 TO N
     : LI(I) = RND(2ØØØØ) - 1ØØØØ
     : NEXT
13Ø GOSUB 5ØØ          ' Display unsorted list of
     integers
14Ø POKE 16921,Ø
     : POKE 1692Ø,Ø
     : POKE 16919,Ø  ' Set system clock to ØØ:ØØ:ØØ
15Ø GOSUB 1ØØØ         ' Sort integer list
16Ø TF = FNT(I) + TF ' Convert system clock to
     seconds
17Ø GOSUB 5ØØ          ' Display sorted list of
     integers
18Ø PRINT " Time required to sort " N " integers = "
     TF
181 PRINT " Time required to sort " N " strings = "
     TS
19Ø END


5ØØ 'Subroutine DISPLAY
     Displays n integers stored in LI(1) -> LI(N) in
     1Ø columns.
51Ø F$ = "######"
52Ø FOR I = 1 TO C INT(N/1Ø)
     : IJ = 1Ø * (I - 1)
     : FOR J = 1 TO 1Ø
     : PRINT USING F$; LI(IJ + J);
     : NEXT J
     : PRINT
     : NEXT I
53Ø IF (N - IJ) < > 1Ø THEN FOR I = IJ + 11 TO N
     : PRINT USING F$; LI(I);
     : NEXT I
54Ø PRINT
     : PRINT
     : RETURN
```

```
1ØØØ 'Subroutine INTEGER SORT
     Calling program must include:
     Integer definition - DEFINT I - N
     Array declaration  - DIM LI(m), LI$(m)
     Data - N (<= m) integers stored in LI(1) ->
     LI(N)
1Ø1Ø 'Returns integer data sorted in ascending order
     in LI(1) -> LI(N).
1Ø2Ø MN = Ø
     : MX = Ø
     : LM = 32767
     : BL$ = "     "
1Ø3Ø FOR I = 1 TO N
     : IF LI(I) < MN THEN MN = LI(I) ELSE IF LI(I) >
     MX THEN MX = LI(I)
1Ø31 NEXT
1Ø4Ø IF (MX - LM) > MN THEN PRINT "OVERFLOW RANGE"
     : STOP
1Ø5Ø FOR I = 1 TO N
     : LI$(I) = RIGHT$ (BL$ + STR$ (LI(I) - MN),5)
     : NEXT
1Ø59 TF = FNT(I)
     : POKE 16921,Ø
     : POKE 1692Ø,Ø
     : POKE 16919,Ø
1Ø6Ø CMD"O", N, LI$(1)
1Ø61 TS = FNT(I)
1Ø7Ø FOR I = 1 TO N
     : LI(I) = VAL (LI$ (I)) + MN
     : NEXT
1Ø8Ø RETURN
```

Print Model III Disk Directory

Mike Salisbury
1711 Skylark Lane
Newport Beach, CA 92660

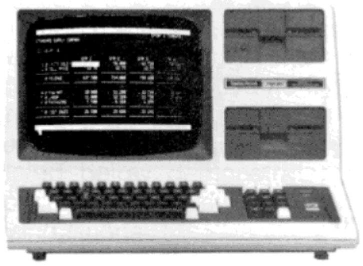
The following program provides you with a directory of your TRS-80 Model III disks. Two drives are required. TRSDOS must be in DRIVE:0.

Insert each disk, as requested into DRIVE:1. You will be asked the name of the disk. After each disk's contents has been read you will be given the choice of inserting another disk or sorting the programs on the disks that have been read.

After sorting, you will be given the option of HARD COPY or DISKSTORAGE.

```
Ø REM --------  27 December 1981  ---------
1 REM --------  MODEL III DISK/DIR  -------
2 REM ---  WRITTEN BY:  Mike Salisbury  ---
3 REM --------  1711 Skylark Lane  --------
4 REM ----  Newport Beach, CA   9266Ø  ----
1Ø CLS
     : CLEAR 1ØØØØ
     : CMD"B", "OFF"
     : DIMF$(5ØØ), DN$(5Ø)
     : A = 1
     : B = 15
     : C = Ø
     : I = Ø
     : UL$ = STRING$(63,14Ø)
     : AB$ = STRING$(22,32) + "For MENU press '@'"
2Ø CLS
     : PRINT STRING$(12,32) + "TRS-8Ø  MODEL III
     DISK DIRECTORY  PROGRAM"
```

```
30 PRINT UL$
    : PRINT@896, UL$
    : PRINT@221, "MENU"
    : PRINT@342, "1  -  Enter disk file names"
    : PRINT@406, "2  -  Print files"
    : PRINT@470, "3  -  Save on disk"
    : PRINT@534, "4  -  Load from disk"
    : PRINT@598, "5  -  Instructions"
    : PRINT@662, "6  -  End"
40 PRINT@790, "Select one:"
50 S$ = INKEY$
    : IF S$ = "" THEN 50 ELSE IF S$ = "1" THEN A = 1
    : GOTO 60 ELSE IF S$ = "2" THEN A = 1
    : GOTO 150 ELSE IF S$ = "3" THEN 210 ELSE IF S$
    = "4" THEN 270 ELSE IF S$ = "5" THEN 400 ELSE IF
    S$ <> "6" THEN 50 ELSE 350
60 CLS
    : PRINT TAB(22)"Enter disk file names"
    : GOSUB 360
    : PRINT@278, "Insert disk in DRIVE:1"
    : PRINT@406, "";
    : INPUT "ENTER disk name";DN$
70 IF DN$ = "@" THEN 20
80 CMD"D:1"
    : PRINT CHR$(15)
90 G = PEEK(15360 + B + C)
    : IF G <> 32 THEN F$(A) = F$(A) + CHR$(G)
    : C = C + 1
    : GOTO 90 ELSE IF C = 0 AND G = 32 THEN 110 ELSE
    IF G = 32 THEN IF PEEK(15375 + B) = 32 THEN B =
    B + 19 ELSE B = B + 15
100 C = 0
    : F$(A) = F$(A) + STRING$(20 - LEN(F$(A)),46) +
    DN$
    : A = A + 1
    : GOTO 90
110 CLS
    : PRINT@325, "Insert next disk and press
    'ENTER'"
    : PRINT
    : PRINT TAB(40) "or"
    : PRINT
    : PRINT TAB(44) "press '*' to SORT"
120 S$ = INKEY$
    : IF S$ = "" THEN 120 ELSE IF S$ = CHR$(13) THEN
    B = 15
    : GOTO 60 ELSE IF S$ = "@" THEN 20 ELSE IF S$ <>
    "*" THEN 120
130 N% = A - 1
140 CMD"O", N%, F$(1)
    : GOTO 20
150 CLS
    : PRINT TAB(27) "Print files"
    : GOSUB 360
    : PRINT@399, "Press 'SPACEBAR' when printer
    ready"
160 S$ = INKEY$
    : IF S$ = "" THEN 160 ELSE IF S$ = "@" THEN 20
    ELSE IF S$ <> " " THEN 160
170 A = 1
    : CLS
    : PRINT@472, "P R I N T I N G"
180 LPRINT
    : LPRINT TAB(5) "TRS-80 MODEL III DISK
    DIRECTORY";
    : LPRINT "   ";
    : LPRINT LEFT$(TIME$,8)
    : LPRINT
    : LPRINT
    : LPRINT TAB(9) "FILE NAMES DISK NAME"
    : LPRINT
    : LPRINT STRING$(49,45)
    : LPRINT
190 LPRINT TAB(10);F$(A)
    : A = A + 1
    : IF F$(A) <> "" THEN 190
200 GOTO 20
210 CLS
    : PRINT TAB(23) "Save files to disk"
    : GOSUB 360
    : PRINT@384, ""
220 PRINT@406, "";
    : LINEINPUT "File name to use ";Z$
230 IF Z$ = "@" THEN 20
240 PRINT
    : PRINT@608, "SAVING to disk"
250 OPEN"O", 1, Z$
260 FOR V = 1 TO A - 1
    : PRINT #1, F$(V)
    : NEXT
    : CLOSE
    : GOTO 20
270 ON ERROR GOTO 340
    : CLS
    : PRINT TAB(25) "Load from disk"
    : GOSUB 360
    : PRINT@384, ""
280 PRINT@406, "";
    : LINEINPUT "Name of disk file ";Z$
290 IF Z$ = "@" THEN 20
300 OPEN"I", 1, Z$
    : A = 1
310 IF EOF(1) THEN 330
320 INPUT #1, F$(A)
    : A = A + 1
    : GOTO 310
330 CLOSE
    : GOTO 20
340 PRINT@608, "Bad file name";
    : FOR I = 1 TO 1000
    : NEXT I
    : RESUME 20
350 CLS
    : PRINT@476, "GOODBYE"
    : CMD"B", "ON"
    : END
360 PRINT UL$
    : PRINT@832, UL$
    : PRINT AB$;
    : RETURN
400 CLS
    : PRINT TAB(16) "DISK DIRECTORY INSTRUCTIONS"
401 PRINT UL$
    : PRINT "This program will provide you with a
    directory of your TRS-80 Model III disks. Two
    drives are required. TRSDOS must be in DRIVE:0"
402 PRINT
    : PRINT "Insert each disk, as requested into
    DRIVE:1. You will be asked the name of the disk.
    After each disk's contents has been read you
    will be given the choice of inserting another
    disk or"
403 PRINT "sorting the programs on the disks that
    have been read."
    : PRINT
    : PRINT "After sorting, you will be given the
    option of HARD COPY or DISKSTORAGE."
    : PRINT UL$
404 INPUT "PRESS ENTER TO CONTINUE";Z$
    : GOTO 20
```



Communication and TRSDOS 6.0

by Carol Morton

Tucked into the TRSDOS 6.0 operating system is a surprising little utility called "COMM." With this utility you can communicate via a device, usually the RS-232 communications line, not only with other local computers but with Bulletin Board Systems, News and Information Systems, Timesharing Systems, and Electronic Mail Services. With COMM you can also communicate with systems that support XON/XOFF (Proceed/Pause) protocol. To the uninitiated, COMM may seem complex but don't be too readily discouraged. This utility is very easy to use once you know your way around its Function Keys.

THE FUNCTION KEYS

COMM operates through the use of two types of Function Keys: Application Keys, which designate a device to which action is to be applied, and Action Keys, which identify the action applied. The CLEAR key is used with each of the Application and Action Keys. This procedure, pressing and holding down the CLEAR key then pressing the desired Action or Application Key, operates as a Function Key. It is through the use of combinations of Application and Action Keys that you initiate communication operations.

Function Keys CLEAR 1 through CLEAR 6 designate the device to which an action will be applied.

Function Key	Device	Abbreviation
CLEAR 1	Keyboard Device	(*KI)
CLEAR 2	Display Device	(*DO)
CLEAR 3	Printer Device	(*PR)
CLEAR 4	Communications Line Device	(*CL)
CLEAR 5	"Data Send" Device	(*FS)
CLEAR 6	"Data Receive" Device	(*FR)

The remaining Function Keys perform an action. Before certain actions can be performed, one of the Application Keys, CLEAR 1 through CLEAR 6, must be specified. The Actions Keys are:

Action Key	Operation
CLEAR 7	"Dump-to-Disk" (DTD)
CLEAR 8	Displays the MENU
CLEAR 9	Specifies send/receive file
CLEAR 0	Closes send/receive file
CLEAR :	Turns a device ON
CLEAR -	Turns OFF a device
CLEAR SHIFT !	Controls character "Echo"
CLEAR SHIFT #	Controls Echoing linefeeds
CLEAR SHIFT $	Controls linefeed acceptance
CLEAR SHIFT %	Restarts send/receive file
CLEAR SHIFT &	Appends data to end of file
CLEAR SHIFT '	Displays control characters being received/sent
CLEAR SHIFT (	Erases contents of screen
CLEAR SHIFT)	(Followed by ON command) Computer will use all 8 bits of character received
CLEAR SHIFT 0	Permits a TRSDOS library command to be entered
CLEAR SHIFT *	Controls data line handshaking

Once COMM is loaded, the screen displays the prompt "Use CLEAR 8 for menu." Any time you need to refresh your memory on the key associated with an action or device, you can call up this menu. Note the asterisks that appear on the menu. Asterisks above or below the Function Keys indicate that the function is active. Two asterisks indicate a device which is capable of both input and output.

```
        *                                                          * 13
DUPLX ECHO  ECOLF  ACCLF  REWIND PEOF  DCC   CLS  8-B  CMD  HNDSH  EXIT
=1=   =2=   =3=    =4=    =5=    =6=   =7=   =8=  =9=  =Ø=  =:=    =-=
*KI   *DO   *PR    *CL    *FS    *FR   DTD   ???  ID   RES  ON     OFF
 *     *           **                   *

FR-SPEC: HECTOR/HEC:Ø MEMORY: 47K
```

Rather than attempt to explain each of these Action Keys in detail, we will describe some typical operations and the commands required to execute them. Since we have a few tricks for simplifying the loading of COMM, we'll save our description of this procedure for last.

USING THE FUNCTION KEYS

After loading COMM and establishing communications with a Bulletin Board, you may find you have messages waiting and want to make a hardcopy. Before executing the required commands to "read" your mail, just turn your printer on. To do this press CLEAR 3 then CLEAR :. The information that appears on your screen will now also appear on hardcopy via your printer. (Providing of course, you've remembered to turn your printer on and its cable is properly connected.) To turn the printer function OFF, press CLEAR 3 then CLEAR -.

If you have a program you want to send via COMM, it must first be stored in an ASCII file on disk in either drive 0 or 1. After loading COMM, identify the file you want to send by pressing CLEAR 5, then CLEAR 9 and enter the name of the file at the prompt. Turn on the handshake mode by pressing CLEAR SHIFT * followed by ENTER. (You will need to determine whether the host supports handshaking. If it does not, try to transfer the file without the handshake mode. If this doesn't work, contact the host and find out what procedures are required to send files.) Open the file at the host end and ready it for receiving information by whatever command process the host requires. Press CLEAR 5, CLEAR : to turn on your file. When the transmission is complete, turn OFF the handshake mode by pressing CLEAR SHIFT * then CLEAR -. Close the file at the host end by whatever command process the host accepts. Then close your send file by pressing CLEAR 5 then CLEAR 0. This turns off the "Data Send" Device (*FS) and closes the file.

The procedure for receiving a file is similar. Use the commands established by the host to call up the file you want to download. Identify your receive file by pressing CLEAR 6, CLEAR 9 and then entering the filename in response to the prompt. CLEAR 6 followed by CLEAR : will open the receive area of memory. When you press ENTER the file will begin listing. To write the file to disk as it is being received press CLEAR 7 followed by CLEAR :. If you are running your RS-232 port above

300 baud, you should wait until you receive the entire file before turning DTD ON. When the listing is complete, press CLEAR 6 followed by CLEAR - to turn OFF the "Data Receive" Device (*FR) and if you have not already done so, press CLEAR 7 followed by CLEAR : to write the file to disk. After disk write has been completed, press CLEAR 6 then CLEAR 0 to turn off DTD and to close the receive file.

STARTUP PROCEDURE

And now for the startup procedure. This procedure is fairly simple but by using some additional TRSDOS 6.0 utilities, you can make it even simpler. The additional utilities can be used to establish control keys with which you can initialize COMM or you can setup an AUTO command that will load COMM each time the disk is booted. If you plan to use the COMM utility frequently, we think you will find creating the AUTO command, and thereby setting aside one disk as your communications disk, will save you time and frustration.

First the utility COM/DVR must be set to the communications line device (*CL). (Non U.S. residents are referred to the manual for special instructions.) This is accomplished easily enough. At TRSDOS ready, simply type:

SET *CL TO COM/DVR

After a little whirling of the disk drives, you will see the message "COM driver is now resident."

BUILDING A KSM FILE

At this point the instructions in the manual would have you type in SETCOM and its parameters and then COMM with its devspec and parameters. Instead, we opted to use BUILD to write a KSM file. KSM stands for Key Stroke Multiply. We will use the KSM/FLT (Key Stroke Multiply Filter) to assign the SETCOM and COMM command lines to a single key each so that we will only have to press CLEAR and the assigned key to execute the commands.

At TRSDOS Ready, type:

BUILD SYSTEM/KSM

The screen will display:

Building SYSTEM/KSM:0

A = 〉

The cursor will appear beside the A = 〉. Type the following:

SETCOM (BAUD = 300,WORD = 7,STOP = 1, PARITY = ON);

Be very sure you include the semicolon at the end. The semicolon will cause an ENTER to be executed at the end of the line. (The SETCOM parameters are our choice, the ones we use most often. You will want to select the parameters appropriate to your needs. For example, if you want to establish an AUTO load but access a bulletin board with unusual parameters, you may want to setup two disks. One for communicating with the bulletin board and one which contains the SETCOM parameters you use with other systems.) After typing the semicolon, press ENTER and you will see the prompt:

B = 〉

Type the following:

COMM *CL(XLATES = X'aabb',XLATER = X'aabb', XON = X'cc',XOFF = X'cc',NULL = OFF);

Remember to include the semicolon at the end of the line before pressing ENTER.

XLATES and XLATER are used to translate characters sent or received that are not available on your keyboard. For example, if you are using COMM as a terminal to communicate with another computer and you want to print a right bracket (]), you can use the XLATES parameter to produce a (]) by entering another character. If you were to type XLATES = X'025D' in your parameters, when you pressed CTRL B (hexadecimal 02), your computer would send the code for a right bracket (hexadecimal 5D) to the other computer. Characters you receive from another computer can be translated to a different symbol using XLATER. (Your Model 4 Owners Manual should have a list of characters and their hex values in the Appendix.)

That's your KSM file. To exit simply press SHIFT CTRL @ simultaneously. Now, in order to get your computer to read this file, you must make a connection between it and KSM/FLT. To do this type:

SET *KS KSM/FLT USING SYSTEM/KSM

Then:

FILTER *KI TO *KS

You can verify that your KSM file has been established by pressing CLEAR A. The SETCOM command you entered at the A = 〉 prompt should appear on the screen and be implemented. If this does not occur, you may need to deactivate the KSM filter before a new connection can be made. To deactivate the KSM filter and change to your new KSM file, type:

RESET *KI

RESET *KS

Then retype the "SET *KS . . . " and "FILTER . . . " commands listed above. Once you have verified that your CLEAR A and CLEAR B commands work, type:

SYSGEN (YES)

With this command you will create a file of current device and driver configurations. Each time you reset your computer, the configuration just established will be loaded into memory. This will insure that the KSM/FLT will continue to read your SYSTEM/KSM file. Any time you want to remove the configuration, type SYSGEN (NO).

ESTABLISHING AUTO LOAD

At this point you could load COMM by typing CLEAR A and CLEAR B at TRSDOS Ready. However, only two more steps are needed to set up an AUTO load. First, build a Job Control File. Type:

BUILD SYSTEM/JCL

The computer will respond with:

Building SYSTEM/JCL

Press CLEAR A and your SETCOM command line is entered as the first line of your Job Control File. Press CLEAR B and the COMM command appears as the second line. Finish this file by typing:

//STOP

Press ENTER and exit using SHIFT CTRL @. COMM can now be loaded each time the disk is booted by typing "DO SYSTEM JCL"—but why stop here? To complete the final step and establish an AUTO load type:

AUTO *DO SYSTEM/JCL

The COMM utility which comes with the TRSDOS 6.0 Operating System for the Model 4 is now resident. Each time you boot your disk, the COMM utility will be automatically loaded. If you plan ahead properly, have the files you want to send set up in ASCII files and have your printer primed and ready, your communications activities should be a breeze. Of course, we have only given you a sample of COMM's capabilities. Read about it in your manual and see just how versatile it can be. We think you'll find this utility a very welcome bonus with TRSDOS 6.0.

Oklahoma Reports and Current Bar Charts Available

LIVESTOCK AND GRAIN ANALYSIS AVAILABLE FROM OKLAHOMA STATE

Livestock and grain analysis is now being offered by the Department of Agricultural Economics at Oklahoma State University on AgriStar. Because three-quarters of Oklahoma's agricultural income is generated by these two commodities, agricultural economists have made a commitment to provide market analysis and outlook for cattle and grains.

Short-term analyses of cattle and wheat markets are prepared each Friday, and report the major factors affecting market trends for the past week and projections of prices for the coming week.

Oklahoma reports follow USDA livestock and grain reports, and provide analyses of monthly Cattle on Feed, biannual Cattle Inventory, Hogs and Pigs reports and grain reports. Oklahoma analyses are prepared on the date of the governmental report release.

John Ikerd, extension economist at Oklahoma, said that producers want to look at a variety of sources when making marketing decisions, and that Oklahoma's reports provide "information as good as a producer can get anywhere."

To access Oklahoma information, enter OKLAHOMA.UNIVERSITY or OKLAHOMA.STATE at any AgriScan for a menu of reports. There has been an Information Service Charge for these reports since November 15, 1983.

CURRENT BAR CHARTS

For your convenience, we have listed the most recently added bar charts and their codes. Because new bar charts become available when contract months begin trading, this list can be used as a handy access guide.

If the contract is new, there won't be a historical bar chart for it. However, if you do try to access a historical bar chart and it is not yet available, there will not be an Information Service charge.

AUG84	LIVE CATTLE	TFC428
OCT84	HOGS	TFC429
DEC84	HOGS	TFC430
SEP84	SOYBEANS	TFC431
NOV84	SOYBEANS	TFC432
SEP84	SOYMEAL	TFC433
OCT84	SOYMEAL	TFC434
DEC84	SOYMEAL	TFC435
SEP84	WHEAT	TFC436
DEC84	WHEAT	TFC492
SEP84	CORN	TFC437
DEC84	CORN	TFC438
SEP85	T-BILLS	TFC484
AUG84	FEEDER CATTLE	TFC480
MAR85	COTTON	TFC488

AgriStar is available at Radio Shack Computer Stores. Interested farmers may stop by their local Radio Shack Computer Center for a free demonstration.

Inside the PC-2

Don Durham
2572 Don Krag Cir. W.
Memphis, TN 38106

This one is for all you hackers who just can't stand not knowing what makes your computer tick. Having bought my first home computer back when you had to know what made it tick or it didn't tick at all, I developed a burning curiosity about the innards of the little beasts. What a frustrating feeling when I got my PC-2 and was only able to turn it on and program it in BASIC! Fortunately, it did have the old standby PEEK and POKE commands.

Armed with this tool, I did some peeking around in memory and discovered the BASIC command table, how a BASIC program is put together, and a few other minor things. It was not until Bruce Elliott's excellent articles on PC-2 Assembly Language, however, that I began to see what a marvel this little jewel really is.

It didn't take but a couple of hours till hand disassembling some of the routines in ROM got to be a real drag, and so I set out to write a disassembler. Like the saying goes, "It ain't elegant, but it works." Basically, the program is a table lookup. It reads an op-code and indexes into a table of numbers which represents how the particular instruction is formatted. Based on this number, it gets the mnemonic from another table, picks up any remaining bytes the instruction needs, reads the register name from the DATA statements, and builds the instruction string. At the same time, it is printing the ASCII equivalent of the bytes obtained from (where else?) another table. At first I was calculating these as I went, but that was too slow.

As the program is configured now, it will print one instruction about every 2 to 3 seconds. Not really lightning fast, but it sure beats doing it by hand. The program requires at least a 4K add-on memory and, of course, the printer/cassette interface.

There are a couple of things to note about the format of the instructions as they are printed. The vectored calls, VEJ, VMJ, etc., which are one and two byte instructions, pick up their real destination addresses from page FF in memory.I have formatted these so that the real address is printed rather than the page FF reference. The relative branch instructions have been treated in a similar manner, so that their real destination addresses are printed. I retained the " + " or "−" so that you immediately see whether the reference is forward or backward. This probably wouldn't conform to the input format for an assembler, but it is handy.

For those areas of memory which are data storage or work areas, there is also a section of the program which gives a dump of memory in both hex and ASCII. More about that later.

Getting the program and its data tables together is a little tricky, but once it's done, it's better than having all the data in DATA statements and reading the arrays each time the program is run. The main program is shown in Listing 1—DISASSEMBLER. It should be typed in first and saved on a new tape. After saving it and verifying that the save is good (CLOAD?), type NEW and enter Listing 2—TABLEBUILD (to build the tables). Position the original program tape to the point where the saved program ends then run this program with the recorder set to RECORD mode, and the computer remote control on. The tables will be saved and then when the DISASSEMBLER program is run, the tables can be read by just pressing ENTER when the message READ TABLES is displayed. Once the program and tables have been loaded into memory, using the DEF D sequence to start the program will leave all the tables intact and they will not have to be reloaded. Of course, the RUN or NEW commands will wipe out the tables.

By the way, you did save the TABLEBUILD program on a separate cassette in case you should ever blow the tables, didn't you?

When the program comes up, it checks the variable R1. If R1 is equal to 1, then the tables are probably intact, since the RUN and NEW commands would set it to zero. If the tables are not in memory, you will be prompted to read them in. If all is well, you will be asked for a beginning address. Input addresses may be specified in hex if they are preceded by an ampersand (&). You will then be asked for an ending address. Next you'll be asked for the color you want the listing in (varying the color helps even the replacement time of the pens). After entering the color, you may select 1 for DISASSEMBLY or 2 for DUMP. Now you can sit back and relax while your electronic wizard reveals all.

Starting Address 65500
Ending Address 65510

1. DISASM

```
FFDC  DE                  VEJ  D6DF
FFDD  BC * UNKNOWN OPCODE *
FFDE  D6                  VEJ  DED1
FFDF  DF                  DEC  A
FFEØ  CD 8B               VMJ  5BEE
FFE2  C4                  VEJ  DCD5
FFE3  ØØ                  SBC  XL
FFE4  CD 89               VMJ  F6ED
FFE6  F7                  CIN
```

2. DUMP

```
FFDC DEBC D6DF CD8B C4ØØ -------
FFE4 CD89 F7ØD F661 F79C -----a-
```

Starting Address 63500
Ending Address 63510

1. DISASM

```
F8ØC  FF * UNKNOWN OPCODE *
F8ØD  FB                  SEC
F8ØE  12                  ADC  YL
F8ØF  1A                  STA  YL
F81Ø  6A Ø6               LDI  UL,Ø6
F812  14                  LDA  YL
F813  B9 Ø7               ANI  A,Ø7
F815  8B Ø1               BZS  +,F818
```

2. DUMP

```
F8ØC FFFB 121A 6AØ6 14B9 ----j--
F814 Ø78B Ø157 4388 Ø99A ---WC--
```

You know, it's getting to be a real drag writing Assembler Language programs and entering the hex code by hand. Anybody out there working on an assembler in BASIC?

LISTING 1—DISASSEMBLER

```
1Ø "D"
     : IF R1 = 1 GOTO "RDY"
2Ø CLEAR
     : DIM MN$(81) * 3,CD(255),FD(255),HX$(255) * 2
3Ø WAIT
     : PRINT "READ TABLES"
     : WAIT Ø
4Ø INPUT #"DISTBL";MN$(*),CD(*),FD(*),HX$(*)
     : R1 = 1
     : GOTO "RDY"
5Ø "TWOHX"
     : T$ = ""
     : V = INT (H/&1ØØ)
     : GOSUB "ONEHX"
     : V = H - (V * &1ØØ)
6Ø "ONEHX"
     : T$ = T$ + HX$(V)
     : RETURN
7Ø "NORM"
     : RESTORE
     : T$ = ""
     : V = B
     : GOSUB "ONEHX"
     : LPRINT T$ + " ";
     : A = A + 1
     : C = CD(B)
8Ø IF C = Ø LPRINT "* UNKNOWN OPCODE *"
     : RETURN
9Ø X = C AND &7F
     : B$ = MN$(X) + " "
     : C = INT (C/2^7)
     : IF C = Ø TAB 21
     : LPRINT B$
     : RETURN
1ØØ IF MN$(X) <> "VEJ" GOTO 12Ø
11Ø V = &FFØØ + B
     : H = PEEK (V) * 256 + PEEK (V + 1)
     : GOSUB "TWOHX"
     : TAB 21
     : LPRINT B$ + T$
     : RETURN
12Ø IF LEFT$ (MN$(X),1) <> "V" GOTO 14Ø
13Ø B = PEEK (A)
     : T$ = ""
     : V = B
     : GOSUB "ONEHX"
     : LPRINT T$;
     : A = A + 1
     : GOTO 11Ø
14Ø X = C AND &1F
     : IF X = Ø LET R$ = ""
     : GOTO "R2"
15Ø FOR I = 1 TO X
     : READ R$
     : NEXT I
     : IF D = 1 LET B$ = B$ + "#"
     : D = Ø
16Ø IF R$ <> "XX" GOTO 2ØØ
17Ø H = (PEEK (A) * 256) + PEEK (A + 1)
     : GOSUB "TWOHX"
     : R$ = "(" + T$ + ")"
18Ø GOSUB 19Ø
     : GOTO 2ØØ
19Ø FOR J = 1 TO 2
     : V = PEEK (A)
     : T$ = ""
```

```
      : GOSUB "ONEHX"
      : LPRINT T$ + " ";
      : A = A + 1
      : NEXT J
      : RETURN
2ØØ B$ = B$ + R$
21Ø "R2"
      : X = INT (C/2^5)
      : IF X = Ø TAB 21
      : LPRINT B$
      : RETURN
22Ø IF R$ <> "" LET B$ = B$ + ","
23Ø IF X <> 1 GOTO 31Ø
24Ø V = PEEK (A)
      : T$ = ""
      : GOSUB "ONEHX"
      : LPRINT T$;
25Ø IF R$ = "+" LET H = A + PEEK (A) + 1
      : GOTO 28Ø
26Ø IF R$ <> "-" GOTO 29Ø
27Ø H = A - PEEK (A) + 1
28Ø GOSUB "TWOHX"
      : B$ = B$ + T$
      : GOTO 3ØØ
29Ø V = PEEK (A)
      : T$ = ""
      : GOSUB "ONEHX"
      : B$ = B$ + T$
3ØØ TAB 21
      : LPRINT B$
      : A = A + 1
      : RETURN
31Ø IF X <> 3 GOTO 33Ø
32Ø B$ = B$ + " " + T$
      : TAB 21
      : LPRINT B$
      : RETURN
33Ø H = (PEEK (A) * 256) + PEEK (A + 1)
      : GOSUB "TWOHX"
      : B$ = B$ + T$
34Ø GOSUB 19Ø
      : A = A - 1
      : GOTO 3ØØ
35Ø "RDY"
      : CLS
      : WAIT Ø
      : RESTORE
36Ø CLS
      : INPUT "START ADDRESS ";A
37Ø INPUT "END ADDRESS ";E
      : IF E < A GOTO 36Ø
38Ø INPUT "COLOR (Ø-3) ";C
      : IF C < Ø OR C > 3 GOTO 38Ø
39Ø COLOR C
      : CSIZE 1
      : TAB Ø
      : LF 2
4ØØ INPUT "DISASM 1. DUMP 2. ";C
      : IF C = 2 GOTO "DUMP"
41Ø "DIS"
      : D = Ø
      : IF A > E GOTO "RDY"
42Ø R$ = ""
43Ø H = A
      : GOSUB "TWOHX"
      : LPRINT T$;
      : TAB 5
44Ø B = PEEK (A)
      : IF B = &FD GOTO "FD"
45Ø GOSUB "NORM"
      : GOTO "DIS"
46Ø "FD"
      : LPRINT "FD ";
      : D = 1
      : A = A + 1
      : T$ = ""
47Ø B = PEEK (A)
      : C = FD(B)
      : IF C = 1 GOTO 44Ø
48Ø V = B
      : GOSUB "ONEHX"
      : LPRINT T$;" ";
49Ø IF C = Ø LPRINT "* UNKNOWN OPCODE *"
      : A = A + 1
      : GOTO "DIS"
5ØØ R = INT (C/1ØØ)
      : M = C - R * 1ØØ
      : B$ = MN$(M)
51Ø RESTORE "FREG"
      : FOR I = 1 TO R
      : READ R$
      : NEXT I
52Ø TAB 21
      : LPRINT B$;" ";R$
      : A = A + 1
      : GOTO "DIS"
53Ø "DUMP"
      : H = A
      : GOSUB "TWOHX"
      : LPRINT T$ + " ";
54Ø FOR K = 1 TO 4
55Ø FOR J = 1 TO 2
      : V = PEEK (A)
      : T$ = ""
      : GOSUB "ONEHX"
      : LPRINT T$;
      : A = A + 1
      : NEXT J
56Ø LPRINT " ";
      : NEXT K
57Ø TAB 25
      : FOR J = 8 TO 1 STEP -1
58Ø V = PEEK (A - J)
      : IF V > 31 AND V < 128 LPRINT CHR$ (V);
      : GOTO 6ØØ
59Ø LPRINT "-";
6ØØ NEXT J
61Ø LPRINT
      : IF A >= E GOTO "RDY"
62Ø GOTO "DUMP"
63Ø DATA "A","XL","YL","UL","XH","YH","UH","(X)",
      "(Y)","(U)","XX","X","Y","U"
64Ø DATA "S","P","+","-","V"
65Ø "FREG" : DATA "A","XH","YH","UH","X","Y","U","S",
      "P"," "
```

LISTING 2—TABLEBUILD

```
1Ø CLEAR
2Ø DIM MN$(81) * 3,CD(255),FD(255),HX$(255) * 2
3Ø H$ = "Ø123456789ABCDEF"
4Ø WAIT Ø
5Ø PRINT "BUILDING HEX TABLE"
6Ø FOR I = Ø TO 255
7Ø HX$(I) = MID$ (H$,(I AND &FØ)/16 + 1,1) + MID$
     (H$,(I AND &F) + 1,1)
8Ø NEXT I
9Ø CLS
1ØØ PRINT "BUILDING MNEMONIC TABLE"
11Ø FOR I = Ø TO 81
12Ø READ MN$(I)
13Ø NEXT I
14Ø CLS
15Ø PRINT "BUILDING CD ARRAY"
16Ø FOR I = Ø TO 255
17Ø READ CD(I)
18Ø NEXT I
19Ø CLS
2ØØ PRINT "BUILDING FD ARRAY"
21Ø FOR I = Ø TO 255
22Ø READ FD(I)
23Ø NEXT I
```

```
240 PRINT #"DISTBL";MN$(*),CD(*),FD(*),HX$(*)
     : END
250 DATA "ADC","ADI","ADR","AND","ANI","DCA"
260 DATA "DCS","DEC","EAI","EOR","INC","ORA"
270 DATA "ORI","SBC","SBI","BII","BIT","CPA"
280 DATA "CPI","ATT","LDA","LDE","LDI","LDX"
290 DATA "LIN","POP","PSH","SDE","SIN","STA"
300 DATA "STX","TTA","AEX","CIN","DRL","DRR"
310 DATA "ROL","ROR","SHL","SHR","TIN","AMØ"
320 DATA "AM1","ATP","CDV","HLT","ITA","NOP"
330 DATA "OFF","RDP","REC","RIE","RPU","RPV"
340 DATA "SDP","SEC","SIE","SPU","SPV","BCH"
350 DATA "BCR","BCS","BHR","BHS","BVR","BVS"
360 DATA "BZR","BZS","JMP","LOP","SJP","VCR"
370 DATA "VCS","VEJ","VHR","VHS","VMJ","VVS"
380 DATA "VZR","VZS","RTI","RTN"
390 DATA 269,1Ø37,256,1Ø24,276,1Ø44,273,1Ø41,669
4ØØ DATA 1Ø27,285,1Ø35,1Ø3Ø,1Ø33,1Ø53,1Ø4Ø,397
41Ø DATA 1165,384,1152,4Ø4,1172,4Ø1,1169,797,1155
42Ø DATA 413,1163,1158,1161,1181,1168,525,1293
43Ø DATA 512,128Ø,532,13ØØ,529,1297,925,1283,541
44Ø DATA 1291,1286,1289,13Ø9,1296,Ø,Ø,Ø,Ø,Ø,Ø,Ø
45Ø DATA Ø,47,Ø,Ø,Ø,Ø,Ø,Ø,Ø,266,1564,263,1563
46Ø DATA 1546,156Ø,1543,1557,4758,5124,4374,5132
47Ø DATA 4754,5135,437Ø,5121,394,1692,391,1691
48Ø DATA 1674,1688,1671,1685,4886,5252,45Ø2,526Ø
49Ø DATA 4882,5263,4498,5249,522,182Ø,519,1819
5ØØ DATA 18Ø2,1816,1799,1813,5Ø14,538Ø,463Ø,5388
51Ø DATA 5Ø1Ø,5391,4626,5377,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
52Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,653,6332,64Ø,6333,66Ø,6334
53Ø DATA 657,6367,4677,6338,8Ø,6339,1Ø29,6336
54Ø DATA 6331,6337,781,646Ø,768,6461,788,6462
55Ø DATA 785,6463,Ø,6466,81,6467,1157,6464,6459
56Ø DATA 6465,9Ø9,1421,896,14Ø8,916,1428,913,1425
57Ø DATA 58,1411,1Ø134,1419,1285,1417,1437,1424
58Ø DATA Ø,4238,Ø,4225,Ø,4246,Ø,4242,53,4228,826Ø
59Ø DATA 4236,Ø,41Ø4,8262,4239,12361,4167,12361
6ØØ DATA 4168,12361,417Ø,12361,4171,12361,4174
61Ø DATA 12361,4175,12361,4172,12361,4173,12361
62Ø DATA 37,12361,1Ø59,12361,39,12361,1Ø58,12361
63Ø DATA 38,12361,36,12361,138,12361,135,12361
64Ø DATA 57,12361,52,12361,Ø,12361,Ø,12361,55Ø8
65Ø DATA 12361,5516,12361,5519,12361,55Ø5,12361
66Ø DATA 32,12361,Ø,12361,4Ø,12361,33,Ø,5Ø,Ø,55
67Ø DATA Ø,Ø,Ø,Ø
68Ø DATA Ø,1,Ø,1,Ø,1,Ø,1,623,1,525,1,1,1,1,1,Ø
69Ø DATA 1,Ø,1,Ø,1,Ø,1,623,1,625,1,1,1,1,1,Ø
7ØØ DATA 1,Ø,1,Ø,1,Ø,1,3,1,725,1,1,1,1,1,Ø,Ø,Ø
71Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,21Ø,Ø,2Ø7,Ø
72Ø DATA Ø,Ø,Ø,Ø,823,1,53Ø,1,1Ø48,1,83Ø,1,31Ø,Ø
73Ø DATA 3Ø7,Ø,Ø,Ø,Ø,Ø,923,1,623,1,Ø,1,93Ø,1
74Ø DATA 41Ø,Ø,4Ø7,Ø,Ø,Ø,Ø,Ø,Ø,1,73Ø,1,Ø,1,Ø,1,Ø
75Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,1Ø56,Ø
76Ø DATA Ø,Ø,Ø,Ø,Ø,526,Ø,125,Ø,1,Ø,1Ø44,Ø,Ø,Ø,Ø
77Ø DATA Ø,Ø,Ø,Ø,Ø,626,Ø,Ø,Ø,1,Ø,Ø,Ø,Ø,1,Ø,1,Ø,1
78Ø DATA Ø,1,726,1,1Ø31,1,1,1,1,1,Ø,1Ø45,Ø,Ø,Ø,Ø
79Ø DATA Ø,Ø,Ø,Ø,1Ø46,Ø,Ø,Ø,1Ø51,Ø,1Ø49,1Ø54
8ØØ DATA Ø,Ø,Ø,Ø,Ø,Ø,126,Ø,5Ø2,Ø,1Ø43,Ø,1Ø41,Ø,Ø
81Ø DATA Ø,Ø,1,Ø,Ø,Ø,1,Ø,Ø,6Ø2,Ø,Ø,Ø,1Ø42,Ø,Ø,Ø
82Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,1,7Ø2,1,1Ø19,1,Ø,1,Ø,Ø,Ø
83Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
```

PC-2 Renumber

R. R. Jueneman
1522 Hardwood Lane
McLean, VA 22102

The February 1983 issue of TRS-80 Microcomputer News contained a program to renumber BASIC lines for the PC-2. Unfortunately, when used with the 8K RAM chip, the starting address is incorrect, and the program POKES garbage throughout memory. An ALL RESET which destroys the program is required to recover.

I am enclosing a safer program which uses STATUS 2 to determine the end of the program and STATUS 2 minus STATUS 1 to find the beginning. Because the FOR loop is all one statement, it will safely renumber itself. The third argument of POKE demonstrates an occasionally useful trick—the value of I is ANDed with hexadecimal &FF to give the result modulo 256. This is much simpler than the division, INT, multiplication, and subtraction that would otherwise be required.

The line numbers are arbitrary. The program is executed by pressing "DEF" then "N" in the RUN mode.

One final caution, the program merely renumbers the lines of your program. It does NOT correct any statements that refer to absolute line numbers, i.e., in GOTO's, GOSUBs, or RESTORE's. But absolute line numbers are a deplorable programming practice, particularly since the PC-2 allows nice long statement labels.

```
54Ø "N"
     : REM RENUMBER LINES
55Ø J=STATUS 2-STATUS 1
     : REM Start of program area
56Ø FOR I=1Ø to 3275Ø STEP 1Ø
     : POKE J,INT (I/256),I AND &FF
565 J=J + 3 + PEEK (J+2)
     : IF J< (STATUS 2)-2 NEXT I
57Ø BEEP 3
     : END
```

HEARTS (From page 11)

```
18Ø IF A1 > 57 THEN 13Ø
19Ø GOTO 15Ø
2ØØ DATA  57,3,2,3,2,4,2,3,2,4,2,3,2,4,2,3,2,4,2,3,2,
     3,2,4,1,4,2,4,1,4,2,4,1,4,2,4,1,4,2,4,1,4,2
21Ø DATA  2,9,2,9,2,9,2,9,2,9,2,3,7,4,7,4,7,4,7,4,7,3
     ,4,5,6,5,6,5,6,5,6,5,4
22Ø DATA 5,3,8,3,8,3,8,3,8,3,5,6,1,1Ø,1,1Ø,1,1Ø,1,1Ø,
     1,6,57
23Ø DATA 3,2,3,2,3,31,3,2,3,2,3,2,4,1,4,2,31,2,4,1,4,
     2,2,9,2,6,5,9,5,6,2,9,2,3,7,3,5,7,7,7,5,3,7,3
24Ø DATA 4,5,4,4,9,5,9,4,4,5,4,5,3,5,3,11,3,11,3,5,3,
     5,6,1,6,3,12,1,12,3,6,1,6,13,3,25,3,13
25Ø DATA 3,2,3,2,3,3,25,3,3,2,3,2,3,2,4,1,4,2,4,23,4,
     2,4,1,4,2,2,9,2,5,21,5,2,9,2,3,7,3,6,19,6,3,7,3
26Ø DATA 4,5,4,7,17,7,4,5,4,5,3,5,8,15,8,5,3,5,6,1,6,
     9,13,9,6,1,6,13,1Ø,11,1Ø,13
27Ø DATA 3,2,3,2,3,11,9,11,3,2,3,2,3,2,4,1,4,2,12,7,
     12,2,4,1,4,2,2,9,2,13,5,13,2,9,2,3,7,3,14,3,14,3,
     7,3
28Ø DATA 4,5,4,15,1,15,4,5,4,5,3,5,31,5,3,5,6,1,6,31,
     6,1,6
29Ø DATA 57,3,2,3,2,4,2,3,2,4,2,3,2,4,2,3,2,4,2,3,2,3
     ,2,4,1,4,2,4,1,4,2,4,1,4,2,4,1,4,2,4,1,4,2
3ØØ DATA 2,9,2,9,2,9,2,9,2,9,2,3,7,4,7,4,7,4,7,4,7,3,
     4,5,6,5,6,5,6,5,6,5,4
31Ø DATA 5,3,8,3,8,3,8,3,8,3,5,6,1,1Ø,1,1Ø,1,1Ø,1,1Ø,
     1,6,57
32Ø END
```

PC-2 Assembly Language—Part 6

By Bruce Elliott

This is the sixth in a series of articles which describe the MPU (microprocessor unit) used in the Radio Shack PC-2 pocket computer. It is our intention to include specific information about the 8-bit CMOS microprocessor, the machine code used by the microprocessor, as well as information about the PC-2 memory map, and certain ROM calls which are available. Please realize that much of what we are talking about refers to the overall capabilities of the MPU, and does not imply that all of these things can be done with a PC-2.

The information provided in these articles is the only information which is available. We will try to clarify any ambiguities which occur in the articles, but can not reply to questions outside the scope of these articles. Further, published copies of *TRS-80 Microcomputer News* are the only source of this information, and we will not be maintaining back issues.

In this article we want to present information on some of the PC-2 ROM calls which are available.

When you are going to use a ROM call, there are four items which you want to be concerned with:

1. Entry Address
2. Entry Conditions
3. Exit Conditions
4. Flags

The Entry Address is the address you use in the CALL statement from BASIC or a SJP call from machine language.

The Entry Conditions are conditions you must fulfill if the routine is to function properly. Normally, entry conditions specify where information must be and what information you must put in the MPU registers for the routine to function properly.

The Exit Conditions tell you where you will find the result of the operation (if there is a result) or provide you with other information about how things will change as a result of using a particular ROM call.

If a ROM call makes particular changes to any of the machine's flags, this information will be noted so you can properly interpret the results you get.

A CAUTION

I have not had time to test the information which is provided below on ROM calls. The information provided is as accurate as I could make it from the materials I am working with. Test any ROM call for proper operation BEFORE you use it in a program. Remember that the 'H' following a numeral indicates hexadecimal notation.

CURSOR INFORMATION

The PC-2 cursor pointer is located at 7875H. This location is used by the PC-2 to keep track of where the cursor should be. If you are working exclusively in machine language, updating 7875H is all that is needed for cursor location.

If you are working from BASIC, and wish to update the cursor location directly using POKEs or CALLs, you must also set bit 0 of location 7874H. Setting this bit from machine language can be accomplished by:

```
ORI 7874H, 01H
```

This operation is done automatically when you use the CURSOR or GCURSOR BASIC commands.

If you execute a ROM call which resets the cursor pointer and are going to return to BASIC, you must set bit 0 of location 7874H as described above.

If you wish to reset the cursor from machine language, you can use the following code:

```
ANI 7874H, 0FEH
ANI 7875H, 00H
```

To increment the cursor pointer, use the following:
If you are displaying characters:

```
(7875H) = (7875H) + 06H
```

If you are displaying graphics:

```
(7875H) = (7875H) + 01H
```

Note: (7875H) must be between 00H and 9BH.

SYSTEM CALLS FOR THE LCD DISPLAY

Output one character to the LCD

1. System call address: ED57H
2. Entry Conditions:
   a. The ASCII character code for the character to be displayed must be in the ACC (Accumulator) before making the call.
   b. The location where the character will be placed is determined by the content of the cursor pointer.
3. Exit Conditions: The cursor pointer does not change.
4. Flags: Carry = 0 The cursor stays between 00H and 95H on call.
   = 1 The cursor stays in 96H on the call.

Output one character to the LCD and increment the cursor position by one character (6H).

1. System call address: ED4DH
2. Entry Conditions: The ASCII character code for the character to be displayed must be in the ACC (Accumulator) before making the call.
3. Exit Conditions: If the cursor position before the call was in the range 00H to 95H, then the new cursor position equals the old position plus 6H. If the cursor position before the call was 96H or larger, then the new cursor position is set equal to zero.
4. Flags:

Outputting n characters to the LCD.

1. System call address: ED00H
2. Entry Conditions:
   a. The 16 bit starting address for the string to be displayed is placed in the U register (0000H ⟨ = U ⟨ = FFFFH).

b. The length of the character string is placed in the Accumulator (01H ⟨ = ACC ⟨ = 1AH).
c. The cursor pointer indicates where on the LCD the computer is to begin displaying the string.
3. Exit Conditions: The cursor pointer is updated.
4. Flags: Carry = 0 The cursor position is set to the right-most end of the displayed character string on the LCD.
= 1 The specified character string ended in the 26th LCD column, or the string was too long to be displayed within 26 columns. The cursor will be steady, indicating the last character displayed.

The number of characters specified in the accumulator is output from consecutive addresses beginning with the address specified in the U register. The characters will be placed on the LCD beginning with the position indicated by the cursor pointer. The cursor pointer can be set from machine language, or by using the BASIC CURSOR or GCURSOR commands. If the information to be displayed exceeds the 156th dot on the LCD, the excess information will not be displayed.

Outputting n characters to the LCD beginning from character position 1.
1. System call address: ED3BH
2. Entry Conditions:
a. The 16 bit beginning address location of the string to be displayed is stored in the U register (0000H ⟨ = U ⟨ = FFFFH).
b. An 8 bit number indicating the length of the character string is stored in XL (The lower half of the X register. 01H ⟨ = XL ⟨ = 1AH).
3. Exit Conditions:
4. Flags: Carry = 0 The character string has been displayed in 25 or fewer columns.
= 1 The character string reached or exceeded the 26th column.

Transferring 1 byte of data (1 dot column of graphic information) to the current cursor position.
1. System call address: EDEFH
2. Entry Conditions: The byte representing the graphic pattern to be displayed is placed in the accumulator.
3. Exit Conditions:
a. The data is transferred to the current cursor position, which does not change.
b. The contents of ACC and the X and U registers may change.
c. The content of the Y register will not change.
4. Flags:

DATA CONVERSIONS

Converting two bytes of ASCII code (0 - 9, A - F only) into a one byte hexadecimal value.
1. System call address: ED95H
2. Entry Conditions: The X register should contain the address of the first of two consecutive bytes in memory which contain the ASCII characters.
3. Exit Conditions:
a. The X register will be incremented by 2
b. The U and Y registers will be unchanged
c. The ACC will contain the converted hex value.
4. Flags:

DISPLAY THROUGH A BUFFER

Data can be placed into an 80-byte buffer (7BB0H - 7BFFH) and then displayed as needed by specifying the proper cursor address in the buffer.
1. System call address: E8CAH
2. Entry Conditions:
a. Any character string which is placed in the buffer must have a 0DH code as the last character. This means that the longest allowable character string is 79 characters plus the 0DH end code.
b. The Y register holds the cursor pointer for the buffer. The documentation does not specify what value goes into Y. Since Y is 16 bits long, I presume that you would use the actual memory address within the buffer.
c. Address 7880H contains a parameter which determines how the contents of the buffer are to be displayed:

If the binary content of 7880H is 0100 0000, then the character string stored in the buffer is output to the LCD using the content of the Y register as the cursor pointer.

Note: If the number of characters in the buffer is 26 or less, then all of the characters are displayed on the LCD starting from the left side of the LCD. The cursor pointer (7875H) has no effect on this operation. If the number of characters in the buffer is greater than 26, the character in the address specified by the Y register and the PRECEDING 25 characters are displayed on the LCD starting at the left side of the LCD.

If the binary content of 7880H is 0000 0000, then the cursor pointer in the Y register is ignored and he first 26 characters stored in the buffer are output to the LCD.

If the binary content of 7880H is 0010 0000, then numeric data stored in memory addresses 7A00H - 7A07H are output to the LCD.

Note: See below for a discussion of the 7A00H - 7A07H buffer.

3. Exit Conditions:
4. Flags:

The 7A00H - 7A07H Buffer

The PC-2 documentation describes three possible sets of data for the 7A00H buffer:

Decimal Values:

A decimal value may fall into the range 9.999999999 x 10E99 = x = 9.999999999 x 10E99.

7A00H contains the exponent (negative exponents are expressed as complements: 03H = x 10E3, 1FH = x 10E31, and FFH = x 10E-1)

7A01H contains the sign of the mantissa (00H = +, 80H = −)

7A02H - 7A06H contains the mantissa.

7A07H contains 00H.

Examples

```
7A00H       7A07H
00H 00H 00H 00H 00H 00H 00H 00H = 0.0
00H 00H 12H 34H 50H 00H 00H 00H = 1.2345
FEH 00H 98H 76H 54H 32H 12H 00H = 0.9876543212
08H 80H 54H 32H 00H 00H 00H 00H = -5.432  x 10
```

Integer Values:
An integer value may fall into the range -32768 〈 = x 〈 = 32767.
7A00H—7A03H - Don't Care
7A04H—B2H
7A05H—7A06H Binary number in complements (e.g. 00H 00H = 0, FFH FBH = -5, 7FH FFH = 32767)
7A07H—Don't Care
Character Strings:
7A00H—7A03H—Don't Care
7A04H—D0H
7A05H—Upper two bytes of string address in memory
7A06H—Lower two bytes of string address in memory (string address can be in the range 0000H - FFFFH)
7A07H—Length of the string (range 01H - 50H)
Note: This last set of conditions (for strings) seems to imply that a string buffer can be anyplace in memory, rather than being restricted to 7BB0H - 7BFFH. Test this before relying on it.

CASSETTE I/O AND CONTROL

During tape I/O activities, the paper feed action of the printer is inhibited.

Turn Tape Drive On
1. System call address: BF11H
2. Entry Conditions: Memory address 7879H is used to specify certain conditions:
   Bit 7: 0 = CMT input port closes; select 0 for CMT input.
   1 = CMT input port opens; select 1 for CMT input.
   Bit 4: 0 = Remote 0
   1 = Remote 1
3. Exit Conditions:
4 Flags:

Turn Tape Drive Off

1. System call address: BF43H
2. Entry Conditions:
3. Exit Conditions: Remote drive 0 is turned off unconditionally. Remote drive 1 is turned off or on depending on bit 7 of an unspecified address (probably 7879H). If bit 7 is 0 the drive is OFF, and if bit 7 is a 1 then, the drive is ON. This bit can be set using the BASIC commands RMT ON and RMT OFF.
4. Flags:

Construct Tape Synchronization Header

The header, a 40-byte data set, consists of the synchronization header, a file name, file mode, and other data. This header is created inside the computer (addresses 7B60H - 7B87H) and output to tape.

1. System call address: BBD6H
2. Entry Conditions: The file mode (00 = Machine Object, 01 = Program, 02 = Reserve, 04 = Data) must be placed in the accumulator.
3. Exit Conditions:
   a. An 8 byte synchronization header will be in 7B60H - 7B67H
   b. File mode will be in 7B68H
   c. 00H characters will be placed in locations 7B69H - 7B87H.
4. Flags:

A program file name (16 or fewer characters) can be placed in memory locations 7B69H - 7B78H, if you wish. Address locations 7B79H - 7B87H may be used for your own purposes.

Output Tape Synchronization Header

1. System call address: BCE8H
2. Entry Conditions:
   a. Bit seven of address 7879H must be zero and bit four will be a zero for remote 0 and a one for remote 1.
   b. Whether the PC-2 will beep or not during cassette I/O is controlled by the BASIC commands BEEP ON and BEEP OFF, or by setting bit zero of 786BH.
3. Exit Conditions:
4. Flags:

Send a Character to Tape

1. System call address: BDCCH
2. Entry Conditions: Character to be output is placed in the Accumulator. The call to write the synchronization header must be used before outputting data using this system call.
3. Exit Conditions:
4. Flags:

Write a tape file

Files can be written by specifying the start address of the data and the number of bytes to be output.
1. System call address: BD3CH
2. Entry Conditions:
   a. The X register should contain the start address (0000H 〈 = X 〈 = FFFFH) for the file to be written.
   b. The U register should contain the number of bytes to be written minus one (0000H 〈 = U 〈 = FFFFH).
3. Exit Conditions: Check sum data is output at the rate of 2 bytes for each 80 bytes written. The number of check sum bytes is not included in the U register number of bytes to be output.
4. Flags: CARRY = 0 if Output ended normally
   = 1 if BREAK key was pressed

Read Tape Synchronization Header

Before the header can be read from tape, you must construct a header using the BBD6H call. This will specify the file type. If you are searching for a particular file, you may place the file name in address locations 7B69H - 7B78H. If you specify a file name, the tape will be searched for a matching name. If you do not specify a file name (file name = all 00H characters) then file names will be ignored during input.

1. System call address: BCE8H
2. Entry Conditions:
   a. build a header with file type
   b. specify a file name if you wish.
   c. Set 7879H: Bit Seven = 1
Bit Four = 0 for Remote 0
   = 1 for Remote 1

3. Exit Conditions:
   a. 7B91H - 7BA0H will contain the 16 character file name (padded with 00H characters if file name was less than 16 characters)
   b. 7BA1H - 7BAFH will contain whatever was in 7B79H - 7B87H when the file was written to tape.
4. Flags: Carry = 0 Reading finished
   = 1 BREAK key pressed

Read a Character from Tape

1. System call address: BDF0H
2. Entry Conditions:
3. Exit Conditions: The data value read from the tape is placed in the accumulator.
4. Flags: Carry = 0 Byte read properly
   = 1 BREAK key was pressed

Read a file from tape

1. System call address: BD3CH
2. Entry Conditions:
   a. The X register contains the first memory address (0000H ⟨ = X ⟨ = FFFFH) that the file is to be loaded into.
   b. The U register contains the number of bytes minus one (0000H ⟨ = U ⟨ = FFFFH) to be read from tape.
   c. Address 7879H bit seven contains zero
   bit six = 0 for data read
   = 1 for data verify
3. Exit Conditions:
   a. Check sum information is automatically checked during tape input.
   b. The X register contains the address of the last data byte plus one.
4. Flags: Carry = 0 if loading ended normally
   = 1 abnormal end, check H and V flags
   H = 1 if C = 1 then BREAK key pressed
   = 0 check V flag
   V = 1 if C = 1 and H = 0 then data in memory and the data from the tape did not verify properly.
   = 0 if C = 1 and H = 0 then a check sum error occurred.

Finishing Tape I/O Activities

When you are finished using tape I/O you should inform the system.
1. System call address: BBF5H
2. Entry Conditions: Bit seven of 7879H should be a zero to terminate data output or a one to terminate data input.
3. Exit Conditions:
   a. The serial port is reset
   b. Printer Paper Feed is enabled
   c. Cassette motor drives are turned off.
4. Flags:

BASIC Program Tapes

The PC-2 creates and reads tapes for BASIC program files using the file read and write routines described here. Before the synchronization header is written to tape, the PC-2 stores the length of the program (in bytes) minus one in locations 7B85H and 7B86H. This information is then recorded as part of the synchronization information for later use in reading the file. When the header information is read back during a synchronization header read, the length information is in 7BACH and 7BADH.

KEYBOARD INPUT CALLS

Scan Keyboard, wait for a key to be pressed

1. System call address: E243H
2. Entry Conditions:
3. Exit Conditions:
   a. Key code is in the accumulator
   b. (SHIFT), (DEF), and (SML) do not cause this routine to return.
   c. Auto power off will occur after about seven minutes if no key is pressed.
   d. If the BREAK key is entered, execute the following: ANI #F00BH, 0FDH (FDH E9H F0H 0BH FDH)
4. Flags: Carry 0 = Accumulator has key code
   1 = BREAK key, Accumulator = 0EH

Key Code Table

	0	1	2	3	4	5	6	7
0			SPACE	0	@	P		p
1	(SHIFT)	F1	!	1	A	Q	a	q
2	(SML)	F2	"	2	B	R	b	r
3		F3	#	3	C	S	c	s
4		F4	$	4	D	T	d	t
5		F5	%	5	E	U	e	u
6		F6	&	6	F	V	f	v
7				7	G	W	g	w
8	←	CL	(	8	H	X	h	x
9	⇕	RCL	)	9	I	Y	i	y
A	↓	CA	*	:	J	Z	j	z
B	↑	(DEF)	+	;	K	rad	k	
C	→	INS	,	<	L		l	
D	ENTER	DEL	-	=	M	π	m	
E	BREAK		.	>	N	∧	n	
F	OFF	MODE	/	?	O		o	

Scan keyboard and Return

1. System call address: E42CH
2. Entry Conditions:
3. Exit Conditions:
   a. If no key was pressed, accumulator = 00H
   b. If a key was pressed, Key code is in accumulator
4. Flags:

NUMERIC FUNCTION CALLS

From the documentation, it appears that numeric functions are called with the X register pointing to 7A00H - 7A07H and the Y register pointing to 7A10H - 7A17H if Y is needed. Results appear to always be stored in 7A00H - 7A07H. Numeric data is stored in these memory areas as previously described.

Two Variable Numeric Functions

Addition	X + Y→X	EFBAH
Subtraction	X - Y→X	EFB6H
Multiplication	X * Y→X	F01AH
Division	X / Y→X	F084H
Exponentiation	X∧Y→X	F89CH

Single Variable Numeric Function

Square Root	SQR X→X	F0E9H
Logarithm	LN X→X	F161H
	LOG X→X	F165H
Exponentials	EXP X→X	F1CBH
	10∧X→X	F1D4H
Sine	SIN X→X	F3A2H
Cosine	COS X→X	F391H
Tangent	TAN X→X	F39EH
Arcsine	ASN X→X	F49AH
Arccosine	ACS X→X	F492H
Arctangent	ATN X→X	F496H
	DEG X→X	F531H
	DMS X→X	F564H
Absolute Value	ABS X→X	F597H
Signum Function	SGN X→X	F59DH
Integer Function	INT X→X	F5BEH

OPERATIONS WITH STRINGS

ASC and LEN Subroutines

1. System call address: D9DDH
2. Entry Conditions:
a. Character string information is stored in 7A04H - 7A07H as previously described.
b. YL = 60H for ASC
= 64H for LEN
3. Exit Conditions:
a. The result is in 7A00H - 7A07H
b. UH contains the error code (00H is a normal finish) if an error occurred.
4. Flags:

CHR$ Subroutine

1. System call address: D9B1H
2. Entry Conditions:
a. Integers from 0 - 255 are placed into 7A07H.
b. 7894H = 10H
3. Exit Conditions:
a. If UH = 0 then a proper exit occurred, otherwise UH contains the error code.
b. 7B10H contains the ASCII code
c. 7A04H - 7A06H contain C1H 7BH 10H
d. If the ASCII code was 00H then 7A07H contains 00H otherwise, 7A07H contains 01H.
4. Flags:

VAL Subroutine

1. System call address: D9D7H
2. Entry Conditions: string information is in 7A00H - 7A07H.
3. Exit Conditions:
a. The result is in 7A00H - 7A07H
b. UH contains the error code (00H is a normal finish) if an error occurred.
4. Flags:

STR$ Subroutine

1. System call address: D9CFH
2. Entry Conditions:
a. numeric value to be converted is in 7A00H - 7A07H
b. 7894H = 10H
3. Exit conditions:
a. The string pointer is in 7A00H - 7A07H
b. The actual character string is stored at 7B10H and following.
c. UH contains the error code (00H is a normal finish) if an error occurred.
4. Flags:

RIGHT$(X$,Y), LEFT$(X$,Y), and MID$(X$,Y,Z) Subroutines

1. System call address: D9F3H
2. Entry Conditions:

	RIGHT$	LEFT$	MID$
(7890H)	((7891H)-8	same	((7891H)-16
(7892H)	(7890H)+8	same	(7890H)+16
(7894H)	10H	10H	10H
7A00H–7A07H	Y	Y	Z
(7890H)–(7890H)+7	X$	X$	X$
(7890H)+8-(7890H)+15	--	--	Y
YL	02H	7AH	7BH

3. Exit Conditions:
a. The string pointer is in 7A00H - 7A07H
b. The actual character string is stored at 7B10H and following.
c. UH contains the error code (00H is a normal finish) if an error occurred.
4. Flags:

Note: (7890H) and (7891H) cannot be overwritten or changed. If these are changed, the routine will not function properly.

String Concatenation

1. System call address: D925H
2. Entry Conditions:
a. 7894H = 10H
b. Information on the first character string is stored in 7A00H - 7A07H
c. Information of the second character string is stored in 7A10H - 7A17H in the same format as previously described.
3. Exit Conditions:
a. Information on the new character string is placed in 7A00H - 7A07H.
b. Actual concatenated string is put in 7B10H and following memory locations.
c. If an error occurs, UH contains the error code.
4. Flags:

A CoCo Editor and File Saving Utility

Robert W. Senser
P.O. 2486
Cody, WY 82414

The first program is a file editor for the Color Computer. This editor creates and modifies text (ASCII) type files on the Color Computer. It always displays a full screen of text information, and all of the insertions, changes, and deletions are done on a single command line. A special character, the period or any other stated character, in position one of the command line is used to distinguish commands to the editor from changes to the current line. The current line is the line printed right above the command line. The command line is prompted with a '>'. This character is replaced by a '+' while inserting lines of text.

The following commands are available in the editor. Some of these commands modify the text while others work the relative pointer which determines the current line.

Command	**Function**
.B	Move current line to bottom
.C=*c*	Set command character to 'c'
.D and .D*num*	Delete line or delete 'num' lines
.E	Exit with no save of file
.I	Insert after current line
.L*string*	Locate 'string'
.P	Move current line down one page
.R1 and .R2 and .R3	Set printing range, cols. 1-30, 31-60, or 61-90
.S	Save file and exit
.T	Move current line to top
.W	Print line number of current line
.*num*	Move current line to 'num'
..*num*	Move current line to 'num'
.+*num*	Move current line down by 'num'
.−*num*	Move current line up by 'num'
null	Exit insert or move current line down 1

c=any character
num=integer number
string=string of characters
null=null string

This editor could be easily modified to work with cassette tape instead of disk by changing the file numbers used. On a 16K machine there is a maximum of 200 lines per file. If the text lines take up more than about 5,000 characters the 16K Color Computer may run out of string space. With some modifications to the dimensions of arrays P and A$, and to the M1 and M2 values, this editor should handle much larger files on the 32K machine.

```
10 REM "ED"
20 REM LINE EDITOR
30 VERIFY ON
40 PCLEAR 1
50 CLEAR 5500
60 DIM P(200),A$(250),SL(3),SR(3)
70 SL(1)=01
   : SR(1)=30
80 SL(2)=31
   : SR(2)=60
90 SL(3)=61
   : SR(3)=90
100 L=6
   : D$="."
   : S=9
   : B=0
   : F=0
   : SS=1
110 M1=200
   : M2=250
120 Z$="123456789+"
130 I$="Y"
140 SP=L-3
   : L9=(L-1)*32
150 CLS
160 PRINT
170 PRINT "   ***  ED  ***"
180 PRINT
   : PRINT
190 INPUT "FILE NAME:";F$
200 IF F$="" THEN STOP
210 INPUT "READ THE FILE (Y/N):";A$
220 IF A$="" THEN STOP
230 IF A$="N"THEN
   : A=0
   : GOTO 380
240 IF A$="Y" THEN 260
250 CLS
   : GOTO 210
260 OPEN "I",#1,F$
270 FOR I=1 TO M1
280 LINE INPUT #1,W$
290 W=I-1
300 GOSUB 1480
310 IF EOF(1)=-1 THEN 340
320 NEXT
330 GOTO 1310
340 I$="N"
350 CLOSE #1
360 A=L-2
370 REM TOP OF LOOP
380 GOSUB 1600
390 C$=">"
   : IF I$="Y" THEN C$="+"
400 PRINT @L9,C$;
410 LINE INPUT W$
420 W1$=LEFT$(W$,1)
430 IF W1$=D$ THEN 530
440 IF W$<>"" THEN 470
450 IF I$="N" THEN 550
460 I$="N"
   : GOTO 380
470 IF I$="N" THEN 910
480 W=A
490 GOSUB 1480
500 A=A+1
510 GOTO 380
520 REM COMMANDS
530 I$="N"
540 IF LEN(W$) > 1 THEN 570
550 A=A+1
560 GOTO 380
570 W$=RIGHT$(W$,LEN(W$)-1)
580 IF W$="D" THEN W=A
   : GOSUB 1400
   : GOTO 380
590 IF W$="I" THEN W=A
   : I$="Y"
   : GOTO 380
```

```
600 IF W$="B" THEN A=B
      : GOTO 380
610 IF W$="T" THEN A=1
      : GOTO 380
620 IF W$="E" THEN CLS
      : STOP
630 IF W$="P" THEN 920
640 IF W$="W" THEN PRINT @L9,"    ";A
      : GOTO 400
650 IF W$="" THEN A=A+1
      : GOTO 380
660 IF W$="R1" THEN SS=1
      : GOTO 380
670 IF W$="R2" THEN SS=2
      : GOTO 380
680 IF W$="R3" THEN SS=3
      : GOTO 380
690 IF W$="S" THEN 830
700 W1$=LEFT$(W$,1)
710 IF W1$="+" THEN 1000
720 IF W1$="-" THEN 1050
730 IF W1$=D$ THEN 950
740 IF W1$="D" THEN 1110
750 IF W1$="L" THEN 1170
760 IF W1$="C" THEN D$=MID$(W$,3,1)
      : GOTO 380
770 W$=" "+W$
780 GOSUB 1740
790 IF ER=1 THEN 1290
800 A=A+N
810 IF A>B THEN A=B
820 GOTO 380
830 OPEN "O",#1,F$
840 FOR I=1 TO B
850 W=I
      : GOSUB 1330
      : PRINT #1,W$
860 NEXT
870 CLOSE #1
880 CLS
890 PRINT F$;" SAVED"
900 STOP
910 W=A
      : GOSUB 1370
      : A=A+1
      : GOTO 380
920 A=A+S
930 IF A>B THEN A=B
940 GOTO 380
950 GOSUB 1730
960 IF ER=1 THEN 1290
970 A=N
980 IF A>B THEN A=B
990 GOTO 380
1000 GOSUB 1730
1010 IF ER=1 THEN N=1
1020 A=A+N
1030 IF A>B THEN A=B
1040 GOTO 380
1050 GOSUB 1730
1060 IF ER=1 THEN N=1
1070 A=A-N
1080 IF A<1 THEN A=1
1090 GOTO 380
1100 REM GROUP DELETE
1110 W=A
1120 GOSUB 1730
1130 FOR I=1 TO N
1140 GOSUB 1400
1150 NEXT
1160 GOTO 380
1170 M$=MID$(W$,2,LEN(W$)-1)
1180 A9=A+1
1190 IF A9>B THEN 1290
1200 FOR IL=A9 TO B
1210 W=IL
1220 GOSUB 1330
1230 L8=INSTR(W$,M$)
1240 IF L8>0 THEN 1270
1250 NEXT
1260 GOTO 1290
1270 A=W
1280 GOTO 380
1290 PRINT @1,"ERROR"
1300 GOTO 390
1310 PRINT @1,"OVERFLOW"
1320 GOTO 380
1330 REM GET
1340 IF W>B OR W<1 THEN W$="":RETURN
1350 W$=A$(P(W))
1360 RETURN
1370 REM REPLACE
1380 A$(P(W))=W$
1390 RETURN
1400 REM DELETE
1410 A$(P(W))=""
1420 B=B-1
1430 IF W>B GOTO 1470
1440 FOR ID=W TO B
1450 P(ID)=P(ID+1)
1460 NEXT
1470 RETURN
1480 REM INSERT AFTER
1490 B=B+1
1500 IF B>M1 THEN B=B-1
      : GOTO 1310
1510 F=F+1
1520 IF F>M2 THEN B=B-1
      : F=F-1
      : GOTO 1310
1530 W1=W+1
1540 FOR IA=B TO W1+1 STEP -1
1550 P(IA)=P(IA-1)
1560 NEXT
1570 P(W1)=F
1580 A$(F)=W$
1590 RETURN
1600 REM SCREEN
1610 CLS
1620 PRINT " ";Z$;Z$;Z$
1630 IF A>B THEN A=B
1640 ST=A-SP
1650 FOR IS=2 TO 15
1660 IF IS=L THEN PRINT " "
      : GOTO 1710
1670 W=ST
1680 GOSUB 1330
1690 PRINT " ";MID$(W$,SL(SS),SR(SSS))
1700 ST=ST+1
1710 NEXT
1720 RETURN
1730 REM NUMBER
1740 ER=0
1750 FOR IN=2 TO LEN(W$)
1760 N$=MID$(W$,IN,1)
1770 IF N$<"0" OR N$>"9" THEN ER=1
      : GOTO 1800
1780 NEXT
1790 N=VAL(MID$(W$,2,LEN(W$)-1))
1800 RETURN
```

The second program for the CoCo is a utility program for copying the contents of diskettes to and from audio tape in binary format.

```
10 REM 'FSAVE' PROGRAM
20 REM SAVE DISK TO TAPE OR
30 REM RESTORE TAPE TO DISK
40 REM STOPS ON ANY DISK ERROR
50 CLEAR 20,&H2CFF
```

```
6Ø W=&H2DØØ
     : W1=Ø
     : W2=Ø
8Ø GOSUB 51Ø
9Ø CLS
95 MOTORON
1ØØ PRINT @132,"PUSH 'ENTER'"
11Ø PRINT @164,"AFTER TAPE"
12Ø PRINT @196,"REWOUND";
13Ø INPUT A$
14Ø MOTOROFF
15Ø CLS
16Ø PRINT @1ØØ, "TYPE 'S' FOR SAVE"
17Ø PRINT @132, "TYPE 'R' FOR RESTORE"
18Ø PRINT @164, "TYPE 'X' TO EXIT"
19Ø INPUT A$
2ØØ IF A$="S" THEN 24Ø
21Ø IF A$="R" THEN 39Ø
22Ø IF A$="X" THEN STOP
23Ø GO TO 16Ø
24Ø REM *** SAVE ***
25Ø INPUT"DO A SAVE ('Y' OR 'N')";A$
26Ø IF A$="N" THEN 15Ø
27Ø INPUT "PUSH RECORD THEN 'ENTER'";A$
28Ø MOTOR ON
29Ø FOR I=1 TO 5ØØØ
     : NEXT I
3ØØ MOTOR OFF
31Ø FOR T=Ø TO 34
32Ø FOR S=1 TO 18
33Ø W9=W+(S-1)*256
34Ø GOSUB 61Ø
35Ø NEXT S
355 POKE &HFF4Ø,Ø
36Ø CSAVEM "SEC",W,W+46Ø7,W
37Ø NEXT T
38Ø GOTO 9Ø
39Ø REM *** RESTORE ***
4ØØ INPUT"DO A RESTORE ('Y' OR 'N')";A$
41Ø IF A$="N" GOTO 15Ø
42Ø INPUT"PUSH PLAY THEN 'ENTER'";A$
43Ø FOR T=Ø TO 34
44Ø CLOADM
45Ø FOR S=1 TO 18
46Ø W9=W+(S-1)*256
47Ø GOSUB 74Ø
48Ø NEXT S
485 POKE &HFF4Ø,Ø
49Ø NEXT T
5ØØ GO TO 9Ø
51Ø REM *** SETUP ***
52Ø A1=PEEK(&HCØØ4)
53Ø A2=PEEK(&HCØØ5)
54Ø A3=PEEK(&HCØØ6)
55Ø A4=PEEK(&HCØØ7)
56Ø D=256*A1+A2
57Ø P=256*A3+A4
58Ø POKE P+1,Ø
59Ø DEFUSR1=D
6ØØ RETURN
61Ø REM *** GET SECTOR ***
62Ø POKE P,2
63Ø POKE P+2,T
64Ø POKE P+3,S
65Ø W1=INT(W9/256)
66Ø POKE P+4,W1
67Ø POKE P+5,W2
68Ø Z=USR1(Ø)
69Ø Z=PEEK(P+6)
7ØØ REM POKE &HFF4Ø,Ø
71Ø IF Z=Ø THEN RETURN
72Ø PRINT "GET ERROR",T,S,Z
73Ø STOP
74Ø REM *** PUT SECTOR ***
75Ø POKE P,3
76Ø POKE P+2,T
77Ø POKE P+3,S
78Ø W1=INT(W9/256)
79Ø POKE P+4,W1
8ØØ POKE P+5,W2
81Ø Z=USR1(Ø)
82Ø Z=PEEK(P+6)
83Ø REM POKE &HFF4Ø,Ø
84Ø IF Z=Ø THEN RETURN
85Ø PRINT "PUT ERROR",T,S,Z
86Ø STOP
```

Scroll Revisited

Michael McNeil
67 Comfort Road
Ithaca, NY 14850

Editor's Note: Scroll was run in the December 1983 issue of Microcomputer News with critical omissions. To do justice to Mr. McNeil's efforts and for the benefit of our readers, we are re-running Scroll in its entirety in this issue.

Notice there are two BASIC program listings, Basic and Demo, and one machine language program listing, Scroll. The program Basic POKEs the machine language routine Scroll into memory where it can be accessed by Demo.The Demo program is simply the BASIC driver for Scroll. When Scroll is accessed by Demo BASIC, the graphics screen shifts up 32 bytes. I have found this routine very helpful in writing BASIC games.

These programs were written on a 16K Color Computer. However, they should work for any Color Computer with Extended BASIC.

THE PROGRAM BASIC

```
1Ø CLEAR 5ØØ, &H3FØØ
2Ø DATA 9E, BA, 1Ø, 9E, BA, 31, A8, 2Ø, A6, AØ, A7,
    8Ø, 1Ø, 9C, B7, 26, F7, 31, A8, EØ, 86, ØØ, A7,
    AØ, 1Ø, 9C, B7, 26, F7, 39
3Ø FOR L=&H3FØØ TO &H3F1D
4Ø READ A$
5Ø POKE L, VAL ("&H"+A$)
6Ø NEXT L
7Ø DEF USR 1=&H3FØØ
8Ø PMODE 2,1
    : SCREEN 1,1
    : PCLS
9Ø CIRCLE (128,96),96
1ØØ FOR L=1 TO 5Ø
11Ø Y=USR1(Ø)
12Ø NEXT L
```

SCROLL—THE MACHINE LANGUAGE LISTING

```
3FØØ                    ØØØ1Ø            ORG      $3FØØ
                        ØØ1ØØ  *DEFINE   LOCATIONS
              ØØB7      ØØ11Ø  BOTTOM    EQU      183
              ØØBA      ØØ12Ø  TOP       EQU      186
3FØØ  9E      BA        ØØ13Ø  START     LDX      TOP
3FØ2  1Ø9E    BA        ØØ14Ø            LDY      TOP
3FØ5  31      A8 2Ø     ØØ15Ø            LEAY     32,Y
3FØ8  A6      AØ        ØØ16Ø  LOOP      LDA      ,Y+
3FØA  A7      8Ø        ØØ17Ø            STA      ,X+
3FØC  1Ø9C    B7        ØØ18Ø            CMPY     BOTTOM
3FØF  26      F7        ØØ19Ø            BNE      LOOP
3F11  31      A8 EØ     ØØ2ØØ  BLACK     LEAY     -32,Y
3F14  86      ØØ        ØØ21Ø  LOOP2     LDA      #Ø
3F16  A7      AØ        ØØ22Ø            STA      ,Y+
3F18  1Ø9C    B7        ØØ23Ø            CMPY     BOTTOM
```

```
3F1B 26       F7        00240          BNE       LOOP2
3F1D 39                 00250 DONE     RTS
              0000      00260          END
00000 TOTAL ERRORS

BLACK     3F11
BOTTOM    00B7
DONE      3F1D
LOOP      3F08
LOOP2     3F14
START     3F00
TOP       00BA
```

DEMO

After running Basic which POKEs Scroll into memory, run Demo to see a demonstration of how Scroll works.

```
10 'SCROLL DEMO
20 CLEAR 500, &H3F00
30 PRINT "HAVE YOU CLOADED 'SCROLL' YET?"
40 INPUT "<Y>ES OR <N>";YN$
50 IF YN$ <> "Y" THEN GOSUB 1000
60 DEF USR1=&H3F00
70 FOR M=0 TO 4
80 PMODE M,1
     : PCLS
     : SCREEN 1,1
90 CIRCLE (128,96),96
100 FOR N=1 TO 96
     : Y=USR1(0)
     : NEXT N
110 NEXT M
120 GOTO 70
1000 PRINT "REWIND TAPE"
1010 MOTOR ON
1020 INPUT "<ENTER> WHEN THE TAPE IS AT THE
     BEGINNING",YN$
1030 CLOADM "SCROLL"
1040 RETURN
```

Using Graphics Pages and Disk Buffers

Thomas Rokicki
Box 258
College Station, TX 77841

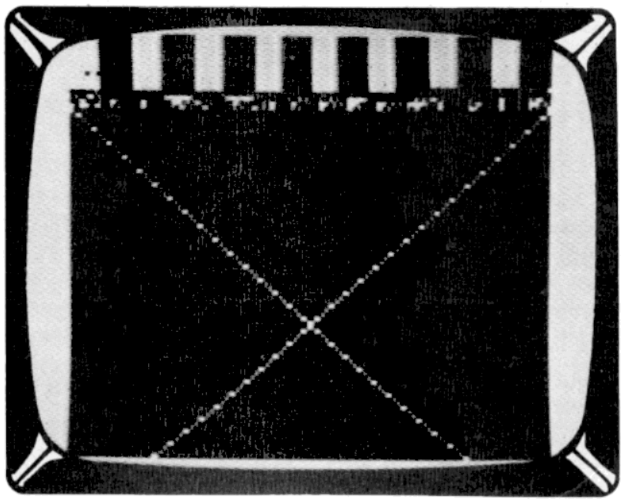
I have noticed a small peculiarity when I use graphics pages and disk buffers concurrently in the TRS-80 Color Computer. At times, the top of the graphics pages (especially in the low resolution modes) show garbage.

For instance, try this short program.

```
10 FILES 1,100
     : PMODE 0,1
     : PCLS
     : SCREEN 1,1
     : LINE(0,0)-(255,191),PSET
     : LINE(255,0)-(0,191),PSET
20 GOTO 20
```

When you run the program, the screen looks like this. I therefore delved into the depths of the Color Computer's manuals. At the end of the disk manual, there is a memory map which states that the graphics pages must start at a 256 byte page boundary.

Having worked with the 6883 SAM and 6847 VDG, I realized that the Color Computer could only display pages



starting at 512 byte pages. Therefore, at times, the Color Computer would show junk on the top of the screen.

Doing some experimenting, I soon came up with Table 1, which will solve the problem. Whenever the FILES statement is used, CLEAR the appropriate amount of string space. To use Table 1, take the string space number given after the number of files. If it is not big enough for the file string space you need, add 512. If it is still not big enough, keep adding 512 until it is.

Take, for instance, a program with four files requiring at least 2000 bytes of file space. The number given for four files is 249. Add 512 to 249 and you get 761. Not big enough. Add 512 again for 1273. Keep adding: 1273 + 512 = 1785, 1785 + 512 = 2297. At last, big enough.

Now just start the program with the statement: FILES 4,2297 and your graphics will work, and you will not be wasting memory.

This example takes no more memory than the statement FILES 4,2042 (try it) and gives one a bit more room. It also insures that the graphics pages work correctly.

Table 1.

Number of Files	String Space
0	349
1	68
2	299
3	18
4	249
5	480
6	199
7	430
8	149
9	380
10	99
11	330
12	49
13	280
14	511
15	230

"13 GHOSTS"

by Bryan Eggers
Software Affair

The majority of games sold by Radio Shack are for the Color Computer. The high resolution screen and color graphics make it easy to draw pictures. However, it is also possible to create a great arcade game on the Model III or 4. I always felt that if someone took full advantage of certain characteristics of these computers, that many interesting visual effects could be obtained, particularly in the area of animation.

The concept of "13 GHOSTS™", as well as the graphics and animation sequences, took me a few years to develop because I could only work on it in my spare time. I eventually admitted to myself that I didn't have enough technical knowledge to assemble my game scenario and finished graphics into a complete machine-language game. I needed some help, actually, a LOT of help! I was looking for someone who not only could put the game together in assembly language and make it play according to the rules, but who could also solve some critical problems:

(1) Since the game consisted of virtually non-stop animation, it was critical that we eliminate the screen "flicker" that plagued all other animated games.

(2) Every ghost on the screen required independent movement logic, including animation capability, direction and speed variation. Each ghost needed to be in its own visual "plane." That is, one ghost could move behind another ghost for a multi-dimensional effect, and yet all the ghosts were supposed to appear "in front of" the background. This would require a special black "border" around each ghost for contrast.

(3) The action could not stop for even a split-second or the animation effect would be ruined. I needed smooth continuous action, even with multiple animated explosions, ghosts flying in and out of the area, spiders going up and down, score updates, background moves, and sound effects occurring simultaneously.

(4) The target sight, the actual "shot" itself, and subsequent explosions each required animation and independent control. Two different types of explosions were needed, and one had to be smart enough to change each of its particles from "white" to "black" for contrast against the background.

(5) I wanted spectacular music and sound effects in three-part harmony played through the cassette port. My friend Jon Bokelman (author of "Orchestra-90") gave me permission to use one of his synthesizer routines within the game. I went ahead and wrote all the music and sound effects, but I still needed to produce the sound effects simultaneously with the animation.

(6) The background graphics and animation sequences took up so much memory that some sort of compiler was needed to reduce the data to manageable size.

To make a long story short, Larry Payne is the co-author of "13 GHOSTS" and the talented programmer who solved all the problems that were driving me crazy. He also added many special effects and ideas of his own. Without him, this game would still be a pile of graphic worksheets and notes collecting dust on my shelf. I'm certainly happy that we put it together and licensed it to Radio Shack instead!

Since the instructions that accompany the game are rather brief, I thought you might like to read a complete game scenario and detailed instructions. I'll also give you a couple of tips on how to improve your score.

The illustrations show some of the graphics used in the game, but obviously the animation can't be shown. You'd have to see this game in action to really appreciate it!

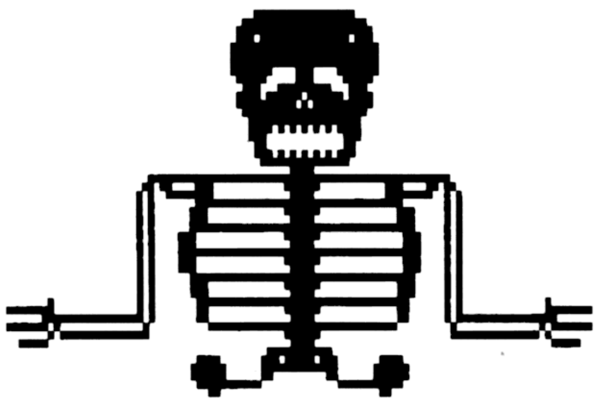
GAME SCENARIO OF "13 GHOSTS"

Out West, you are known as the Great Venicksia—a legendary conjuror, sorcerer, and wizar , capable of amazing feats of magic.

The legend of your ability to rid a town of undesirable evil spirits is widely known. Armed only with a special weapon of your own design, a "Ghost-Blaster," you are often summoned to local towns to exterminate unwanted GHOSTS that have been haunting the streets.

You arrive by train at the local DEPOT. Once you leave the depot, there is no turning back. You plan to walk through town, all the way to the HAUNTED HOUSE, then back again, shooting ghosts as you go. Unlike the other towns, however, you find so many ghosts here that you will be lucky to make it back to the train alive!

You are in trouble! This is a real GHOST TOWN and there are 13 GHOSTS trying to scare you to death! From your previous battles with ghosts, you know that no matter how hideous and grotesque a ghost is, it can't actually harm you unless you let it frighten you to death, so try to stay calm!



You keep shooting ghosts, knowing that for each one you let escape, a more dangerous ghost will take its place! And you know that lurking somewhere in the darkness is the SKELETON. Once the last four (most dangerous) ghosts appear, any ghost you let escape will wake the SKELETON, which means sudden DEATH! The SKELETON will run out and kill you!

There is no defense against a SKELETON. You must try to keep shooting ghosts, because even though they rematerialize almost immediately after being "destroyed," at least you'll keep them from escaping and waking up more dangerous ghosts and, ultimately, the SKELETON.

Each time you shoot a ghost, you move a few steps further in your journey. The more dangerous the ghost, the more ground you gain by destroying it.

Even if you return to the DEPOT safely, you are compelled to take the perilous journey again and again. Each trip becomes more dangerous. You only get a brief rest each time you reach the HAUNTED HOUSE or the DEPOT.

One spirit, known as "The Laughing Ghost" has a very dangerous habit. He pops up in various places and laughs at you. If you don't shoot him before he disappears, he'll also wake up a more dangerous ghost!

Watch for SPIDERS! They can help you! GHOSTS do not like spiders. When SPIDERS appear, ghosts start avoiding the area. If you actually shoot a spider, the most dangerous ghost currently chasing you will be scared away.

PLAYING THE GAME

SYSTEM REQUIRED: 48K Disk Model III/4

Connect the cassette port plug to a small amplifier! This is important, since 13 GHOSTS plays AMAZING music and sound effects in 3-PART HARMONY through the cassette port! Some of the effects will help you during the game.

Insert the "13 GHOSTS" disk in drive 0 and press (RESET).

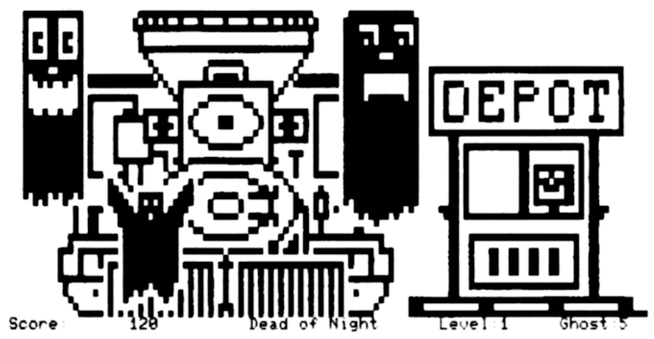
Type the name "GHOSTS" at TRSDOS READY. After a brief graphic and musical introduction, press any key (except (CLEAR)) to start the game. If no keys are pressed, the HIGH SCORES will be displayed, and a few seconds later, a self-playing demonstration game will begin. Hit any key to stop the demonstration game and start a real game.

The "Ghost-Blaster" is represented by a blinking cursor. The ARROW KEYS may be used to move the cursor anywhere on the screen. The (SPACE) bar is used to shoot your "Ghost-Blaster." Up to THREE SHOTS may be in the air at one time! Note that each shot takes a moment to reach its target, just like any other conventional weapon, so it may be necessary to shoot slightly in front of your target to score a hit. It takes a direct hit to dematerialize a ghost. A hit on the edge of a ghost may be ineffective.

Use your "Ghost-Blaster" to shoot the ghosts before they escape from the top OR bottom of the screen. In the beginning, the ghosts will try to frighten you by appearing at the bottom of the screen and escaping at the top of the screen. Once they discover that you can't be scared so easily, they'll go off the top of the screen, then come back and try to scare you again on the way down! This gives you a second chance at them, since now they can only escape by leaving the screen at the bottom. Shoot them before they escape! Each ghost that escapes will wake up a more dangerous ghost and eventually, wake up the SKELETON!

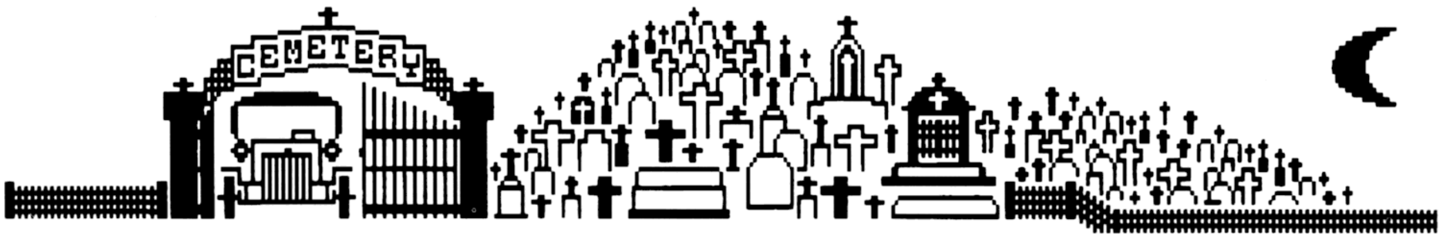
Remember, you must continuously blast and dematerialize the ghosts to keep them from escaping! Each ghost that escapes wakes up a more dangerous ghost. When there are no more ghosts left to wake up, the next ghost that escapes wakes up the SKELETON!

One shot can dematerialize only one ghost.



FUNCTION OF CONTROL KEYS

KEY	FUNCTION
(←)(→)(↑)(↓)	Move "Ghost Blaster" in desired direction.
(SPACE)	Shoot "Ghost Blaster." You can have three shots in the air at once. Holding down the (SPACE) bar auto-fires, but not as fast as manual firing. Also, you can't move your "Ghost Blaster" as fast while you're auto-firing!
(CLEAR) + (BREAK)	Restart the game. Effective any time except while music is playing.
(CLEAR)	Pressing this key while the "13 GHOSTS" title page is on the screen will display the HIGH SCORE screen, except while music is playing. If no keys are pressed then the HIGH SCORES appear automatically after a few seconds, followed by an auto-play demonstration mode.
(P)	Pause. Effective anytime during game ex-



cept while music is playing. Press any key to continue.

NOTE: All keys are disabled while music is playing.

This game requires a system RESET to return to TRSDOS.

SCORING

Each TRIP between the DEPOT and the HOUSE (either direction) is scored as a LEVEL. The current level of play is continuously displayed on the bottom line of the screen.

In LEVEL 1, each ghost that you shoot results in a score equal to ten times the ghost's number. For example, shooting Ghost 4 would result in 40 points. In LEVEL 2, and every subsequent LEVEL, each ghost is worth its number times 100, as shown in this chart:

	LEVEL 1 SCORE	LEVEL 2 (and higher) SCORE
Ghost 1	10	100
Ghost 2	20	200
Ghost 3	30	300
Ghost 4	40	400
Ghost 5	50	500
Ghost 6	60	600
Ghost 7	70	700
Ghost 8	80	800
Ghost 9	90	900
Ghost 10	100	1000
Ghost 11	110	1100
Ghost 12	120	1200
Spider	100	1000
Laughing Ghost	500	5000

The LAUGHING GHOST and SPIDER are not numbered ghosts. Shooting these characters results in the special points as shown and also affects the play of the game in certain ways.

ADDITIONAL FEATURES

Each ROUND TRIP (depot-to-house and back) has a different name, visual situation and difficulty level. These represent journeys at various times of day and under special conditions.

The name of the current situation appears at the bottom center of the screen during play. For example, the first round trip is called "THE DEAD OF NIGHT." This action obviously takes place at night. You are in this situation through LEVELS 1 and 2. (The situation names are not displayed during the auto-play demonstration mode.)

The second ROUND TRIP is called "GOOD MOURNING" and takes place during the daylight hours. This situation occurs in LEVELS 3 and 4.

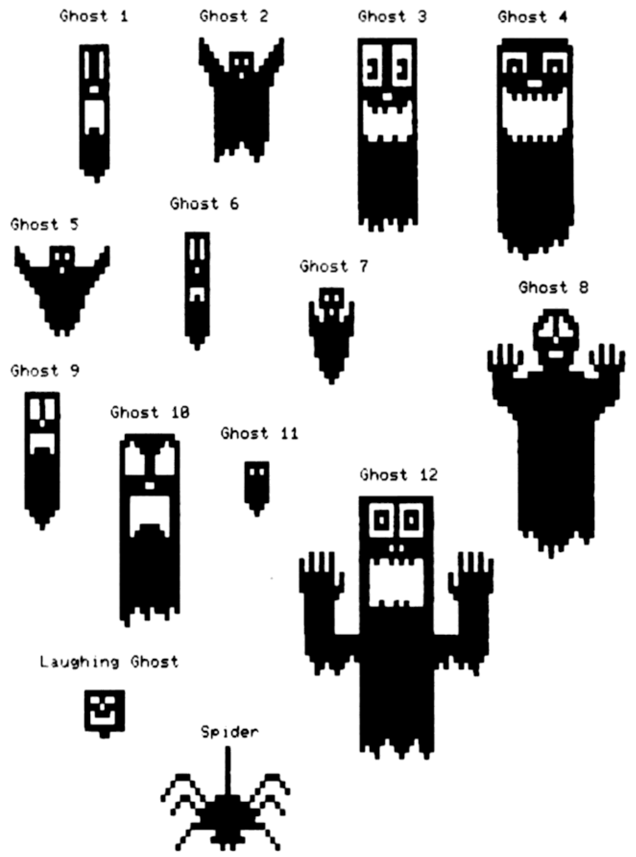
Subsequent trips, situations, visual effects and difficulty levels will be left for you to discover! If you survive all of the first 8 LEVELS (4 round-trips), you will eventually return to the original DEPOT scene and get the opportunity to do it all again THREE MORE TIMES (24 more LEVELS!).

You will need an exceptional amount of skill (and luck) to make it through these additional LEVELS, since the ghosts get smarter and use completely different flight patterns after every 8 LEVELS! The spiders move faster, too. That makes every one of the 32 LEVELS different!

ADDITIONAL DISPLAY INFORMATION

The most critical information on the display is the GHOST COUNTER. This number represents the HIGHEST NUMBERED (most dangerous) GHOST currently active. Each time you let a ghost escape, that ghost wakes up a higher-numbered ghost to take its place.

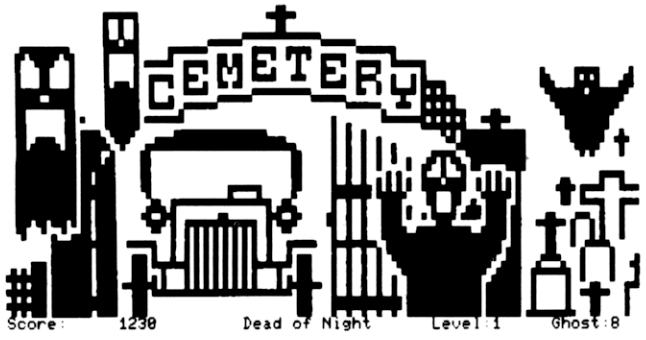
But, if there are no more ghosts to wake up, the next ghost that escapes wakes up the SKELETON and he kills you! The skeleton cannot be defeated! The game is over.



If ghost #12 becomes active, the entire display line starts flashing. This is a WARNING indicating that there are no more sleeping ghosts, and the next ghost that you let escape will wake the SKELETON!!

Shooting a SPIDER scares away the highest numbered active ghost! You can verify this by watching your GHOST COUNTER.

You must shoot the LAUGHING GHOST before it stops laughing and disappears! It will wake up a more dangerous ghost if you let it escape (disappear). But, if you shoot some other ghosts first, causing the LAUGHING GHOST to be moved off the side of the screen before it can disappear, it won't wake up another ghost. It will be back very soon to try again, though!



As long as you continue to blast the ghosts, they will be unable to leave the area to warn the others. They'll just keep rematerializing and trying to escape. Each time you shoot a ghost you move farther along in your trip.

When you finally reach the end of a trip, the GHOST COUNTER is reset and you get a short rest accompanied by a musical interlude. When the music finishes playing the ghosts appear again and you start traveling in the OPPOSITE direction.

Every trip (level) STARTS with ghosts 1, 2, 3, and 4. At the end of the game your final score is displayed on the left side of the skull. The level number you achieved is displayed on the right side, along with the final ghost counter status (13). After the skull laughs, and if you have a TOP TEN score, you'll see a "GREAT SCORE" message and you'll be given the opportunity to enter your name. Your score will be merged into the TOP TEN SCORES in the appropriate position.

If the skull laughs and you do not immediately see a "GREAT SCORE" message, you'll know that your score wasn't high enough to be in the TOP TEN. You have two options at this point. You can press any key to immediately start a new game or, if you wait a few seconds, the program will automatically display the current TOP TEN HIGH SCORES, then restart the game. In either case, you'll always get to hear the "13 GHOSTS" theme music that introduces each new game.

BONUS TRIP

If you reach the end of your trip, either to the DEPOT or the HOUSE, with the GHOST COUNTER on 12, you get a BONUS TRIP!

You'll hear a special Bonus Tune, then you'll ZOOM all the way to the other end of the town! During this BONUS TRIP you will receive a bonus score of TWICE what would normally be possible if you had played the trip! And, you'll move so fast that the ghosts won't have time to wake up!

It's very dangerous to try for a BONUS TRIP because when the GHOST COUNTER is on 12, you will LOSE if you let just one more ghost escape. Be careful!

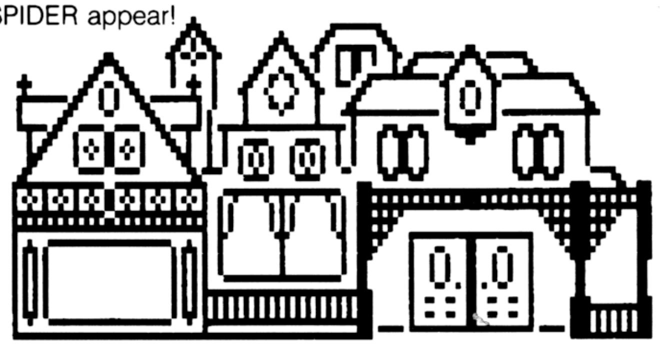
ADVANCED PLAYER STRATEGY

Once you learn to shoot the ghosts easily, you can speed things up in the first few levels by letting a few ghosts escape. Shooting the high-numbered ghosts will advance you to the end of the trip much faster. The higher the number of the ghost you shoot, the further you advance on your trip.

Obviously, there is a big risk involved in this, since you now have fewer ghosts to lose before the SKELETON makes his appearance. You'll have to be very careful not to miss any ghosts. Keep an eye on the GHOST COUNTER!

You can also let some ghosts escape to get BONUS TRIPS. You get a BONUS TRIP and extra points if you arrive at the DEPOT or HOUSE with the GHOST COUNTER on 12. Very dangerous, though! Always try to shoot SPIDERS! As we mentioned earlier, ghosts are afraid of spiders. This can create a problem for you because if some ghosts escape, the new ghosts probably won't immediately enter the area because of the spiders. The GHOST COUNTER can't detect these lurking ghosts, so when the spiders move back up to their web, you might suddenly have a bunch of unexpected ghosts show up! Maybe even enough to wake up the SKELETON!

Try to discover what makes the LAUGHING GHOST and SPIDER appear!



HIGH SCORES

13 GHOSTS saves the HIGHEST TEN SCORES on your disk. These can be displayed when the "13 GHOSTS" title is on the screen by pressing the CLEAR key (wait until the music stops).

If your current score ranks within the TOP TEN scores, and after some appropriate music, you will be asked to input your name for the records. Carefully type in your name, then press ENTER. If you make a mistake, pressing CLEAR or a ← will clear the complete line and allow you to start again.

If you do not wish to add your score to the HIGH SCORES screen, simply press BREAK to start a new game.

The HIGH SCORES are stored on your disk in a file named GHOSTS/HIS. You can kill this file if you want to start a new set of HIGH SCORES.

If you complete all 32 LEVELS of the game, an asterisk (*) will appear next to your score. This proves that you BEAT the game! Immediately after beating the game you will be rewarded with a special visual/musical display. Very few people will ever get to this level. We consider it to be quite an accomplishment!

If you do beat the game, you might want to take a picture of this special screen image to prove to your friends that you actually BEAT "13 GHOSTS," otherwise they will probably never believe you! This final display remains on the screen until any key is pressed.

"13 GHOSTS" is a Trademark of Software Affair, Ltd.

Radio Shack Computer Centers Now 427 Nationwide

ALABAMA
BIRMINGHAM 2428 Green Springs Hwy., 9233 Parkway East
HUNTSVILLE 1400 N. Memorial Pkwy
MOBILE 405 Bel-Air Blvd
MONTGOMERY #24 Union Square S/C

ARIZONA
PHOENIX 10233 Metro Pkwy. E., 4301 N. 7th Ave.
SCOTTSDALE 2525 N. Scottsdale Rd
TEMPE 83 E. Broadway
TUCSON 5622 E. Broadway, Campbell Plaza, 2830 N. Campbell Ave

ARKANSAS
LITTLE ROCK Town & Country S/C, University & Asher

CALIFORNIA
ANAHEIM 509 Katella
BAKERSFIELD 2018 Chester Ave
BERKELEY 1922 Grove St.
BEVERLY HILLS 8500 Wilshire Blvd
BREA Imperial Shopping Center, 391 South State College
CANOGA PARK 8371 Topanga Canyon
CARMICHAEL 6305 Fairoaks Blvd
CHICO 1834 Mangrove Ave
CHULA VISTA 1201 3rd Ave
CITRUS HEIGHTS 7405 Green Back Ln. at San Juan
DOWNEY 8031 Florence Ave
EL TORO El Toro Rd. at San Diego Freeway
ESCONDIDO 347 W. Mission Ave
FREMONT Fremont Hub, 39114 Fremont
FRESNO Princeton S/C, 2721 N. Blackstone Ave
GARDEN GROVE 12821 Knott St
GLENDALE 236 N. Brand Blvd.
HAYWARD 24784 Hesperian Blvd
HOLLYWOOD 6922 Hollywood Blvd
IRVINE Redhill at Main St
LAKEWOOD 5830 Lakewood Blvd
LA MESA 5346 Jackson Dr.
LONG BEACH 2119 Bellflower Blvd.
LOS ANGELES 740 S. Olive St., 5240 Century Blvd. (Airport area)
MODESTO 221 McHenry Ave
MONTCLAIR 5237 Arrow Hwy
MONTEREY 484 Washington St.
MOUNTAIN VIEW 1933 El Camino Real W
OAKLAND 1733 Broadway
PASADENA 575 S. Lake Ave
PLEASANT HILL 508 G. Contra Costa Blvd
RIVERSIDE 3844 La Sierra Ave
SACRAMENTO 4749 J. St.
SAN BERNARDINO 764 Inland Center Dr.
SAN DIEGO 3062 Clairemont Dr.; 3902 El Cajon Blvd.
SAN FRANCISCO One Market Place; 2920 Geary Blvd
SAN JOSE 1228 S. Bascom Ave.
SAN MATEO 3180 Campus Dr.
SANTA ANA 2320 S. Fairview St. (at Warner)
SANTA BARBARA 4141 State St. A-1
SANTA FE SPRINGS 14138 East Firestone Blvd.
SANTA MONICA 511 Wilshire Blvd.
SANTA ROSA 823 4th St
SHERMAN OAKS 14936 Ventura Blvd
STOCKTON College Sq. S/C, 963 West March Lane
TARZANA 18545 Ventura Blvd
TORRANCE 3840 Sepulveda at Hawthorne
VENTURA 4005 E. Main St.
WEST COVINA 2516 E. Workman St

COLORADO
AURORA Hoffman Heights Shop. Ctr
BOULDER Arapahoe Plaza, 3550 Arapahoe
COLORADO SPRINGS 4341 N. Academy
DENVER 8000 E. Quincy, Green Briar Plaza, 7075 Pecos
LAKEWOOD 2099 Wadsworth Blvd.

CONNECTICUT
EAST HAVEN 51 Frontage Rd
FAIRFIELD 1196 Kings Hwy. & Rt. 1
HARTFORD The Richardson Bldg., 942 Main St.
MANCHESTER 228 Spencer St.
NORWALK Rt. 7-345 Main Ave
ORANGE Edwards Food Warehouse Shop. Ctr., 538 Boston Post Rd.
WATERBURY 105 Bank St.
WATERFORD 122 Boston Port Rd.
WEST HARTFORD 39 S. Main St

DELAWARE
DOVER Edgehill Shop. Ctr., Rt. #13
WILMINGTON 3847 Kirkwood Hwy.

DISTRICT OF COLUMBIA
N.W. WASHINGTON 1800 M St., 15th & L St. Van Ness Station, 4250 Connecticut Ave.

FLORIDA
ALTAMONTE SPRINGS 766 B. East Altamonte Dr
BOCA RATON 1662 N. Federal Hwy
CLEARWATER 2460-D US 19 North
DAYTONA BEACH Volusia Plaza
FT. LAUDERDALE 4368 N. Federal Hwy.
FORT MYERS Edison Mall
FORT PIERCE Center West Shopping Center, U.S. Hwy. 1
GAINESVILLE 3315 Archer Rd
HOLLYWOOD 429 S. State Rd. #7
JACKSONVILLE 8252 Arlington Exprwy., Roosevelt Mall, Roosevelt Blvd
LAKELAND 2810 S. Florida Ave
LAUDERDALE LAKES 4317-25 N. State Rd. 7
MERRITT ISLAND 700 E. Merritt Island Causeway
MIAMI 9459 S. Dixie Hwy.; 1601 Biscayne Blvd., 15 SE. 2nd Ave., 20761 S. Dixie Hwy
N. MIAMI BEACH The Promenade, 1777 N.E. 163rd St
ORLANDO 1238 E. Colonial Dr
PENSACOLA Eastgate Shopping Center, 6925 N. 9th Ave
ST. PETERSBURG 3451 66th St. N
SARASOTA 5251 S. Tamiami Tr. (Hwy.41)
TALLAHASSEE 2529 S. Adams
TAMPA 4555 W. Kennedy; 1825 E. Fowler Ave.
W. PALM BEACH 2271-A Palm Beach Lakes Blvd

GEORGIA
AUGUSTA 3435 Wrightsboro Rd.
ATLANTA 2108 Henderson Mill; 49 W. Paces Ferry; Akers Mill S/C, 2937 Cobb Parkway NW., 113 Peachtree St.
COLLEGE PARK 5309 Old National Hwy.
COLUMBUS Columbus Square Mall, 3050 Macon Rd.
DORAVILLE 5697 Buford Hwy
MACON 1467 Eisenhower Pkwy.
SAVANNAH Chatham Plaza, 7805 Abercorn St.

HAWAII
HONOLULU Pacific Trade Center-Ground Floor, 190 South King St.

IDAHO
BOISE 691 S. Capitol Blvd.

ILLINOIS
AURORA 890 North Lake St.
BLOOMINGTON 401 N. Veterans Parkway (Eastland Commons)
CHICAGO 4355 S. Archer Ave., CNA Plaza, 309 S. Wabash; 72 West Adams, 524 N. Michigan Ave., 101 W. Washington St. at Clark
DEERFIELD Deerbrook Mall
ELMWOOD PARK 7212 W. Grand Ave
FAIRVIEW HEIGHTS #4 Market Place
HOMEWOOD/GLENWOOD 329 Glenwood Lansing, 18230 S. Halsted at 183rd
JOLIET 2415 W. Jefferson St.
LaGRANGE One S. LaGrange Rd
LIBERTYVILLE 1350 S. Milwaukee Ave.
LOMBARD 4 Yorktown Center
MOLINE 4401 44th Ave
NILES 8349 Golf Rd.
OAK LAWN 4815 W. 95th St.
PEORIA 4125 N. Sheridan Rd
ROCKFORD North Town S/C, 3600 N. Main St
SCHAUMBURG 651 Mall Dr
SPRINGFIELD Sherwood Plaza, 2482 Wabash

INDIANA
EVANSVILLE 431 Diamond Ave
FT. WAYNE 747 Northcrest S/C
GRIFFITH 208 W. Ridge Rd
HOBART Save-More Plaza, Rtes. #6 & 51
INDIANAPOLIS 6242 E. 82nd St., Castleton Plz., Speedway Plaza, 6129 B Crawfordsville; 10013 E. Washington St., Two W Washington St
SOUTH BEND 1827 South Bend Ave
TERRE HAUTE 3460 U.S. Hwy. 41 S

IOWA
CEDAR RAPIDS 111 First Ave., S.E. (Downtown)
DAVENPORT 616 E. Kimberly Rd
DES MOINES 7660 Hickman Rd., Sherwood Forest S/M

KANSAS
OVERLAND PARK 8619 W. 95th
TOPEKA White Lakes Plaza, West Tower, 3715 Plaza Dr.
WICHITA 2732 Blvd. Plaza S/C

KENTUCKY
FLORENCE 7727 Mall Rd
LEXINGTON 2909 Richmond Rd
LOUISVILLE Louisville Galleria, 4133 Shelbyville Rd.

LOUISIANA
ALEXANDRIA 1213 Texas Ave
BATON ROUGE 7007 Florida Blvd
GRETNA 400 La Palco Blvd
HOUMA 2348 W. Park Ave. (Hwy. 24)
LAFAYETTE University Square at Congress Blvd.
METAIRIE 3750 Veterans Hwy
NEW ORLEANS 327 St. Charles Ave
SHREVEPORT 1545 Line Ave.

MAINE
BANGOR Maine Square

MARYLAND
BALTIMORE 7942 Belair Rd., Putty Hill Plaza, 115 N. Charles St at Lexington
BETHESDA 7900 Wisconsin Ave.
CATONSVILLE One Mile West S/C, 6600 B Balt. Nat'l Pike
FREDERICK Shoppers World, Rt. 40W
NEW CARROLLTON-LANHAM 7949 Annapolis Rd.
PASADENA 8120 Ritchie Hwy.
ROCKVILLE Congressional Plaza, 1673 Rockville Pike
SALISBURY Shoppers World S/C, Rt. 50
TEMPLE HILLS 4520 St. Barnabas Rd.
TOWSON-LUTHERVILLE Yorktown S/C York Rd. at Ridgley Rd

MASSACHUSETTS
BOSTON 730 Commonwealth Ave., 111 Summer St.
BOSTON 372 Boyston
BRAINTREE South Shore Plaza, 250 Granite St
BROCKTON 675 Belmont
BURLINGTON Crossroads Plaza, Rt. 3 S.
CAMBRIDGE Harvard Square, 28 Boylston St.
CHESTNUT HILL 200 Boylston St
FALL RIVER 90 William S. Canning Blvd
NATICK 1400 Worcester Rd.
SAUGUS 343 Broadway
SPRINGFIELD 1985 Main St., Northgate Plz
WORCESTER Lincoln Plaza

MICHIGAN
ANN ARBOR 2515 Jackson Rd.
BIRMINGHAM 3620 W. Maple Rd
DEARBORN Westborn Shop. Ctr., 23161 Michigan Ave
DETROIT DWNTN 1559 Woodward Ave.
FLINT G3298 Miller Rd., Yorkshire Plaza
GRAND RAPIDS 3142 28th St. SE.
KALAMAZOO 25 Kalamazoo Center
LANSING 2519 S. Cedar St.
PLAINFIELD North Kent Mall
PONTIAC North Oak Plaza-Pontiac Mall, 2436 Elizabeth Lake Rd
LIVONIA 33470 W. 7 Mile Rd.
ROSEVILLE 31873 Gratiot Ave
SOUTHFIELD 17651 West 12 Mile Rd.,
TROY Oakland Plaza, 322 John R. Rd.
WARREN 29038 Van Dyke Ave.

MINNESOTA
BLOOMINGTON 10566 France Ave. S
FRIDLEY 7974-76 University Ave. North
GOLDEN VALLEY Golden Valley S/C, 8016 Olson Memorial Hwy
MINNEAPOLIS 830 Marquette Ave.
ST. PAUL 6th & Wabasha

MISSISSIPPI
GULFPORT 516A Courthouse Rd.
JACKSON 979 Ellis Ave.

MISSOURI
DES-PERES 11960 Manchester Rd
FLORISSANT 47 Florissant Oaks S/C
INDEPENDENCE 1325 S. Noland Rd.
KANSAS CITY 4025 N. Oak Trafficway
ST. ANN 10472 St. Charles Rock Rd.
ST. LOUIS 500 No. Broadway (Commerce Bank Bldg., Downtown)
SPRINGFIELD 2684 S. Glenstone

NEBRASKA
LINCOLN 4601 "O" St.
OMAHA 3006 Dodge St.; 1318 72nd St. at Pacific

NEVADA
LAS VEGAS Commercial Center, 953 E. Sahara #31-B
RENO 3328 Kietzke Lane

NEW HAMPSHIRE
MANCHESTER Hampshire Plaza, 1000 Elm St.
NASHUA 429 Amherst St., Rt. 101A

NEW JERSEY
BRIDGEWATER 1472 U.S. Highway 22 East
E. BRUNSWICK 595 A Rt. 18
E. HANOVER Rt. 10, Hanover Plaza
LAWRENCEVILLE Rt. 1 & Texas Ave.
NEWARK 595 Broad
NORTHFIELD 322-24 Tilton Rd.
PARAMUS 175 Rt. 17 S.
SPRINGFIELD Rt. #22 Center Isle
TOMS RIVER 700 Rt. 37 West
VOORHEES 35 Eagle Plaza

NEW MEXICO
ALBUQUERQUE 2108 San Mateo NE.

NEW YORK
ALBANY Shoppers Pk., Wolf Rd.
BAYSHORE 1751 Sunrise Hwy
BETHPAGE 422 N. Wantagh Ave.
BROOKLYN 531 86th St.
BUFFALO 839 Niagara Falls Blvd.
FRESH MEADOWS 187-12 Horace Harding Exp.
GARDEN CITY 960 Franklin Ave
JOHNSON CITY Giant Shopping Center, Harry L. Drive
KINGSTON Kings Mall, Rt. 9W
MANHASSET 1550 Northern Blvd.
MELVILLE TSS Mall, Rt. 110
NEWBURGH Zayre Plaza, Rt. #17K
NEW ROCHELLE 211 North Ave
NEW YORK 385 Fifth Ave.; 139 E. 42nd St.; 19 W. 23rd St.; 347 Madison Ave.; 270 Park Ave. South; 1282 Broadway, 9 Broadway
NIAGARA FALLS Pine Plaza, 8351 Niagara Falls Blvd
REGO PARK 97-77 Queens Blvd
ROCHESTER 3000 Winton Rd.
SCARSDALE 365 Central Park Ave.
SCHENECTADY Woodlawn Plaza
SPRING VALLEY White House Center, 88 W. Rt. 59
STATEN ISLAND 2409 Richmond Ave
SYRACUSE 2544 Erie Blvd., Hotel Syracuse, 510 S. Warren St
UTICA Riverside Mall
VALLEY STREAM Green Acres Shop. Ctr.
YONKERS Cross Country Shop. Ctr.

NORTH CAROLINA
ASHEVILLE K-Mart Shopping Center, Tunnel Rd
CHARLOTTE 3732 Independence Blvd., Tyvola Mall, 5401 South Blvd
DURHAM South Square Mall
FAYETTEVILLE Eutaw Shopping Center, 815 Elm St.; Fayetteville Street Mall
GREENSBORO 3718 High Point Rd.
RALEIGH Townridge Sq., Hwy. 70 W.
WILMINGTON Independence Mall
WINSTON-SALEM 629 Peters Creek Pkwy.

OHIO
AKRON Fairlawn Plaza, 2727 W. Market St
BEDFORD HEIGHTS 5217 Northfield Rd.
CANTON 5248 Dressler Rd. NW., Mellet Plaza, 3826 W. Tuscarawas,
CENTERVILLE 2026 Miamisburg-Centerville Rd.
CINCINNATI 9725 Montgomery, 16-18 Convention Way (on Skywalk)
CLEVELAND 419 Euclid (Dwntwn), 27561 Euclid Ave
COLUMBUS 862 S. Hamilton, Great Eastern S/C, The Patio Shop. Ctr., 4661 Karl Rd.; 400 N. High St.
DAYTON Northwest Plaza, 3279 West Siebenthaler
ELYRIA 286 Midway Blvd.
FAIRFIELD 7255 Dixie Hwy. (1/4 Mi. North of I-275)
NORTH OLMSTED Great Northern S/C
PARMA 7551 W. Ridgewood Dr
TOLEDO 5844 W. Central Ave., Brownstone Plaza, 1724 S Reynolds Rd.
YOUNGSTOWN Union Square Plaza, 2543 Belmont Ave

OKLAHOMA
OKLAHOMA CITY 4732 SE 29th St., Springdale S/C, 4469 NW 50th, 1101 SW 59th St.
TULSA 7218 & 7220 E. 41st St

OREGON
EUGENE 390 Coburg Rd.
PORTLAND 7463 SW Barbur Blvd., 9131 SE Powell, 3rd and Washington Sts. (Downtown)
SALEM Salem Plaza, 403 Center

PENNSYLVANIA
ALLENTOWN Crest Plaza S/C, Cedar Crest Blvd. US 22
BALA CYNWYD 67 E. City Line Ave
EASTON 25th St. Shopping Center
ELKINS PARK Elkins Park Square, 8080 Old York Rd.
ERIE 5755 Peach St.
HARRISBURG Union Deposit Mall, Union Deposit Rd. #17
LANCASTER Park City Plaza, US 30
MONROEVILLE 3828 Wm. Penn. Hwy
MONTGOMERYVILLE Airport Sq., Rt. 309
PHILADELPHIA 7542 Castor Ave., 1002 Chestnut St., 1801 Market St., 10 Penn Center
YORK York County Shopping Center
PITTSBURGH 5775 Baptist Rd., Hills Plaza; 303 Smithfield St., 4643 Baum Blvd.; 4768 McKnight Rd.
SCRANTON 206 Meadow Ave.
WILKES BARRE, 23 W. Market St.
WYOMISSING Berkshire Mall West, 1101 Woodland Rd.

PUERTO RICO
HATO REY 243 Franklin D. Roosevelt Ave

RHODE ISLAND
E. PROVIDENCE 850 Waterman Ave.
PROVIDENCE 177 Union St.

SOUTH CAROLINA
COLUMBIA Old Sears Bldg., 1001 Harden St.
GREENVILLE N. Hills S/C
N. CHARLESTON 5900 Rivers Ave.

SOUTH DAKOTA
SIOUX FALLS 1700 S. Minnesota at 25th

TENNESSEE
CHATTANOOGA 636 Northgate Mall
JOHNSON CITY Peerless Center
KNOXVILLE Cedar Bluff S/C, 9123 Executive Park Dr.
MEMPHIS 4665 American Way; 1997 Union Ave.
NASHVILLE 2115 Franklin Pike; Rivergate Plaza

TEXAS
AMARILLO Wellington Sq. S/C, 1619 S. Kentucky
ARLINGTON 2500 E. Randol Mill, Suite 113
AUSTIN 8764 E. Research Blvd.; Southwood Mall, 1501 Ben White Blvd.
BROWNSVILLE 1639 Price Rd. (Hwy. 77)
BEAUMONT 5330 Eastex Frwy.
COLLEGE STATION 2414 Texas Ave., South
CORPUS CHRISTI 1711 S. Staple St.
DALLAS 15340 Dallas Pkwy., Suite 1100; 2930 W. Northwest Hwy.; 1517 Main St.; 2588 Royal Ln.
EL PASO 9515 Gateway West; Kern Plaza Shopping Center, 3100 N. Mesa
FT. WORTH 231 One Tandy Center; 2801 Alta Mere
GALVESTON 5924 Broadway
HARLINGEN 1514 S. Hwy 77, Sunshine Strip
HUMBLE 19300 "B" Hwy. 59 (at FM 1960)
HOUSTON 211C-FM 1960; 10543 Gulf Fwy.; 5900 North Fwy., 6813 SW Fwy., 809 Dallas St.; Holland Square Center, 10920 East Freeway, Champion Forest Plaza, Champion Forest Dr. and F.M. 1960 West, 1018 Gessner, 3278 South Loop West (So. Main at 610)
HURST Northeast Mall
IRVING 2011 West Airport Fwy
LAREDO 102 East Calton Rd.
LUBBOCK 3625 34th St.
ODESSA 1613 "A" East 8th Street
RICHARDSON Fleetwood Sq. S/C, 202 W. Campbell Rd.; 320 S. Central Expr.
SAN ANTONIO 6018 West Ave.; 4249 Centergate; Riverbend Parking Garage, 211 W. Market St. (Downtown)
WICHITA FALLS 1720-A 9th St.

UTAH
OGDEN K-Mart Shopping Center, 3672 Wall Ave.
OREM Grand Central Plaza, 384 East & 1300 South
MURRAY 6051 S. State Ave.
SALT LAKE CITY 301 South State St.

VIRGINIA
ALEXANDRIA 3425 King St. at Quaker Ln.
ARLINGTON Crystal City, 2301 So. Jefferson Davis Hwy.
FAIRFAX Westfair Center, 11027 Lee Hwy.
LYNCHBURG Hill's Plaza, Ward's Rd
NEWPORT NEWS Newmarket South Shop. Ctr.
NORFOLK 5731 Poplar Hall Dr.; Wards Corner, 122 E. Little Creek Rd.
RICHMOND Willow Lawn S/C, 1617 Willow Lawn Dr.; 7728 Midlothian Turnpike
ROANOKE Franklin Bldg., 3561 Franklin Rd. S.W.
ROSSLYN 1911 N. Ft. Myer Dr. at Rt. 29

WASHINGTON
BELLEVUE Crossroads Mall, North East 8th & 156 St.
BELLINGHAM 1111 Cornwall Ave., Suite B & C
FEDERAL WAY 33505 Pacific Hwy. South
OLYMPIA 106 N. Wilson
SEATTLE 18405 Aurora Ave. N., 1521 3rd Ave., 5030 Roosevelt Way NE
SPOKANE 7702 N. Division; E. 12412 Sprague
TACOMA 7030 S. Sprague
TUKWILA 15425 53rd Ave. S.
YAKIMA 1111 N. First St.

WEST VIRGINIA
DUNBAR, Dunbar Village Shop. Ctr
HUNTINGTON 2701½ 5th Ave.

WISCONSIN
APPLETON 2310 West College Ave.
MADISON 57 West Towne Mall
MILWAUKEE 6450 N. 76th St.; 729 N. Milwaukee (Downtown)
WEST ALLIS 2717 South 108th St.

Radio Shack

TRS-80 Microcomputer News
P.O. Box 2910
Fort Worth, Texas 76113-2910

BULK RATE
U.S. POSTAGE
PAID
Radio Shack
A Div. of Tandy Corp.

ADDRESS CHANGE
☐ Remove from list ☐ Change as shown
Please detach address label and mail to address shown above