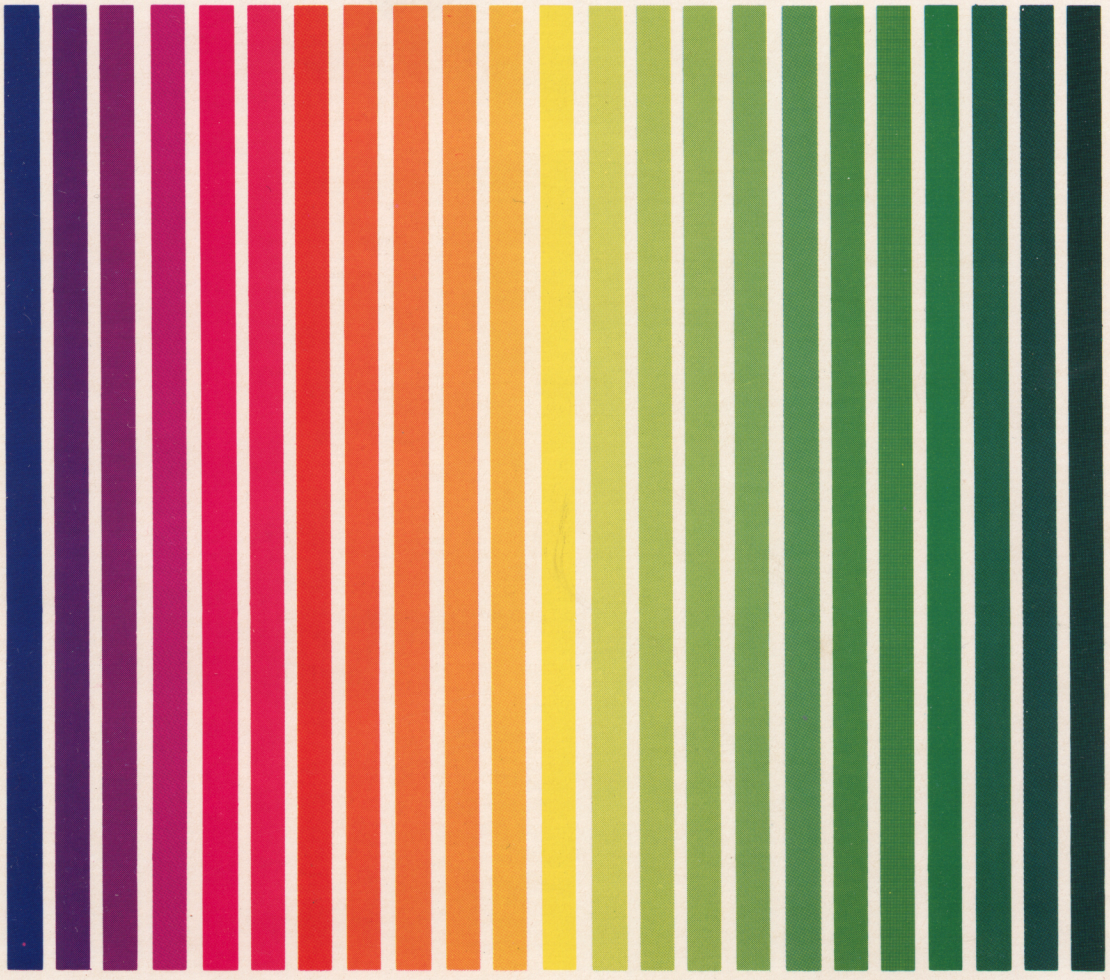
APX ATARI® PROGRAM EXCHANGE



Bob Fraser

SUPERSORT, Rev.3

A high-speed sorting subroutine
usable in BASIC programs

Diskette: 24K (APX-20030)

User-Written Software for ATARI Home Computers

Bob Fraser

SUPERSORT, Rev.3

A high-speed sorting subroutine
usable in BASIC programs

Diskette: 24K (APX-20030)

SUPERSORT

by

Bob Fraser

Distributed By

The ATARI Program Exchange
P.O. Box 3705
Santa Clara, CA 95055

To request an APX Product Catalog, write to the address above, or call toll-free:

800/538-1862 (outside California)
800/672-1850 (within California)

Or call our Sales number, 408/727-5603

Trademarks of Atari

The following are trademarks of Atari, Inc.

ATARI®
ATARI 400™ Home Computer
ATARI 800™ Home Computer
ATARI 410™ Program Recorder
ATARI 810™ Disk Drive
ATARI 820™ 40-Column Printer
ATARI 822™ Thermal Printer
ATARI 825™ 80-Column Printer
ATARI 830™ Acoustic Modem
ATARI 850™ Interface Module

Printed in U.S.A.

CONTENTS

LEGAL NOTICE

You are granted a license to use this program for your personal use only. If you want to incorporate SUPERSORT into software you plan to sell or otherwise distribute, please call the ATARI Program Exchange for license information.

INTRODUCTION

OVERVIEW

"LOOK! It's a bubblesort!"
"No!"
"It's a mergesort?"
"Wrong again! It's SUPERSORT!

SUPERSORT is an ultra high-speed sorting routine written in machine language. You incorporate it into your BASIC programs according to your needs. It takes up less than 1000 bytes of memory, yet it can sort a thousand 30-byte names in less than 3.9 seconds and a thousand 1-byte items in less than one second. SUPERSORT supports records as large as 256 bytes, and it can process as many as 10,000 records, depending on your memory size.

The diskette contains a number of files to meet several needs. You can't use SUPERSORT with DOS I because SUPERSORT uses an autoload feature. However, the diskette contains DOS II. The AUTORUN.SYS file on the SUPERSORT diskette is the autoload. Don't confuse this file with ATARI's normal AUTORUN.SYS, which loads the RS-232C handler. The SUPERSORT diskette also includes the assembler editor source code to allow you to modify the program to fit you needs. Other files on the diskette are mentioned in the Troubleshooting and Advanced Technical Information sections.

SUPERSORT combines C. Hoare's QUICKSORT with a standard insertion sort. For a more thorough discussion of SUPERSORT and sort routines in general, see Knuth, Donald E., The Art of Computer Programming, Vol. 3, 723 pp., Addison-Wesley, 1973. (Page 114 contains a discussion of C. Hoare's QUICKSORT.)

REQUIRED ACCESSORIES

16K RAM
ATARI 810 Disk Drive
ATARI BASIC Language Cartridge

OPTIONAL ACCESSORIES

ATARI Assembler Editor Cartridge

PRELIMINARY STEPS

Using SUPERSORT is easy. First, duplicate AUTORUN.SYS from the SUPERSORT diskette onto the diskette containing the BASIC program with data you want to sort. Next, add several lines of code to your BASIC program to set up SUPERSORT. (1) You add a calling routine specifying several parameters, such as record and number of keys, by which SUPERSORT sorts your data. (2) You store in one long string the items you want sorted. (3) Optionally, you code an output format for your sorted records. A calling routine appears in the Sample Applications section. Lines 1000-1100 set up the parameters to call SUPERSORT.

When you're ready to run your program, SUPERSORT loads itself into memory as an AUTORUN.SYS file when you boot up your diskette. It installs itself below BASIC in location $1D0A but remains invisible until your program calls it. To sort your data, run your BASIC program as usual. SUPERSORT sorts the entire string in ascending order by your specified key.

GETTING STARTED

1. Insert the ATARI BASIC Language Cartridge in the (Left Cartridge) slot of your computer.

2. Turn on your disk drive.

3. When the BUSY light goes out, open the disk drive door and insert the SUPERSORT diskette with the label in the lower right-hand corner nearest to you.

4. Turn on your computer and TV set.

5. SUPERSORT will install itself below BASIC in RAM when you boot the DOS. To verify its presence, either compare your FRE(0) memory before writing any code (you should be about 857 bytes short), or PEEK at location 7434, where you should find the number 104. If you can't verify the presence of SUPERSORT, turn off your computer and start again.

USING SUPERSORT

PREPARING YOUR BASIC PROGRAM

However you code SUPERSORT into your program, your calling routine must include seven parameters: (1) record length, (2) number of keys, (3) starting location of the SUPERSORT subroutine, (4) starting location of the sort string, (5) starting location of the last record in the sort string, (6) starting location of each sort key and (7) ending location of each sort key. A brief discussion of each parameter appears below, followed by suggestions for incorporating these variables into your calling routine.

Record length

This parameter specifies the fixed length of your records. POKE this length into location $600 (1536). (Line 1000 of the sample routine shows one way to code this.) All the records must be the same length, and corresponding data fields must be the same length across records. Make your data fixed-length by padding data shorter than its field length with trailing blanks. A data record is one set of fields to be sorted as a unit. For example, a record for a mailing list might comprise fields for a name, address, and zip code. Figure 1 shows a schematic of two records.

Figure 1. Sample Data Records

Any of these fields can also be the keys by which SUPERSORT sorts your records. (A key is the criteria used to sort records.) SUPERSORT can sort on any key, and it can sort on as many as 89 keys. In the mailing list example, suppose we want to sort by two keys, first by zip code and then by name. We would specify the zip code field as our primary key and the name field as our secondary key. SUPERSORT would therefore sort numerically by zip code and withinin zip code alphabetically by name. If we set the name field at 10 bytes, we would pad any name less than 10 bytes with trailing blanks. (Assuming zip codes are all 5 bytes long, padding isn't necessary in this field.) An example of how to pad a field is shown in line 710.

Number of keys

This parameter specifies the number of keys by which you want to sort your file. POKE this value into location $64C (1612). (Line 1010 of the sample routine shows one way to code this.)

Starting and ending locations of each sort key

You need to POKE the starting and ending character number within the string of each key by which you want to sort your data. POKE the starting location of your primary key into 1613, and the ending location into 1702. POKE the starting location of each additional key into 1614, 1615, and so on, and the ending location of each additional key into 1703, 1704, and so on. Lines 1020-1100 of the sample calling routine contain the POKEs. Characters are counted starting with 0 for the first character of a record, to a maximum specified by record length, which is POKEd at location 1536-1.

THE USR CALL

A call of SUPERSORT consists of three values:

```
JUNK = USR(7434,START,LAST)
```

Starting location of SUPERSORT

Code a USR function containing three values--starting location of SUPERSORT, starting location of the sort string, and starting location of the last record in the sort string. (Line 1270 of the sample routine shows one way to code this.) The first value, starting location of SUPERSORT, calls the program and is always the value 7434.

Starting location of sort string

This second value in the USR function tells SUPERSORT where to begin sorting. Use an ADR function to specify this value and place this code before the line of code for your USR function. (Line 1270 of the sample routine contains the ADR function.) For example, to sort string BUF$, the value ADR(BUF$) indicates its starting location.

Starting location of last record in sort string

This third value in the USR function lets SUPERSORT determine the end of the sort string. Be especially careful in specifying this number. Using the wrong number could cause SUPERSORT to continue sorting past your string and rearrange your program code or the Operating System RAM! Before using SUPERSORT, check that the starting location of the last record in the sort string is a valid value. You can do this by including the following two lines of BASIC code in your program before your USR call:

```
X=(LAST-START)/RLENG
IF X<>INT(X) THEN PRINT "ERROR IN ADDRESS CALCULATIONS":STOP
```

The formula to calculate the address of the starting location of the last record in the sort string is:

```
ADR(BUF$) + LEN(BUF$) - 2*RLENG
```

where ADR(BUF$) is the starting location of your sort string, LEN (BUF$) is the total

length of your sort string, and RLENG is your record length. You subtract two RLENGs because SUPERSORT requires you to include an empty record at the beginning and at the end of your sort string. Place this code before the line of code for your USR function. (Line 1210 of the sample routine contains the function.)

INCORPORATING THESE VARIABLES INTO YOUR CALLING ROUTINE

A calling routine appears in the Sample Applications section and also on the diskette (the file name is DEMO.BAS) so that you can try it out.

Remember to include an empty record at the beginning and the end of your sort string. Failure to start your sort string with an empty record causes the first real record to be overwritten, and so you'll lose data. Failure to end your sort string with an empty record causes SUPERSORT to overwrite the bytes following your last record--either you'll just lose some data or the whole program could crash.

FORMATTING YOUR SORT STRING IN YOUR BASIC PROGRAM

For SUPERSORT to sort your string correctly, you'll have to format it first. A field must be the same length across records. Therefore, all the values for a field will always be as long as the longest value for that field.

```
DATA "JOHNSON    121 OAK WAY     ANYTOWN"
DATA "SMITH      1310 MAIN ST.   ANYTOWN"
```

These strings each contain one record having three fields. JOHNSON and SMITH are the first field, padded with trailing blanks to lengthen the data to 10 bytes each. The second field in each record is padded to lengthen its data to 15 bytes each, and the third to 10 bytes each. To sort this string, we would first need to add an empty record at the beginning and at the end.

After SUPERSORT completes its sorting, it stores the results back in the original string. Therefore, save your original, unsorted data string somewhere else (in your program or in another file, depending on the size of your sort string) if you want to preserve it.

TROUBLESHOOTING

ERROR CODES AND MESSAGES

Because you create your own calling and formatting program, every BASIC error is possible. You'll need to track down these problems yourself. SUPERSORT does no error-checking and it contains no special error codes or messages. If you enter bad values, you could get one of three results. You might run SUPERSORT but get your data back unsorted. Or, with a more severe error, you might get back a jumbled sort. Or, in the worst case, the program will crash and you'll need to turn off your computer. To guard against such an event, save your BASIC program before calling SUPERSORT. Then, should something go wrong, simply reload your program and look over your variables before rerunning the program.

PROGRAM OPERATION LIMITATIONS AND WARNINGS

Using vs not using MEM.SAV

The SUPERSORT diskette contains the utility MEM.SAV, which prevents your program from being destroyed when you call DOS II and prevents SUPERSORT from being lost in RAM. However, MEM.SAV slows disk operations since it saves and reloads a portion of RAM each time you call DOS II, and it reserves 45 sectors on the diskette for its own use. You can omit this function by deleting the file on your disk (use the DOS II menu selection D to delete the file). You can reinstate it when you want to by using menu selection N (CREATE MEM.SAV). Remember, though, that whenever you omit MEM.SAV from a diskette, calling DOS will destroy SUPERSORT. If you later run your BASIC program calling SUPERSORT, the system will crash and you will have lost your BASIC program as well!

Pressing SYSTEM RESET or calling DOS

Pressing the SYSTEM RESET key has no ill effect on SUPERSORT.

If you call DOS and you haven't included MEM.SAV on your diskette, then DOS will overwrite SUPERSORT in RAM. If MEM.SAV is in effect, it reloads SUPERSORT once DOS has reloaded into RAM.

ADVANCED TECHNICAL INFORMATION

The following modifications are recommended for advanced hobbyists only. The first modification tends to slow down SUPERSORT.

USING DOUBLE-BYTE RECORDS

As written, SUPERSORT accepts only single-byte record lengths, even if you move TREC. However, you can force SUPERSORT to accept double-byte records since the record length (RLENG) is used only during additions. Because the addition is already double-byte (to support propagation), you need only change hi byte additions from ADC #0 to ADC RLENGHI.

SAVING MEMORY SPACE

SUPERSORT currently has no subroutines because parameter passing is time-consuming. However, if you need more memory, you can reduce the size of the routine as much as one-third to one-half by rewriting to include subroutines. You can also eliminate 155 bytes by removing the insertion sort (labels Q9 - L21A). If you choose to eliminate the insertion sort, you must assign label M to one and then reassemble the program. The partition size is normally set to 9.

SAMPLE APPLICATIONS

SAMPLE CALLING ROUTINE

Below is an example of a calling routine you could include in your BASIC program to set up the conditions for using SUPERSORT. This is the listing for the same calling routine on the diskette with the file name DEMO.BAS.

```
10 REM SUPERSORT REV 3.0 DEMO PROGRAM
20 REM D. YOCUM, 11/19/81
30 REM
40 REM THIS PROGRAM ALLOWS THE USER TO DEFINE THE NUMBER
50 REM OF FIELDS, FIELD SIZE AND NUMBER OF RECORDS FOR SORTING.
60 REM YOU THEN SPECIFY WHICH FIELDS TO SORT BY...
70 REM SUPERSORT DOES THE REST. ALL FIELDS ARE THE
80 REM SAME SIZE IN THIS EXAMPLE, THIS IS NOT
90 REM REQUIRED BY SUPERSORT, HOWEVER.
100 REM ************************************************
110 REM MAJOR VARIABLES:
120 REM BUF$     ... MAIN SORT BUFFER. ALL RECORDS ARE STORED HERE.
130 REM TEMP$    ... TEMP. INPUT STRING FOR FIELDS
140 REM BLANK$   ... FILLED WITH BLANKS. USED TO PAD FIELDS.
150 REM KEY()    ... HOLDS NUMBERS OF FIELDS TO SORT BY.
160 REM FIELDS   ... NUMBER OF FIELDS PER RECORD.
170 REM FLENG    ... NUMBER OF CHARS PER FIELD.
180 REM NREC     ... NUMBER OF RECORDS
190 REM RLENG    ... NUMBER OF CHARACTERS PER RECORD
200 REM NKEYS    ... NUMBER OF SORT KEYS IN ARRAY KEY()
210 REM BUFSIZE  ... TOTAL LENGTH OF BUF$
220 REM KEYSTART... STARTING LOCATION IN BUF$ OF A KEY.
230 REM C        ... POINTER TO START OF NEXT FIELD ON OUTPUT.
240 REM LAST     ... ADDRESS OF START OF LAST RECORD IN BUF$.
250 REM ************************************************
260 REM
270 REM ************************************************
280 REM INPUT FIELD NUMBER, SIZE AND RECORD COUNT
290 REM ************************************************
300 REM
310 PRINT CHR$(125);"   SUPERSORT REV 3.0 DEMO PROGRAM "
320 PRINT :PRINT
330 PRINT "HOW MANY FIELDS PER RECORD";
340 INPUT FIELDS
350 PRINT "HOW MANY CHARACTERS PER FIELD";
360 INPUT FLENG
370 RLENG=FIELDS*FLENG
380 PRINT "HOW MANY RECORDS TO INPUT";
390 INPUT NREC
400 REM
410 REM ************************************************
420 REM COMPUTE SIZE REQUIRED FOR BUF$ INCLUDING 2 BLANK RECORDS
```

```
430 REM DIMENSION STRINGS ACCORDINGLY
440 REM ************************************************************
450 REM
460 BUFSIZE=(NREC+2)*RLENG
470 DIM BUF$(BUFSIZE),TEMP$(FLENG),BLANK$(FLENG),KEY(FIELDS)
480 REM
490 REM BLANK$ IS FILLED WITH BLANKS FOR PADDING
500 REM
510 BUF$=""
520 FOR I=1 TO FLENG:BLANK$(I)=" ":NEXT I
530 REM
540 REM STICK A BLANK RECORD AT THE BEGINNING OF BUF$
550 REM
560 GOSUB 1470
570 REM
580 REM **************************
590 REM INPUT DATA FOR EACH RECORD
600 REM **************************
610 REM
620 FOR RECNT=1 TO NREC
630 PRINT
640 FOR FCNT=1 TO FIELDS
650 PRINT "INPUT RECORD ";RECNT;" FIELD ";FCNT
660 INPUT TEMP$
670 BUF$(LEN(BUF$)+1)=TEMP$
680 REM
690 REM PAD WITH BLANKS IF THIS FIELD IS TOO SHORT
700 REM
710 IF LEN(TEMP$)<FLENG THEN BUF$(LEN(BUF$)+1)=BLANK$(1,FLENG-LEN(TEMP$))
720 NEXT FCNT
730 NEXT RECNT
740 REM
750 REM STICK A BLANK RECORD AT THE END OF BUF$
760 REM
770 GOSUB 1470
780 REM
790 REM ************************************************************
800 REM INPUT THE PRECEDENCE OF THE FIELDS FOR SORTING
810 REM ************************************************************
820 REM
830 PRINT :PRINT
840 PRINT "SORT BY WHICH FIELD (1-";FIELDS;")";
850 FOR NKEYS=1 TO FIELDS
860 INPUT X
870 IF X=0 THEN POP :GOTO 930
880 KEY(NKEYS)=X
890 IF NKEYS=FIELDS THEN 920
900 ? :PRINT "IF THOSE FIELDS MATCH, WHICH SHOULD"
910 PRINT "I SORT BY NEXT (0 WHEN DONE)";
920 NEXT NKEYS
930 REM NKEYS NOW HAS NUMBER OF KEYS
940 NKEYS=NKEYS-1
```

```
950 REM
960 REM ***************************
970 REM SET UP SUPERSORT
980 REM ***************************
990 REM
1000 POKE 1536,RLENG:REM RECORD LENGTH
1010 POKE 1612,NKEYS:REM NUMBER OF KEYS
1020 FOR KCNT=1 TO NKEYS
1030 REM
1040 REM COMPUTE BEGINNING CHAR # OF KEY FIELD
1050 REM
1060 KEYSTART=(KEY(KCNT)-1)*FLENG
1070 POKE 1612+KCNT,KEYSTART:REM STARTING CHAR FOR KEY
1080 REM POKE ENDING CHAR# OF KEY
1090 POKE 1701+KCNT,KEYSTART+FLENG:REM ENDING CHAR FOR KEY
1100 NEXT KCNT
1110 REM ************************************
1120 REM DISPLAY UNSORTED RECORDS
1130 REM ************************************
1140 PRINT :PRINT
1150 PRINT "UNSORTED RECORDS"
1160 GOSUB 1520:REM PRINT FORMATTER
1170 REM
1180 REM COMPUTE STARTING ADDRESS OF LAST RECORD IN BUF$
1190 REM THIS SHOULD IMMEDIATELY PRECEED THE USR CALL.
1200 REM
1210 LAST=ADR(BUF$)+LEN(BUF$)-2*RLENG
1220 REM
1230 REM **********************
1240 REM CALL SUPERSORT!
1250 REM **********************
1260 REM
1270 JUNK=USR(7434,ADR(BUF$),LAST)
1280 REM
1290 REM **********************
1300 REM DISPLAY SORTED RECORDS
1310 REM **********************
1320 REM
1330 PRINT :PRINT :PRINT "SORTED RECORDS"
1340 GOSUB 1520:REM PRINT FORMATTER
1350 END
1360 REM
1370 REM ************************************
1380 REM SUBROUTINES
1390 REM ************************************
1400 REM
1410 REM THIS SUBROUTINE CREATES A BLANK RECORD
1420 REM IN THE SORT BUFFER. IT IS USED TO MAKE
1430 REM BOTH THE FIRST AND LAST BLANK RECORDS
1440 REM REQUIRED BY SUPERSORT.
1450 REM ************************************
1460 REM
```

```
1470 FOR I=1 TO FIELDS
1480 BUF$(LEN(BUF$)+1)=BLANK$
1490 NEXT I
1500 RETURN
1510 REM
1520 REM ************************************
1530 REM OUTPUT DISPLAY FORMATTER
1540 REM ************************************
1550 REM
1560 C=FLENG*FIELDS+1:REM SKIP FIRST BLANK RECORD
1570 FOR I=1 TO NREC
1580 PRINT
1590 FOR J=1 TO FIELDS
1600 PRINT BUF$(C,C+FLENG-1),
1610 C=C+FLENG
1620 NEXT J
1630 NEXT I
1640 RETURN
```

Limited Warranty on Media and Hardware Accessories. We, Atari, Inc., guarantee to you, the original retail purchaser, that the medium on which the APX program is recorded and any hardware accessories sold by APX are free from defects for thirty days from the date of purchase. Any applicable implied warranties, including warranties of merchantability and fitness for a particular purpose, are also limited to thirty days from the date of purchase. Some states don't allow limitations on a warranty's period, so this limitation might not apply to you. If you discover such a defect within the thirty-day period, call APX for a Return Authorization Number, and then return the product along with proof of purchase date to APX. We will repair or replace the product at our option.

You void this warranty if the APX product: (1) has been misused or shows signs of excessive wear; (2) has been damaged by use with non-ATARI Home Computer products; or (3) has been serviced or modified by anyone other than an Authorized ATARI Computer Service Center. Incidental and consequential damages are not covered by this warranty or by any implied warranty. Some states don't allow exclusion of incidental or consequential damages, so this exclusion might not apply to you.

Disclaimer of Warranty and Liability on Computer Programs. Most APX programs have been written by people not employed by Atari, Inc. The programs we select for APX offer something of value that we want to make available to ATARI Home Computer owners. To offer these programs to the widest number of people economically, we don't put APX products through rigorous testing. Therefore, APX products are sold "as is," and we do not guarantee them in any way. In particular, we make no warranty, express or implied, including warranties of merchantability and fitness for a particular purpose. We are not liable for any losses or damages of any kind that result from use of an APX product.

For the complete list of current APX programs, ask your ATARI retailer for the APX Product Catalog

APX ATARI® PROGRAM EXCHANGE
P.O. Box 3705
Santa Clara, CA 95055

Review Form

We're interested in your experiences with APX programs and documentation, both favorable and unfavorable. Many of our authors are eager to improve their programs if they know what you want. And, of course, we want to know about any bugs that slipped by us, so that the author can fix them. We also want to know whether our instructions are meeting your needs. You are our best source for suggesting improvements! Please help us by taking a moment to fill in this review sheet. Fold the sheet in thirds and seal it so that the address on the bottom of the back becomes the envelope front. Thank you for helping us!

1. Name and APX number of program.

2. If you have problems using the program, please describe them here.

3. What do you especially like about this program?

4. What do you think the program's weaknesses are?

5. How can the catalog description be more accurate or comprehensive?

6. On a scale of 1 to 10, 1 being "poor" and 10 being "excellent", please rate the following aspects of this program:

   _____ Easy to use
   _____ User-oriented (e.g., menus, prompts, clear language)
   _____ Enjoyable
   _____ Self-instructive
   _____ Useful (non-game programs)
   _____ Imaginative graphics and sound

7. Describe any technical errors you found in the user instructions (please give page numbers).

8. What did you especially like about the user instructions?

9. What revisions or additions would improve these instructions?

10. On a scale of 1 to 10, 1 representing "poor" and 10 representing "excellent", how would you rate the user instructions and why?

11. Other comments about the program or user instructions:

From

STAMP

ATARI Program Exchange
P.O. Box 3705
Santa Clara, CA 95055

[seal here]