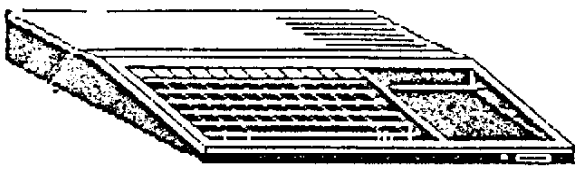

President Ira Lieberman 820-6332
Vice-pres Jeff Bleam 346-7590
Secretary Ann Halko 262-8206
Treasurer Barbara Rejician 767-9679

Vol. V, No. 7 November, 1987
Editor Jack Zawediuk 821-1043

LEHIGH 99'ER COMPUTER GROUP

Next meeting: 7:30 PM, Monday
December 21,1987

Conference Room A-D, Second Floor
Sacred Heart Hospital
4th and Chew Streets

PRESIDENT

If you have seen our meeting notices on the TV cable bulletin boards or the local newspapers, you can thank Jerry Boyer who made the contacts last month. He is setting up a regular mailing to let the media know we are alive and struggling.

Volunteers are still needed in other areas such as disk or cassette libraries, software evaluations or demos, hardware demos, etc. If these areas might be of intrest to you why not leave your name and phone number with our secretary and we'll get back to you. The clubs' computer should be back at the meeting this month and available for use. It will be kept by members who expect to attend regularly.

At our last meeting several new disks were added to the libraries including 10 new disk utilities. The librarians are going to try to prepare a list of our software. We also had some further updates on the mini expansion system.

Don't forget, members can use this newsletter for selling or swapping computer equipment. Just turn in typewritten copy or a TI Writer disk file at a meeting, or transmit the data to Editor Jack Zawediuk via modem.

Ira Lieberman

EDITOR

Every month our bulk mailing of newsletters pays off more. Among the eighty plus newsletters we received this month there were three from user groups we had not received from before.

The first letter is from the MOArk User Group of Springdale, Ark. It incuded reviews and comments. The second was from the Great Lakes Group in Roseville, MI. covering asorted articles and a swap corner. But the third new one is the best I've seen in a long time. It's the CHICAGO times. We received two issues, the super summer issue, a seventy five page book just packed with reviews, programs (in basic, assembly, c99, XB and pascal), articles on (high res graphics, the GENEVE, basic and XB programing), and on and on. There is even a Computer Shopper index for all TI articles in 1986. The September issue has forty five pages of the same great stuff.

As usual the packs of newsletters are avalible from the secretary for a five dollar deposit, refundable when you return them.

Jack Zawediuk

THE GENEVE IS HERE , FINALLY .

Part 1
by Jerry Boyer

Well, I finally received my new GENEVE, pronounced JIN-EVE. I had been calling around the country trying to find someone who had the GENEVE in stock, but every company had the same answer " We're only taking orders for the GENEVE and you would be about number 150 or so. We'll be shipping them out about 20 a week, which would mean about an 8 week delay." This wasn't good enough. I read an advertisement in MICROpendium from " INNOVATIVE PROGRAMMING ". They stated that they had the GENEVE in stock: I called them immediately (800)255-2985 . They said they had quite a few units in stock. I ordered it with the Enhanced Keyboard and 6 days later UPS delivered it, which seemed a miracle.

After reading the massive manual that came with it, and hooking it up as instructed, I soon discovered that nothing would work like the manual showed. I called Innovative Programming and talked to Galen A. Read, which was what MYARC had instructed me to do. Mr. Read was very curtious and attentive as I explained my problem. He said that the package must have been X-rayed and the software messed up. He stated that he would immediately send me another set, and I was to return the defective disks to them. Well, in 2 days they arrived in the mail. Needless to say, eyerything worked fine. Now that was what I'd call service. Mr. Read called me a day or so later to see how eyerything was working. I was pleased to see that he was very concerned about me and my problems. He showed what TI people were made of, no matter where you go .

The first thing you have to do is to transfer all of your modules to disk programs, much the same as Gram Kracker. In fact, any modules saved from Gram Kracker, will work on the Geneve. The Geneve doesn't have any provisions for plugging in your modules. What they supply is a program called Cartridge Saver, which runs on the TI keyboard. It is a very easy and very fast. I saved 40 modules to disks in less then 1 hour. I had the Widget cartridge expander which made it a little easier but the program works with out the Widget as well.

The ADVANCED BASIC that comes with the Geneve is only a copy of MYARC'S EXTENDED BASIC II set up to run in 80 column format with a few new commands thrown in. I had a lot of trouble running my extended basic programs. They would simply lock up the computer and I would have to turn it off and start all over again. This was remedied by loading my Extended Basic program (saved to disk) and then running the XB programs. No more lock up problems. MYARC will be sending out copies of ADVANCED BASIC when it is finished. The M-DOS and the MY-WORD programs are also not the final versions. They also will be sent out along with the PASCAL interpreter that I didn't receive.

This article was written with the MY-WORD word processor that's included. With 80 columns on screen all the time, it's a blessing to use. As I progress with my GENEVE , I'll try to find time to keep you all posted. By for now.

Reprinted from
SNUGLETter

FILE PROCESSING

File processing on the TI is not as difficult as you might believe. The hardest part for me was figuring out the "examples" that were in the owner's manual. They all went something like this:

```
100 OPEN #2:"CS1",INTERNAL,INPUT,FIXED
-
-
- program lines
-
-
290 CLOSE #2
300 END
```

This, in my opinion falls under the heading of "poor documentation". What was left out was the most important part! I tried and tried to get my computer to process files. I failed because I didn't know what to tell the computer to do with the files once it was open. I couldn't get past the mental block that told me "file processing is different from programming". In fact, programming is just a form of file processing.

The TI 99/4A handles ALL input and output through files. Most of the time, we are completely unaware that we are dealing with a "file" while programming. Page II-119 of the User's Reference Guide states "ALL TI BASIC statements which refer to files do so by means of a file number between 0 and 255 inclusive," "...file number 0 refers to the keyboard and screen of your computer and is always accessible...". Since file 0 is always accessible, statements such as PRINT, INPUT, RESTORE, etc. which refer to the keyboard or screen do require a file number with them. You can however, write a statement such as:

```
100 PRINT #0:"print this to screen"
```

and have it do exactly the same thing as:

```
100 PRINT "print this to screen"
```

You can also INPUT from file #0, but since file 0 is always open, statements like OPEN #0 or CLOSE #0 will generate an error message.

All other open files must be referred to by their number. Remember that this number is only used by the program to remember which file is which and is not a part of the file at all. As a matter of fact, you could open a file with one number, process it somehow, close it, and then reopen the same file with a different number...all this within the same program!

Now that I've got you thoroughly confused, I'll give you a short sample file processing program to try to clarify what I've been saying. Most of us think of a file as being a disk or cassette. While these are indeed files to the computer, they are by no means the only ones we have available. This short program opens a file to the Speech Synthesizer, sets up a FOR-NEXT loop to print a couple of sentences to both the screen and the Synthesizer, and then closes the file. You will need a TE-2 module to run the program. If you don't have a TE-2, just change the file name in line 110 from SPEECH to PIO or whatever your printer requires. This will give output to the screen and the printer instead.

```
100 CALL CLEAR
110 OPEN #1:"SPEECH",OUTPUT
120 FOR Y=1 TO 7
130 READ X$
140 FOR X=0 TO 1
150 PRINT #X:X$
160 NEXT X
170 NEXT Y
180 CLOSE #1
190 DATA THIS IS A TEST OF
THE SCREEN AND SPEECH FILES ON THE
200 DATA TEXAS INSTRUMENTS
99/4A HOME COMPUTER.  IT
SHOULD HELP
210 DATA TO DEMONSTRATE HOW
ALL INPUT AND OUTPUT IS
TREATED,AS A FILE BY THE
COMPUTER
```

In this program, line 110 OPENs a file to the speech synthesizer (or printer). Lines 120 to 140 set up some loops to read from the DATA statements and switch between files (0 and 1). Line 150 PRINTs the output to both outputs (0 and 1). Lines 160 and 170 increment the loops. Line 180 CLOSEs the computer's association with file #1, and lines 190 to 210 are the DATA read by line 130.

The point is that the lines between 110 and 180 are the ones that do all the work. Whether you are working with a file or just printing to the screen, the programming is the same. All you have to do is tell the computer where you want the data to go to or to come from.

Try modifying line 110 from OPEN #1:"SPEECH",OUTPUT to OPEN #1:"DSK1.TESTFILE",OUTPUT. This will cause the second output (remember that #0 is going to the screen) to go to a disk in drive #1 under the filename of "TESTFILE". Try some other experiments in line 110 like using "CS1"; "PIO", or "RS232" instead of "SPEECH". These will cause the output to go to the cassette recorder, printer, or modem respectively in addition to the screen.

Once you have mastered OUTPUTing to peripheral devices, the next logical step is to learn how to get INPUT from them. Some devices, such as the printer or speech synthesizer, by their very nature are one-way devices. Trying to get input from them would surely lead to hours of frustration. Keeping that in mind, we will concentrate on the devices that have two-way communication with the computer. The disk drive and cassette recorder are the primary devices we use for file storage. My experience with cassette based files has left me somewhat dissatisfied. While there are provisions for storing SEQUENTIAL files on cassette, it is a cumbersome operation as best. There also seems to be a bug in the I/O routines for input from cassette. If you do any file storage and retrieval from cassette, keep in mind that the delay between the prompt:

```
*PRESS CASSETTE PLAY CS1
 THEN PRESS ENTER
```

and the actual reading of data is longer in most cases than the tone leading to the data. I have found that if I press ENTER first, then wait for the screen to scroll up 1 line before pressing cassette play that I have no problems. If you don't do this the computer may miss the beginning of the file and give an error.

Since getting input from cassette and disk is very similar, I won't spend any more time on cassettes.

Getting input from a disk file is almost the same as sending output to it. First, you have to OPEN the file to the disk. This is done exactly the same as before, except instead of "OUTPUT" following the file name, we use "INPUT". The words INPUT and OUTPUT are two of the 4 modes that can be used to open a file. The third, UPDATE, is the default and means you can either read from it or write to it. If you don't specify one of the 4 modes, UPDATE will be assumed by the computer. The last mode is called APPEND and will only allow OUTPUT to the end of a file. Let's look at our program again. If you haven't already done so, change line 110 to OPEN #1:"DSK1.TESTFILE",OUTPUT and run the program. Type in the new program below (or modify the old one to match).

```
100 CALL CLEAR
110 OPEN #1:"SPEECH",OUTPUT
115 OPEN #2:"DSK1.TESTFILE",INPUT
120 FOR Y=1 TO 7
130 INPUT #2:X$
140 FOR X=0 TO 1
150 PRINT #X:X$
160 NEXT X
170 NEXT Y
180 CLOSE #1
190 CLOSE #2
```

The main differences between this program and the first one are that we have added a second file number and name to the program (line 115), changed the "READ X$ to "INPUT #2:X$, and deleted the data statements at the end of the program. We are now getting the data from the disk file that we just saved under the name of "TESTFILE", and #0 means the keyboard and screen. File #0 is an "UPDATE" file, #1 is an "OUTPUT" type file and #2 is an "INPUT" type file.

This has been very basic stuff so far, but in order to learn "FILE PROCESSING", you must understand the basics of how your TI-99/4A computer communicates with it's peripherals. Once you figure out that the computer treats EVERYTHING as a file, you will be on your way to writing your own file processing software.

(SNUGLETter - December 1986)

Reprinted from TISHUG

ARRAYS AND SORTS

by Jim Peterson

The concept of arrays, and especially of multi-dimensional arrays, is very difficult for many people to grasp. The following is the best explanation that I know of.

A variable name is a box in which you store some thing. When you write A$="X" you are telling the computer to "go to the box labeled A$ and put the character "X" in it". Or, more accurately, "go to the box labeled A$, throw away any- thing you find in it, and put "X" in it."

A simple array such as A$(3) is a row, labeled A$, of at least 3 boxes, labeled (1), (2), (3), and maybe more. When you tell the computer that A$(3)="X" you are again telling it to go to the row of boxes labeled A$, find the box labeled (3), and put "X" in it.

A 2-dimensional array such as A$(3,3) is a row, labeled A$, of at least 3 filing cabinets, labeled (1, and (2, and (3, and each having at least 3 drawers labeled 1) and 2) and 3). So, you can use A$(3,3)="X" to tell the computer to find the row of filing cabinets labeled A$, go to the one labeled (3, and open the drawer labeled 3) and put "X" in it.

And in a 3-dimensional array, A$(3,3,3)="X" tells the computer to find the A$ row of cabinets, find the one labeled (3 and find the drawer labeled ,3, and find the folder in that drawer labeled 3) and put.....

Finally, you can write A$(2,2,2,2,2,2,2)="X" to tell the computer to find row A$; cabinet (2 ; drawer ,2 ; folder ,2 ; paper 2, in the folder; line 2, on the paper; word 2, on the line; and letter 2) of the word!

Yes, TI Extended Basic can handle 7-dimensional arrays, but it is not very practical. Try running this - 100 DIM A(3,3,3,3,3,3,3) - and you will get MEMORY FULL IN LINE 100. Arrays with several dimensions are very wasteful of memory. I don't think I have ever seen a program that used more than a 4-dimensional array, and very rarely more than 3 dimensions.

Now then - A$(J)="X" means "go to the box labeled "J", find the number in it, then go to the row of boxes labeled A$ and find the box in that row which is labeled with that number....."

And even something as horrible-looking as A$(Y(J),Z(A,B))="X" just tells the computer to -

1. go to box J and find the number in it;

2. go to row of boxes Y and find the number in box number J of that row;

3. go to box A and find the number in it;

4.go to box B and find the number in it;

5. go to the row of filing cabinets labeled Z, find the one labeled with number A, open the drawer labeled with number B and find the number in it;

6. go to the row of filing cabinets labeled A$, find the one labeled with the number you found in Y(J), open the drawer labeled with the number you found in Z(A,B) and;

7. put the "X" in it!

Simple, isn't it?

Remember that, in a multidimensional array, only the last dimension holds the value; the others are just pointers to its location. A$(2,3)=A$(3,3) throws out whatever is in the 3rd drawer of the 2nd cabinet of the A$ row, and replaces it with whatever is in the 3rd drawer of the 3rd cabinet of that row, but the contents of the 3rd drawer of the 3rd cabinet are unchanged.

Also remember that box X or box X(1) or cabinet drawer X(1,1) or whatever, contain a 0 until you put something else in; box X$ or X$(1) or drawer X$(1,1) contain nothing at all until you put a string value into them. When you put something in the box, you throw away whatever was previously in the box. And to empty a box without putting anything in, you put a 0 in a numeric box or "" into a string box.

Enough, on that subject. Now, when you have all your data crammed into an array, the next thing you will probably need to do is to sort it into alphabetic or numeric sequence.

Sorting is one of the hardest jobs that you can give to a computer, and one of the things that a computer is the slowest at doing. Your TI can figure your bank balance in a split second, but might take half an hour to sort your mailing list.

Here's why. You can sort a bridge hand of 13 cards into sequence in 13 moves or less, by simply pulling out each card and slipping it back into its proper place. But, suppose those 13 cards were in 13 boxes, and you had to sort them without removing them from the boxes, except that you could hold one card in your hand? Even if you could figure out the best way, it would take you far more than 13 moves.

That is the problem that the computer has. You have just learned that the computer stores all those values in labeled boxes, or file drawers, and therefore must sort them by shuffling them from one box to another, emptying a box to shuffle into by holding one value in a temporary box while its value is compared with the others to find its proper place.

Of course, you could just set up a new row of empty boxes, and then search through the old boxes for the lowest value and move that to the first box in your new row, etc. - but that would double the amount of memory that the job would require. This would be no problem for a small array, but the computer can sort small arrays fast enough by the one-row method - it is the largest arrays that are too slow by the one-row method and would need too much memory by the two-row method.

Many ingenious routines have been written to accomplish these one-row sorts. I have written a program called "Sort Watcher" which enables you to actually watch various sorts taking place on the screen. It will also tell you the number of swaps and comparisons that were made.

This program demonstrates that the time required for a sort increases greatly as the size of the array increases. Sorting an array of 20 does not take just twice as long as sorting an array of 10 - it may take 4 times as long. For this reason, some of the faster and more complex sorting routines divide an array into smaller segments to be individually sorted and then merged.

After an array has been sorted, my program will also let you change any value in any part of the array, and then let you watch the array being resorted. From this, you will learn that a sorting routine which is very fast for a completely random array may be very slow for an array which is already almost in sequence!

In fact, to add just one additional value to a sorted array, the fastest method is the simple "shoehorn" - just set up an empty box at the end of the row, and move each value down by one box until you come to the proper place for the new value.

Continued on P10

A sorting routine can be either numeric or alphabetic depending on whether the variable names used are numeric or string. A numeric sort will be in strict numeric sequence and an alphabetic sort will be in ASCII sequence. That means that if all your strings are composed of upper case alphabetic characters, or all are lower case alphabetic characters, you will get an alphabetic sort - but if they are mixed, all of the upper case strings will come before any of the lower case strings, because the upper case ASCIIs are 65-90 and the lower case are 97-122. And if you have lower case words with capitalized initial letters...!

For the same reason, if you perform an alphabet sort of strings containing numeric digits, you will not get a numeric sequence - 10000 will come before 2 because 1 has a lower ASCII code than 2. It would be extremely difficult to devise a sorting routine which could sort numeric digits numerically within strings. However, if all the numbers are the same length, such as ZIP codes, the ASCII sort will be numeric.

Sorting a multidimension array becomes a very complex task. If you swap values around without also swapping all the related values, you will end up with complete garbage. Swapping all the related values takes time, and a dimensioned temporary variable name is also required.

Another way around this is to combine the data from an array into simple strings, or set it up originally as simple strings, and then perform a simple sort based on a specified segment of the string. For instance, you could use TI-Writer with tab settings to create a mailing list having first name at tab 1, second name at tab 15, address at tab 25, city at tab 45, state at tab 55 and zip code at tab 65. Then you could sort into last-name alphabetic sequence by sorting on SEG$(M$(J),10,255), or into zip code sequence by sorting on VAL(SEG$(M$(J),70,5)).

When using TI-Writer to set up such a file, be very sure to save it by PF with the C option, not by SF, and don't leave any blank lines at the end or elsewhere.

Alternatively, elements of data can be crammed into a string separated by control codes, and sorted by position of the code -

```
FOR J=1 TO 5 :: READ A$ :: M$=M$&CHR$(J)&A$ :: NEXT J
```

and then sort on element X by -

```
SEG$(M$(J),POS(M$(J),CHR$(X) ,1),255)
```

ANALYSIS OF SORTING ROUTINES

by Jim Peterson

Number of value changes made is shown above the number of value comparisons made. All sorts were made on the same portions of the same random array.

Number of records - 10 to 100

	10	20	30	40	50	60	70	80	90
BUBBLE	208	316	1018	1868	3218				
	52	182	450	805	1269				
SHAKER	109	311	1009	1855	3195				
	55	152	417	755	1172				
SWAP	52	130	207	286	354	457	539	642	
	54	209	464	819	1274	1829	2484	3239	
SHUTTLE	73	224	735	1360	2357				
	27	83	260	475	813				
EASY	102	323	611	979	1297				
	45	137	254	407	538				
QUICK	121	318	480	653	816	1032			
	108	292	440	613	764	969			
RESORT	43	120	317	552	911	1197			
	30	87	264	479	818	1084			
SHELL	35	109	206	351	557	633	691	857	1071
	30	92	150	226	364	422	485	581	683
WAZZIT?	59	184	345	578	775	1005			
	55	210	465	820	1275	1830			
INSERT	49	126	323	558	917	1203			
	21	68	235	440	769	1025			

Observations: the Wazzit? sort is one that I wrote, but I presume it has been done before under some other name. Some others of these may also be known under other names. The popular Bubble Sort is obviously the least efficient of them all, even for small arrays. The Quick Sort is not very quick. The Shell Sort is by far the best general-purpose sort when the file may be of any length and degree of randomness.

ANALYSIS OF SORT OF 5 DIFFERENT RANDOM ARRAYS OF 20 RECORDS BY 10 DIFFERENT SORTING ROUTINES.

- by Jim Peterson

Number of value changes is shown above number of value comparisons.

	1	2	3	4	5
BUBBLE	572	530	564	476	316
	204	174	209	165	182
SHAKER	467	433	455	379	311
	194	183	194	173	132
SWAP	129	117	132	123	131
	209	209	209	209	209
SHUTTLE	335	311	326	272	224
	121	113	118	99	83
EASY	318	334	371	392	323
	130	139	155	164	137
QUICK	298	298	267	290	318
	274	272	244	266	292
RESORT	155	151	154	140	120
	124	116	121	103	87
SHELL	142	118	139	127	109
	99	91	99	97	92
WAZZIT?	151	169	148	166	184
	210	210	210	210	210
INSERT	163	155	160	142	126
	105	97	102	84	68

OBSERVATIONS: The speed of a sort depends on the degree of randomness of the file, or by the distance that each record is from its correct position, but some sorting routines are less affected by this than others.

ANALYSIS OF RESORTING A SORTED ARRAY OF 50 RECORDS AFTER THE FIRST RECORD WAS CHANGED TO ZZZ OR THE LAST RECORD WAS CHANGED TO AAA OR THE CENTER RECORDS WERE CHANGED TO ZZZ AND AAA.

	ZZZ in 1st RECORD	AAA IN LAST RECORD	ZZZ/AAA IN MIDDLE
BUBBLE			
	199	295	242
	99	1275	950
SHAKER			
	202	204	201
	100	149	149
SWAP			
	246	246	243
	1274	1274	1274
SHUTTLE			
	148	148	148
	97	97	97
EASY			
	728	731	740
	342	345	347
QUICK			
	747	630	781
	686	638	759
RESORT			
	148	52	101
	98	98	98
SHELL			
	154	154	155
	270	271	272
WAZZIT?			
	198	198	199
	1275	1275	1275
INSERT			
	148	148	149
	49	49	49

OBSERVATIONS: The simple sorting routines may be better than the more complex and faster ones, for resorting a presorted file after a few changes have been made, or for adding a few records to an existing presorted file.

ANALYSIS OF ARRAY CAPACITY IN TI EXTENDED BASIC

by Jim Peterson

NUMERIC

NO. OF DIMENS	MAXIMUM DIMENSION ACCEPTED	TOTAL RECORDS	LEAVING BYTES FREE	BYTES PER RECORD	BYTES FREE AFTER RUN
1	3050	3050	64	8.07	20
2	54,54	2916	269	8.3	202
3	14,13,13	2366	944	9.95	852
4	7,6,6,6	1512	2511	14.535	2398
5	4,4,4,4,3	768	4459	26	4322
6	3,3,3,3,2,2	324	6023	56.99	5862
7	3,2,2,2,2,2,2	192	1123	121.69	938

STRINGS - MAXIMUM ACCEPTED BUT NOT RUNNABLE

1	5900	5900	28	2.002	
2	76,75	5700	124	2.055	
3	18,17,16	4896	198	2.377	
4	8,8,8,7	3583	160	3.259	
5	5,5,5,5,4	2000	1022	5.409	
6	4,3,3,3,3,3	972	1580	10.555	
7	3,3,3,2,2,2,2	432	1450	24.05	

STRINGS - MAXIMUM RUNNABLE FOR 1-BYTE RECORDS

1	1682	1682	8464	7.009	50
2	41,40	1640	8384	7.118	166
3	12,11,11	1452	8082	7.606	795
4	6,6,6,6	1296	7022	8.75	497
5	4,4,4,4,4	1024	5572	11.17	398
6	3,3,3,3,3,2	486	5676	17.81	3183
7	3,3,2,2,2,2,2	288	4042	32.295	2539

REMARKS - Any array of more than 3 dimensions will crash with MEMORY FULL if it is not DIMensioned, even A(1,1,1,1)=1.

String arrays can be dimensioned for many more records than they will actually hold. o

LEHIGH 99'ER COMPUTER GROUP
P.O. Box 4837 * 1501 Lehigh St.
Allentown, PA 18103