YESTERDAY'S NEWS

VOLUME 3 NUMBER 6 Established 2016 JUNE 2018

INSIDE INFORMATION

30 Years Ago...

Historical Information taken from Bill Gaskills TIMELINE

JUNE 1988:

DaTaBioTics announces that the long-awaited Grand RAM will begin shipping in quantity in July.

FunnelWeb v4.1 is released in the United States.

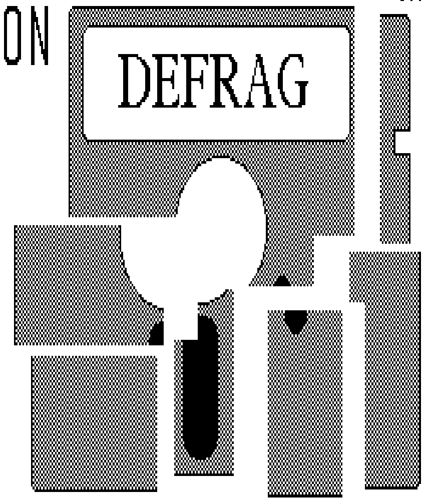
Word begins to get around the TI Community warning users to shy away from Galen Read's Innovative Programming company. It is reported in the Lima, Ohio User Group newsletter that Read has absconded with some $10,000 in orders for DataBioTics' Grand RAM.

Pilgrim's Pride, a long-time supplier and supporter of 99/4A products, closes it's Hatboro, Pennsylvania retail store. Mail-order business is retained.

Asgard Software begins publishing the ASGARD NEWS.

DEFRAGMENTATION

By Mark Schafer



In this article I aim to discuss the problem of fragmentation and what you can do about it. First of all, it's only a problem in the way that being messy is a problem; you can live with it, but it would be better if you didn't have it.

Now let's talk about what it is. It exists only on disks. But that's all kinds of disks: floppy disks, hard disks, ramdisks. The disk is divided into 256-byte units called sectors, which is the unit used to express the sizes of files in disk catalogs. They are numbered 0 to 359 on a single-sided, single density disk.

Each file reserves one whole sector that gives the computer everything it needs to know about the file. That sector is called the File Descriptor Record, or FDR. The item we are concerned with is the part that tells where on the disk the file contents reside. If it is split into more than one piece, it is fragmented.

To illustrate this, look at this newsletter. Think of each page as being a sector, and the articles as being files. If an article starts on page 1 and is continued on page 6, it is fragmented. But if it were continued on page 2, it's contiguous and therefore, not fragmented; it's still in one piece.

How newsletters and disks get to be fragmented are two different things. The TI disk controller reserves the first 32 sectors (after sectors 0 and 1 which are used for disk information) for FDR's. When a new file is added, it looks for the lowest sector available to put its FDR. Then it will put its contents starting at the lowest sector available outside of this area and continue writing until it bumps into a sector being used for another file. Then it keeps looking until it finds unused sectors to continue writing in. If it reaches the end of the disk, then it will look for some space in the FDR area. If there are more than 32 files on a disk, then some of the FDR's will not be in this area, and they could be located many sectors away from the first 32. This is because it is likely that the low sectors are already in use by the time the 33rd file is added.

When a file is deleted, its FDR and contents are then marked as free which could create "holes" of free sectors on the disk. When a file is rewritten to the disk, it will leave its FDR where it is, but it will still to try

See "DEFRAG", Page 2

ELEMENTS OF BASIC

By Dave Howell

Courtesy of the Erie 99'ers

Part 8

In last months column, an attempt was made to explain a computer's memory (RAM and ROM). RAM is the addressable memory. This means we can enter and temporarily store programs and data (information).

One of the ways we can enter data is by using the LET statement. RUN these programs:

```
10 LET A = 2             10 A = 2
20 LET B = 5             20 B = 5
30 LET C = 1             30 C = 1
40 LET D = A * B * C     40 D = A * B * C
50 PRINT D               50 PRINT D
60 END                   60 END
```

Both versions produce exactly the same output - with or without the LET statement. The use of LET is optional.

Think of the computer's memory as consisting of thousands of memory cells much like mail boxes in a post office. In the programs above, the computer stores the numbers 2, 5 and 1 in memory cells and labels them A, B and C respectively. The variables A, B and C can be thought of as addresses where the CPU can find them at some later point in a program. After the computer figures A * B * C to be 10 (5 x 2 x 1), the 10 is stored in yet another memory cell and labeled "D".

The same idea also holds true for "strings" as shown in the following program:

```
10 A$ = "SAM,"
20 B$ = " TOMORROW IS A HOLIDAY. "
30 C$ = "HAVE FUN!"
40 PRINT A$;B$;C$
50 END
```

The string character ($) must be used for any non-numeric characters such as letters, spaces and punctuation marks. Numbers can also be included within the quotes as strings provided no arithmetic is involved.

A shorter method of entering data into memory is the use of the READ...DATA statement. In re-writing the first LET program above, such a READ...DATA program might look like this:

```
10 READ A,B,C
20 D = A * B * C
30 PRINT D
40 DATA 2,5,1
50 END
```

The output of the LET and the READ...DATA programs will be identical as long as the data is in the same order. In a READ...DATA program, the DATA list uses a pointer to indicate which value within the list is to be assigned to the next variable in the READ statement. Before the first READ statement is encountered, the DATA list pointer will point at the first value in the DATA list. As values from the DATA list are assigned to variables in the READ statement, the pointer will move sequentially to each successive item in the DATA list.

The total number of variables in the READ statement in a program must match the total number of items in the DATA list. Otherwise, an error message will result.

The READ statement must always come before PRINT statements or calculations using the numeric data. On the other hand, DATA lists can be located anywhere in the program - even ahead of the READ statement!

These same rules hold true for programs using strings:

```
10 READ A$,B$,C$
20 DATA SAM," TOMORROW IS A HOLIDAY. "
30 PRINT A$;B$;C$
40 END
```

The strings in the DATA statements need not have quotes around them as they did in the LET statements unless spaces are required between words in a printout.

Numeric and string data can be mixed as in the following program:

```
10 READ A$,B$,C
20 DATA HAS," DOG "
30 READ D$,E$
40 DATA 110,MY,FLEAS
50 PRINT D$;B$;A$;C;E$
60 END
```

Notice that the values from the DATA list match the type of variable to which they were assigned in the READ statement. Also observe that the quotes around DOG includes a space on eather side. If these spaces were not included, the semi-colons in the PRINT statement would cause this print-out: MYDOGHAS. This is not the case with numeric data, however. Numbers will always be displayed on the TI-99/4A with spaces between them.

Enter and RUN this program:

```
10 READ M,N
20 PRINT M,N
30 GOTO 10
40 DATA 1,2,3,4
50 END
```

Since there are only 2 variables but 4 items in the DATA list, the GOTO statement sends the computer back to the READ statement to pick up the 3 and 4. When the GOTO statement sends the computer back to the READ statement the third time, the computer finds that the data has been exhausted. Since the READ statement can find no more data to "read" into memory, the computer prints an error message to that effect.

There are several ways this dilemma can be avoided. One of them includes the use of the RESTORE statement. If this line- 25 RESTORE -were inserted in the above program, the DATA list could be re-used on the next loop. Here are two more examples on the use of the RESTORE statement:

```
10 READ A,B,C         10 READ A,B
20 PRINT A;B;C        20 PRINT A;B
30 RESTORE            30 RESTORE
40 READ X,Y,Z         40 READ C,D,E,F
50 PRINT X;Y;Z        50 PRINT C;D;E;F
60 DATA 111,222,333   60 RESTORE 100
70 END                70 READ G,H
                      80 PRINT G;H
                      90 DATA 10,20
                      100 DATA 30,40
```

YN

DEFRAG continues...

to move its contents to the earliest sector available even if its size hasn't changed. So it could get moved into a part of the disk vacated by a deleted file.

So all of this can cause fragmentation. So what? Well, when you're reading an article, can you read it faster if it's contiguous or continued on another page? And then continued on another, then another, etc. It's the same with the disk drive. It can read a file faster if it's all together. Also, it can write a file faster if it doesn't get split up. The more pieces a file is in, the longer it will take to read or write it. The drive can find the next sector faster than it can find any other.

And this leads to my next point. All of this file adding and deleting can cause another problem. Unfortunately, I don't know of any single word that describes it. It is even more common than fragmentation. It occurs when the files on a disk are not in the ideal order.

The ideal order is the first FDR is on sector 2, the second FDR is on sector 3, the third FDR is on sector 4, and so on. Whenever the computer searches for a named file, it has to search the disk catalog. Just like newsletters, disks do not have indexes. But it does know where all the FDR's are because that information is stored in sector 1 in alphabetical order.

So it has to read in the first FDR, see if that's the right file, read in the next FDR, see if that's the right one, and so on until it either finds it or reaches the end of the catalog. This process would go MUCH faster if the disk were in the ideal order, which never happens except intentionally. Also cataloging the disk is much faster in ideal order for the same reason.

Luckily, you can take care of both fragmentation and less than ideal order at the same time. The cheapest, easiest to understand way to do this is to copy all the files from a disk onto a blank disk. All file copiers copy the files in alphabetical order, and there are no files on the blank disk to "bump" into.

Unluckily, this is not guaranteed to work. You could still have one problem or the other (but not both). Fragmentation could result if the disk has less than 32 files on it (the lower, the worse) and is full (or nearly full, but especially if it's full). One of the latter files could reach the end of the disk while it's being written and have to be continued after its FDR, which is way up near the beginning of the disk. You could get lucky if the file ends right when it reaches the end of the disk which is more likely to happen if there are short files at the end of the catalog. But if it will happen, there's nothing you can do to prevent it. Oh, you could copy all the larger files first, but then the catalog will not be in ideal order.

The other problem will definitely occur if there are more than 32 files on the disk. This is because when it reaches the 33rd file, all of the space reserved for FDR's is filled, and it will have to go all the way to the end of the disk to find the first free sector. Every file thereafter will get even worse. So the catalog will start in ideal order and then really go awry.

But you can do something to prevent this from happening, and you even know IF it will happen, so you can prepare for it. However, it can be time-consuming, especially if there are signficantly more than 32 files on the disk. To this you have to have a sector editor. What you do is mark sectors as being used starting with sector 34 on the blank disk for as many sectors as there files beyond 32. So if there are 38 files, mark sectors 34-39 as used (6 beyond 32). Then only copy the first 32 files. Then mark sector 34 as free. Then copy the next file; mark sector 35 as free; copy the next file, and so on until they're all copied. This method will keep all the FDR's together and in the right order.

You may notice that the problems occur if there are MORE or FEWER than 32 files. What if there's exactly 32 files? Hooray! If this is the case, you will encounter no problems!

But if you're talking, say, 36 files or more, this process is probably not worth the trouble. Truth be told, you can actually prevent the fragmentation problem by cleverly marking sectors as used beforehand, but it would be difficult for me to tell you HOW to do it, much less do it. Fortunately, there's another way to go.

Use a defragmenting program. Even though their cause is to defragment the disk, they will also put the disk in ideal order. Although there were already some on the market, I wrote my own anyway. I like to do things my way, and I didn't see other defragmenters doing everything I wanted. Yes, friends, even defragmenters have features.

I called my program simply, "Defragmenter", it is written in 100% assembly language and offers the following advantages over using a file copier:

1. It only requires one disk. It defragments the disk itself without having to have another disk to write on.
2. I don't remember what this one is.
3. It's guaranteed to work. File copiers don't always work for reasons discussed above.
4. It takes steps to prevent the problems from recurring.
5. I believe it's even faster.

Number 4, I believe, is a unique feature of Defragmenter. Refragmentation could occur if a file grows, thereby slamming into the file that follows it. But this won't happen if there is no file following it, so Defragmenter puts the file most likely to grow (so designated by the user) at the end of the disk.

If a file shrinks and another file is rewritten that follows it (but not immediately following), it will move into the space vacated by the shrinking file and could be split. This won't happen if only one file follows it, so Defragmenter puts the file most likely to shrink (so says the user) next to last.

If a file is added to a defragmented disk, its FDR will be put all the way at the end of the disk, a long ways away from its ideal position. So Defragmenter has the ability to reserve some space after the FDR's for future files to occupy. They will probably still not be in the best place, but they will be a lot closer, and won't slow down searches quite so much.

One advantage you might think a file copier would have is that after the process, you still have all the data on the original disk, so if there's any information in the unused sectors you want to see (like a deleted file), it's still there for you to look at. But Defragmenter has the ability to preserve the data in unused sectors to combat this advantage. So it will be possible to recover deleted files after the process (but not with an automatic file recoverer).

To be fair, file copiers would still have the following advantages:

1. Risk-free. If the process is interrupted, all the data is still intact and usable. You may lose data with Defragmenter (but no more than one sector), and the data you didn't lose may be hard to recover. Also, it's easier to place your confidence with a file copier even if you didn't write it.
2. You know, at any given moment, where in the process a file copier is. One has no idea how close Defragmenter is to finishing. Part of its speed gain is not having to update the screen as it goes.

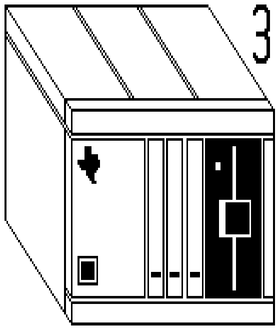
I had to dig pretty deep just to come up with these. The risk of using Defragmenter is minimal. How often have you had a power failure or a crash while copying files? Since Defragmenter is so fast, it's even less likely to happen to it. And number 2 is just a trade-off. If Defragmenter told you what it was doing, it would be slower.

Notice I didn't put "easier to use" on either list. That's because that is a matter of opinion and may depend on the file copier you use. Personally, I think Defragmenter is a joy to use, but beauty is often in the eye of the author. You tell it which disk to defragment, you answer four simple questions, all of which can be answered with a default answer with no ill effects, and it starts right up. It won't let you answer any of the questions wrong.

One advantage Defragmenter offers over anything else is that it comes with complete source code, as I believe all programs should. I'm offering it as fairware, but you are free to modify the source code to suit your own tastes as long as you don't give it out without my permission. The source code isn't documented, but there are some comments.

It comes with object code, source code, documentation, this article, and the source and object code to the cataloging program that I started with.

YN



3 SLOT EXPANSION KIT

By William M. Lucid

This kit gives the TI user a mini-expansion system that connects directly to the right side of the console, there is NO FAN required on this expansion kit.

Source of the Kit: Captain's Wheel - Farmington, MN

Cost of the Kit is $35.00 plus $5.00 shipping and handling. Disk power supply option is an additional $10.00. The Kit comes with all edge connectors (gold-plated) soldered in place, this saves the Kit builder from having to make 180+ delicate soldering connections.

All parts needed to complete the Kit are included, except an enclosure. Dimmensions are given in the documentation for constructing your own enclosure.

Captain's Wheel no longer sells three slot Kit enclosures.

Balance of the assembly consists of a few resistors, a few capacitors, two intergrated circuits and 30 jumpers, (another some 30 solder points). Wire provided with the Kit, is rainbow, muticonductor ribbon cable, that is stripped for jumpers, (insulation on this wire has a very low melting point). Solder traces on the one sided pc board are very fine line and closely spaced.

Documentation is complete with good illustrations. During assembly you proceed by checking off each step on the instructions. Verification check out explained in the documentation, this consists of using a voltmeter to verify voltages called out at points on the pc board. This is IMPORTANT to assure proper voltages are present, so NO DAMAGE results when hardware cards are installed into the system.

Captain's Wheel allows you to return the assembled Kit for check out verification and installation of cards at no charge; however, you must send $5.00 to cover the return cost of shipping and handling.

YN

CROSS COUNTRY CAR RALLY

By Steve Schwartz
99'er Vol 1 No. 6

I can say without hesitation that the best game I've seen so far for the **TI-99/4** is Cross Country Car Rally from Norton Software (Picton, ON, Canada). If you have Extended Basic, this is one fantastic game you won't want to pass up. It has everything you could possibly be looking for in a computer game...and much more! This arcade-quality game is relatively easy to learn, challenging to play, visually exciting...and it actually seems to become more fascinating the more you play it. Just when you think you've got the game mastered, some surprises are thrown at you that make you feel like you've just loaded it onto your computer for the very first time.

Basically, the scenario is this: You're driving a car from California to New Jersey and you have a limited amount of money to pay for traffic tickets and car repairs. You start off at the left side of your screen and accelerate (Keyboard input) until you're flowing with the other traffic. You'll have to do some weaving from lane to lane to get ahead as you slowly inch your way across the screen to the right-hand side (though it seems like you're going much faster due to the passing scenery).

If you're able to make it to the right-hand side --presto!-- the screen changes and you're now driving through Nevada at night. Utah (if you're able to get there) has different graphics and, presumably, every other state has its own unique graphics layout and its own type of traffic flow. For example, in California and Nevada, you're on a two-lane eastbound highway. But when you get to Utah, there's one lane eastbound and one lane westbound, so you'll have to pass with extreme caution or you'll wind up in a head-on collision for sure. (I'm afraid I can't tell you what the other states are like, because I've never been able to get across the state of Utah; but give me a break! ..I've only been at it for a few hours..)

To make things even more interesting, there are detours you have to take, and there are gravel-road "shortcuts" which, as in real life, often turn out to be a longer way of getting where you're going. Then there are the police cars that spot you speeding along and warn you to stop: You can try to outrun them (they'll start shooting at you if you don't stop), or you can stop and either pay the fine, or try to bribe the cop.

When you run out of money, the game comes to an end, and you're assigned a point value --ideal for competitive play or for trying to beat your old score. The game loaded from cassette the first time I tried, which was a pleasure in itself. To sum it all up, Cross Country Car Rally is the type of game that makes you wonder, "How did they manage to put such a big game into a program that runs on a 16K computer?" I don't know how they did it, but I'm very glad they did!

Yesterday's News Information

Yesterday's News is a labor of love offered as a source of pleasure & information for users of the TI-99/4A & Myarc 9640 computers.

TI-99/4A HARDWARE
Black & Silver computer
Modified PEB
WHT SCSI card with SCSI2SD
Myarc DSQD FDC
Myarc 512K Memory Card
Horizon 1.5 meg Ramdisk
TI RS232 card
Corcomp Triple Tech Card
1 360K 5.25 floppy drive
1 360K 3.50 floppy drive
1 720K 5.25 floppy drive
1 720K 3.50 floppy drive
80K Gram Kracker
Samsung Syncmaster 710mp

TI-99/4A SOFTWARE
PagePro 99
PagePro Composer
PagePro FX
PagePro Headline Maker
PagePro Gofer
TI Artist Plus
GIFMania

PC HARDWARE
Compaq Armada 7800 Notebook
Compaq Armadastation
Samsung Syncmaster 710mp

PC SOFTWARE
Dead,Dead,Dead Windows 98se
FileCap
prn2pbns
Irfanview
Adobe Distiller
Adobe Acrobat

Yesterday's News is composed entirely using a TI-99/4A computer system. It consists of 11 PagePro pages which are "printed" via RS232 to PC to be published as a PDF file.

Yesterday's News
c/o Sparkdrummer
AtariAge forum
Phoenix, AZ. 85027