SYNTAX 2.1

MAR APR 86

TABLE OF CONTENTS

FCAUG

FIRST CANADIAN ADAM USERS' GROUP
P.O. Box 547 Victoria Station
Westmount, P. Q.
H3Z 2Y6

POSTAGE PAID

Mtl. PQ Permit Pending

FROM: Joel R. Morris, Prince George, B.C.
SUBJECT: PowerPRINT -- A Review

I use the word processor of my ADAM a great deal in writing various papers and documents. I have never been happy with the inability of the ADAM to right justify or make changes easily within the text insofar as format is concerned. PowerPRINT (by Strategic Software) however, is the program that changes all this.

I should make known the disadvantages of PowerPRINT first. It is S L O W! Loading takes about 8 minutes and after you have loaded the program from SmartBASIC you then have to take a disk or datapack and, following the menu of the program, create a Smartwriter file within the PowerPRINT program. After this has been done you can start entering your text. So you must load your tape or disk with the file name on board and then start doing your typing (in the SmartWriter mode). After you have done with your documentation you have to store it in the loaded PowerPRINT file.

As if this isn't enough, in order to print within the PowerPRINT program, you have to reload SmartBASIC, PowerPRINT and then choose the option on the menu to print the document. You then reload the tape or disk with the documentation and have it printed. This must all be done on d1 since all parameters are set for d1 (or Disk 1).

Finally, after you have about lost your mind with all of this you get the final insult -- it takes 16 seconds to print each line of your documentation. I have overcome this by setting the whole printing process in motion and then going out to see what kind of day we are having and then joining friends for coffee! If my documentation isn't too long, I come back in half an hour and I have a relatively nice looking document that is somewhat professional in appearance.

The nice things that can be said for the program are these: There are 14 menu'd options in the print mode. Page length, left margin, right margin justifications; line widths, top margin; line spacing; page pause (to change paper -- two minutes); head and foot margins; page numbering and, finally, number of copies (which, in my copy does not work).

Within your documentation, you can (at the time of the typing of the document) use embedded commands which make format changes within the document. These include: Centre justify; right and left justify; line width; line spacing; page numbering on/off and start and end numbers; next page; daisy wheel change (two minutes) and headers and footers.

Now all of these things make for convenience and nice looking documents in the PowerPRINT mode BUT: when you return to the SmartWriter to print the document normally, guess what! All of these embedded commands show up and whoever might read that copy

would think you had lost your mind entirely.

I would hope that somebody would come along and make a program that could be done without all this fuss and muss. The time it takes to print a PowerPRINT document is much too long and, frankly, you can get lost in all the loading and unloading you have to do to get there. Strategic Software does send along an 11 page operation manual. The Cost? $25.00 U.S.

Dear Sirs:

I'd like to include a game review as my contribution to a fine newsletter. 2010:Text Adventure is probably Coleco's last game and it is also probably the hardest to find. But it's well worth it! The premise is simple: you are to board, repair, and send the Discovery back to earth before its decaying orbit drops it into oblivion over Jupiter. However, this is not as simple as it sounds. Every system aboard the Discovery is shut down or needs repair and it seems that for everything you fix, two new problems crop up. 2010, it should be noted, is not your typical text game. Unlike the fantastic games from Infocom, this game has no parser. Instead, almost everything is Smartkey driven so that you never have to type in whole words. To be perfectly honest, I hated this when I first started playing because I personally prefer to input my own commands and try different ideas, but once you get used to it, the game is actually quite enjoyable (not to mention extremely frustrating!) In playing the game, there are a few things to keep in mind and those of you who actually read game instructions will find this out. First of all, your character's movements are determined by the arrow keys. However, this is not as straight forward as it appears. Suppose you were headed port. Now, you press the "up" arrow. Instinctively, you'd expect to go FORWARD in the direction you were going, right? In this game, however, the arrows guide you with respect to the ship, so if you push the "up" arrow, you go FORE, regardless of which way you were headed before. Confusing? Not once you get used to it. Also, you should always remember to use the <return> key in each room. This is the key that allows you to "examine" rooms, equipment, etc. Finally, listen to the breathing sounds coming over the speaker. When the rate goes up, you'd better start looking for some oxygen! Not that this is the only way for your character to die. It seems that after only a little exploring, I start to get hungry, thirsty, sleepy...well, you get the picture! All in all, it's not a bad game and I'd highly recommend it...if you can get your hands on a copy!

Before I go, I was hoping that Syntax could print my name and address as I was hoping to get in touch with any doctors, medical students, nurses or other medical personnel who might have ADAMs. I myself am a graduating medical student and I was just curious as to what others in the medical field are doing with their ADAMs. If anyone is interested in corresponding with me, please feel free to write!

Thanks a lot! I look forward to more great things from FCAUG in the coming year!

Sincerely,
Stan Wong
13411-72 St.
Edmonton, ALTA
T5C 0R3

In SMARTBASIC, the memory location that stores the value for the size of the storage space on the tape when you INITIALIZE it is 25305. If the value of 160 is poked in, disks can be initialized to their proper directory size while in BASIC.

I discovered this location while I was trying to reset all the places in BASIC that contain the CALL needed to set up the VDP registers to the 32 column display that we have. I was hoping to be able to change the text mode to 40 columns and although I succeeded in getting the screen set to 40 columns I ran into a snag with the operating system routine that automatically calculates the offset. I am hoping that I can get some help from one of our members to reset the offset value and use the 40 column mode that is built into our VDP. If we can accomplish this it would be an idea for someone to tinker with the 64 column graphics mode and see if we can implement this directly from BASIC. (ed.'s note - we will discuss all aspects of the VDP in upcoming Syntax's. We hope that your queries will be answered to your satisfaction very soon.)
If you CALL memory location 16599, you start a system warm boot and Adam looks in the default drive for a HELLO file by placing the drive number first, and then CALLing this location. When you CALL this location you can run a HELLO file from any drive you want. This is useful if you want a surprising way to protect your program from hackers and others.

Don Nichol
Arkona, Ontario

Dear sirs,
I would like to pass on a quirk I have found with the data pack version of "DAMBUSTERS". If most of you are like me, you have probably never successfully bombed the dam despite weeks of effort. Is this a problem with the game or am I just a klutz? Anyway, here's how to see the "dam" sequence: When you remove the data pack before crashing, Adam tries to load from an empty data drive then shows a successful bomb drop, then crashes. If you make the bomb drop while close to proper position with an empty data drive, Adam shows the winning routine. Install the data pack before the routine is over and you can add your name to the hero list and continue play on the next level.

Dale Anderson
Carbon, Alta.

HI RESOLUTION SCREEN STORAGE ROUTINE

This routine is used to store a Hi Resolution Graphics screen in user memory. The routine is designed to store one graphics screen, 256 pixels wide by 159 pixels high. The ADAM computer defines this size of screen as a HGR mode screen. To allow sufficient space for the machine codes of this routine plus a memory buffer area, Lomem must be set to: LOMEM : 33500

The machine code routines necessary to read from Vram and to write to Vram are stored starting at address 27407 through to address 27480. They are as follows:

READ From Vram = 27407

WRITE To Vram = 27420

The storage area utilized by the machine code routines is as follows:

PATTERN Table Buffer = 28250 to 33367

Calling the Vram READ routine at 27407 will load the PATTERN table buffer from Vram location 8192 through to 13309. Note, however that the color table is not read and stored, instead the color table is initialized to a specific value (color) when the Vram WRITE routine is called.

Calling the Vram WRITE routine will take the contents of the PATTERN buffer (from user memory 28250 - 33367) and load it into the video RAM. The PATTERN buffer is loaded into the video RAM starting at location 8192 through to 13309. Finally, before the routine returns to the users main program the Vram color table is initialized to value 240 which is the same color as used by SmartBasic V1.0.

DISASSEMBLED MACHINE CODE ROUTINE

ADDRESS	CODE	MNEMONIC	REMARK	
27407	33 90 110	ld hl nn	nn = 28250	<---
27410	17 0 32	ld de nn	nn = 8192	READ
27413	1 253 19	ld bc nn	nn = 5117	ROUTINE
27416	205 26 224	call nn	nn = 57370	
27419	201	ret		<---
27420	33 90 110	ld hl nn	nn = 28250	<---
27423	17 0 32	ld de nn	nn = 8192	|
27426	1 253 19	ld bc nn	nn = 5117	
27429	205 0 224	call nn	nn = 57344	
27432	33 18 224	ld hl nn	nn = 57362	
27435	54 43	ld(hl) n	n = 43	WRITE
27437	33 120 130	ld hl nn	nn = 33400	ROUTINE
27440	54 240	ld(hl) n	n = 240	

```
27442       17  0   0       ld de nn        nn = 0
27445       1   253 19      ld bc nn        nn = 5117
27448       205 0   224     call nn         nn = 57344
27451       33  18  224     ld hl nn        nn = 57362     |
27454       54  0           ld(hl)          n  = 0      <---
27456       201             ret
```

REGISTERS

HL Starting address of storage buffers
DE Vram address
BC Number of bytes to transfer
CALL 27407 READ from Vram
CALL 27420 WRITE to Vram
PATTERN table buffer is at 28250 - 33367

The program below demonstrates the use of the HI-RES screen store routines. This program will produce a triangle on the ADAM's HGR mode screen. It then prompts the user to press any key to store the shape. After the shape has been stored the program again prompts the user to press any key to clear the screen. Finally after the screen is cleared the program prompts the user to press any key to re-show the triangle. The program subsequently recovers the triangle from user memory and displays it on the HGR screen.

```
  5  REM  HI-RES SCREEN STORE DEMO
 10  REM  Ted Bik  (1986)
 20  LOMEM :33500: HOME
 30  DATA 33,90,110,17,0,32,1,253,19,205,26,224,201
 40  DATA 33,90,110,17,0,32,1,253,19,205,0,224,33,
       18,224,54,43,33,120,130,54,240,17,0,0,1,253,19,
       205,0,224,33,18,224,54,0,201
 50  FOR x = 27407 TO 27456: &  Poke in machine routines
 60  READ d
 70  POKE x, d
 80  NEXT x
 90  HGR: HTAB 5: PRINT  "HI-RES SCREEN STORE DEMO"
100  HCOLOR = 13
110  HPLOT 130, 10 TO 250, 130 TO 10, 130 TO 130, 10
120  FOR w = 1 TO 2000: NEXT w: HOME: &  Wait Loop
130  PRINT  "PRESS ANY KEY TO STORE SCREEN"
140  GET g$: HOME
150  CALL 27407: &  Routine to store screen
160  PRINT  "PRESS ANY KEY TO CLEAR SCREEN"
170  GET g$
175  FOR w = 1 TO 1000: NEXT: &  Wait Loop
180  HGR: HOME
190  PRINT  "PRESS ANY KEY TO RESHOW SCREEN"
200  GET g$: HOME
210  CALL 27420: &  Routine to reshow screen
220  VTAB 20: HTAB 10: PRINT  "COMPLETED"
```

How many pages of text can I fit on a blank data pack? This is a question that we hear quite often. In order to answer this let us first go to the blank tape itself. The insert jacket or cover of every blank DDP says that the tape can "store approximately 250 typewritten pages of data or program". That should be the end of the problem - read the cover of your blank DDP. If the answer was so simple, however, there wouldn't be so many people asking this question.

Many people get the "no more room - use another tape or disk" message when they know quite well that they have a lot of room left on their tape. This can be verified in Basic when typing the "catalog" command. A blank tape gives a 253 Blocks Free message. Tapes that give no more room messages have anywhere between this number and 0 blocks free. (A block is equal to 1 K of information and that's about a page.) In many cases, it appears that the tape gets filled up long before the maximum limit is reached.

Believe it or not the tape is working correctly (or at least as it was designed to work). Your tape or system is not faulty and (God forbid!) Coleco did not lie to you on the insert jacket of your DDP. What Coleco doesn't make clear is that besides the 250 page limit there is another limit that you have to observe when saving files to your tape. That is, that the directory will only hold a maximum of 35 files.

In order to understand why this is so you have to understand a little bit about how Adam stores files to tape. A DDP has 256 blocks numbered from 0 to 255. The first block, 0, is called the boot block. A blank has little information here, but if this was a pre-programmed tape there would be a small program that loads the program from your tape to memory. The next block, block #1, is called the directory block. This is the block that is the source of our problems. The rest of the tape, blocks 2 to 255 are used to store your actual file information.

Let's take a quick look at the directory block. The directory block, like any other block, is 1024 bytes long. Within these bytes information is stored about each file that you have put on your tape. The information found here tells Adam what type of file it is, what it is called, where it is on the tape, in other words all the information that it needs to find and load the file that you request into memory. Each file needs 26 bytes in the directory block to contain all the necessary file information. Dividing the number of bytes per file into the total number of bytes, gives us the number of files entries per block. I hope that you can see that this is the same number of files per tape as well.

Working out the above formula you should get 39.38 (1024/26), This means that your directory block has room to store only 39 complete files and still has a few bytes left over. The space for four files is taken up by 4 permanent "resident" programs. These

are non-listable system files. There is a file entry for the boot block program and another for the directory block. The volume name (FIRST DIR) that shows up in Basic when you type catalog takes up another file entry. Last is the "Blocks Left" program that is involved in the estimation of how many blocks remain free when the catalog command is entered in Basic. After accounting for these four files, the user is left with space for 35 file entries.

Some of you may notice that even when you have less than 35 files and many blocks free, you can still get the dreaded "NO MORE ROOM" message. Look at your backup files. You must also include these files when you count up how many files you have in your file directory. If after you have counted up all these files and you still have the same problem, it is probably due to deleted files in the directory.

The file entry for a deleted file is not removed from the directory block but only tagged as deleted. This means that one of the file bytes is changed to represent a deletion for that file. Adam sees this and knows that this file name should not be displayed. But, and this is the important thing, the file entry stays in the directory block, taking up space. Therefore, when you delete a file remember that you are not really removing the file entry in the directory block even though the file itself may be written over and completely lost.

The blocks that the deleted file was stored on become available as soon as the file is deleted. Often this gives rise to another problem that you may have seen. The next file stored writes over the same blocks that the just deleted file occupied. If the new file is the same size or smaller it will occupy this space. And the new file will take up all the blocks that the old file previously took. If you catalog your tape in Basic, you will notice that a 1 block file can take up many more blocks than it should. This further decreases the storage capacity of your data pack.

To summarize then, your blank tape can hold a maximum of about 250 pages and/or 35 files, whichever comes first. Backup and deleted files are included in this file count. Blocks can be lost by storing a small file just after deleting a larger one.

Here is how to efficiently use your tape. Only store completed files on a special "finished version" tape. Keep your file sizes to a maximum of about 7 pages. Word processing files of this size are easier to work with and you should get to the maximum page size about the same time that you get to the maximum file number. This formula emphasizes what I mean: optimal file size = maximum pages per tape / maximum files per tape (250/35). The answer is 7.14. Finally, remember that when your directory block does get cluttered and you want to clean it out, use the INIT command in Basic. This will blank out your directory and allow you to start over again with a fresh tape.

SMARTBASIC 2.0:
A Review and Explanation

Finally! SmarttBASIC 2.0 is out. And why has it taken so long to appear you ask? It seems that this tape was collecting dust somewhere in Coleco (U.S.) before someone had the sense to get it out to the public. Here's the story in a nutshell. When Coleco canned Adam they canned all the software that wasn't yet on the market as well. Apparently, this tape had been finished for a long time and it was only the manual that was incomplete. A few pages before the manual was finished Coleco pulled the plug and this package had been in limbo ever since ... until, that is, Coleco (Canada) gave us a copy.
SmartBASIC 2.0 works just like 1.0, so I'll give you a breakdown of the changes and additions in regard to 1.0. There is a new commmand "MERGE" which takes 2 files and then joins them together. But by far the biggest attraction of 2.0 is its ability to use the memory expander thereby giving you about 90K of memory. Typing"EXTMEM" reboots Basic and no apparent change is seen. It's only when you write " ? FRE(0)" that you see the change. "STDMEM" reboots the tape and returns you to where you left off. If you don't have a memory expander and you type "EXTMEM" the screen pauses and then the cursor returns. Nothing happens. So if you want to take advantage of this new command you will have to get the 64K memory expander.

A nice change is with some of the editting keys. For example, no longer do you have to hit ^P to print the screen. The following command keys are now used: "Print" to print a screen, "Insert" to insert a space on a line, and "Delete" to delete a space on a line. These changes make Basic easier to use as the same keys are also used with the word processor and are thus familiar to most of us already. In addition the keys are conveniently placed and are activated with a single key stroke. So it's also easier to remember where the appopriate editting keys are as well.

You will also notice that whenever you get an error statement Adam will now beep. The graphics problem of bleeding between two adjacent lines has been looked at. The problem has not been rectified but it has been improved. (Thus the command chart on the next page should be read with this in mind.) There are other improvements such as having built sprite tables right into Basic thus making sprite access much easier. (Instead of having to write your own routines to access the sprites as we have been doing, these access routines are built right into the language now. Consequently it is easier to write programs that use sprites.) Finally the problem of spaces being inserted in between DATA (and REM) statements and the actual data after every rewrite to a file has been rectified.

We have provided a command chart on the next page that gives you all these and more changes. We suggest that you photocopy it and put it in your Basic book or wherever else so that you can have easy access to these commands.

(article concluded on back page)

NEW COMMANDS FOR SMARTBASIC II

COMMAND: EXTMEM
Accesses Adam's 64K memory expander. Gives you just over 90K of available memory space. (90646)

COMMAND: STDMEM
Returns ADAM to standard memory space. Gives you just over 26K of available memory space. (26391)

COMAND: MERGE
Merges two files together. (MERGE <FILENAME>, D#) Line numbers that are repeated are replaced by new file's numbers. New numbers will be put in their proper places in the old program.

COMMAND KEY: Print
Prints screen like ^P. Printing process can be stopped by pressing ^C.

COMMAND KEY: Delete
Deletes character over cursor & moves the rest of the line over one space to the left. Like ^O

Command Key: Insert
Inserts character into line where cursor is positioned. Like ^N.

SPECIAL NOTES

POKE: In SmartBasic II you can poke without restriction (above 54160).

ERROR STATEMENTS: Basic 2.0 beeps when error statements appear on screen.

LINE LENGTH: In Basic V1.0 you can have up to 127 characters in one line. Basic 2.0 allows up to 255 characters per line, which is about 8 lines of text.

BOOT: Basic 2.0 boots to any drive.

GRAPHICS: No bleeding of colors with 2.0

SPRITES: Basic 2.0 has a sprite table built in, which can be accessed in text as well as graphics modes. Sprite flag located at PEEK (16788). Sprite table located at PEEKS (16787) & (16786).

REM and DATA SPACING: Basic 2.0 puts only 1 space between REM & DATA statements & the info next to them. V1.0 puts 2 spaces.

This issue we're going to make a modification to the "SYSGEN.COM" file and look at some system control characters.

SYSGEN is a standard CP/M program used to transfer the system tracks (blocks 0 to 13) from one disk to another. (If you have tape substitute the word "tape" for "disk" everywhere that you see "disk" in this article.) By system tracks I mean the part of the disk that contains the actual CP/M operating system. So SYSGEN essentially performs like a very specialised COPY or PIP program.

Let's look at what SYSGEN does when you write SYSGEN from the A> prompt. First the program is loaded into memory starting at location 100H. You are then prompted to insert the disk that contains the system tracks that you want to copy into memory. If you press <return> the program ends - by the way, it's this particular aspect of SYSGEN that we will be modifying. But to continue with the explanation, and assuming you have entered the letter of a legitimate drive the system tracks will now be entered into memory just above where the SYSGEN program ends. After this is done you can either write the system to another disk or again choose to terminate the SYSGEN program.

So quite clearly it can be seen that SYSGEN allows you to do one thing - transfer system tracks from one disk to another. But what if you want to modify the system before writing it back to another disk? The system tracks are inaccessable to PIP, COPY, or DDT. Here's where we can thank Coleco again. You see, any other computer that uses CP/M has a SYSGEN that can allow the user, not only to transfer the system tracks but also to modify them.

When SYSGEN prompts for the source drive, the Coleco SYSGEN informs us that the system will reboot (ie the system will not be placed into memory above SYSGEN ready to be transferred and the SYSGEN program will terminate). What we have to do to get our SYSGEN in line with non-Coleco SYSGEN's. In order to use SYSGEN as a tool to modify the system tracks we must change this step. We will make the program skip this step and then go on to the write to destination disk step. By making this simple change we will then be able to use DDT to modify the system tracks. But I am getting ahead of myself just now, first I must explain how to use DDT in order to change SYSGEN.

Remember that all programs load at 100H in memory except for DDT. This is why DDT can modify programs - the DDT program and the program to be modified are placed in 2 different areas in memory. So let's modify SYSGEN. As in SYNTAX 1.5 (p.8) lower case is what you type, upper case is typed by ADAM, but because we are also entering ASCII code (changing the word "Reboot) ?" to "Skip) ?" we must use the "s" command. Adam gives the first number (current memory value, you must enter the second (new) value.

```
A>ddt sysgen.com
DDT V2.2
NEXT  PC
0800 0100
-a125
0125 nop
0126 .
-s26d
026D 52 53
026E 65 6b
026F 62 69
0270 6F 70
0271 6F 29
0272 74 20
0273 29 3f
0274 20 20
0275 3F 20
0276 20 .
-g0
A>era sysgen.com
A>save 8 sysgen.com
```

By entering nop at location 0125 we change the jp 0000 instruction to nop, nop, nop. Now if we press <return> we will not reboot the system but continue on to the next part of the SYSGEN program. The changes after this are "cosmetic" but still necessary as you will want the action of the commputer to correspond to what is shown on the screen. So we change reboot to skip.

Now we can learn how to modify the system. Start by loading SYSGEN and pressing "a" to load the system into memory. Now don't write the system anywhere but just hit <return> to get you out of the program. SYSGEN and the loaded system is still in memory, so save this area into a new file called CPM1.COM by typing "save 60 cpm1.com". This new file contains the 2 programs. Using DDT you can now modify the system part of this hybrid file. The system is in the last 13 blocks or at 900H to the end. We'll look at some changes that you can make next issue. For now we'll act as if you've just made your changes. With the SYSGEN occupying the first 2 blocks, the CPM1.COM performs just like a normal SYSGEN, except that the system is already loaded along with SYSGEN. So execute CPM1. Now you can use the skip change; when prompted for a source drive hit <return>. Now the program doesn't end but continues to the write to destination stage. You can now write your modified system to another disk.

Next we'll look at a few handy control characters. It's already known that hitting the <shift> and <undo> keys together will shut the smart keys off. This takes the keys off the bottom of the screen but also relieves the smart keys of their functions (ie I is now ^A, II is ^B, III is ^C, etc.). Hitting the <shift>-<undo> combination again will then restore the keys to their initial state. If you don't want to see the keys on the screen but you still want to be able to have them function for you, press ^V and then <return> when the keys are on the screen. The keys disappear but still function. On a similiar note, ^Y <return> acts as a smart key toggle off and ^W <return> or ^Z <return> toggle the keys on again.

If you want to write in upper case letters and numbers without switching back and forth with the <shift> key then press ^ and the up arrow and then <return>. Now all letters are in upper case yet the top row of keys (\1234567890-=^) are all lower case. <Shift>ing allows access to the upper case characters of the top row as usual.

Randomizing BASIC's RND Function

As mentioned in answering a letter in issue 1.2, SmartBASIC's RND function isn't truly random. Like all computer "random" functions it creates a pseudo-random sequence of numbers. The problem with SmartBASIC is that on booting, it starts the random number sequence at the same place each time. Therefore, when you have RND(seed) in your program, where seed is any number you want, you will get the same sequence of numbers every time you run it. To overcome this most Basics provide a RANDOMIZE function. What the RANDOMIZE function does is create an initial seed or parameter. This will allow the RND function to start in a "random" place in its sequence each time the RANDOMIZE function is called. Some Basics have an internal RANDOMIZE function, that is, each time a program is run a new initial seed is created automatically. This article will describe a few ways to create a RANDOMIZE function in SmartBASIC.

The answer to the above mentioned letter gave one technique of creating an initial seed. It involves creating an input polling loop which can be exited when a user presses a key or such. The loop updates the potential seed on each loop. As a result, the randomness of the seed is based on pressing a key at a random time. Below is an example of this technique.

```
100 FOR seed = 1 TO 5000
110 IF PDL(9) = 1 THEN 130
120 NEXT seed
130 seed = RND(-seed)
```

Another technique involves using the computer's clock. This involves a very short machine code routine. This routine takes a number out of the refresher register of the Z80 CPU and stores it at location 1056. The refresher register is incremented by the computer clock. Its use is internal to the Z80 so I won't go into details about it. The routine should be loaded at the beginning of the program. Below is the coding to do this.

```
100 FOR i = 0 to 5: READ d: POKE 1056+i,d: NEXT i
110 DATA 237,95,50,38,4,201: & RANDOMIZE routine
```

The routine is loaded in a safe place in the BASIC interpreter so no LOMEM or HIMEM statements are necessary. You use the routine as follows:

```
CALL 1056: x = RND(-PEEK(1062))
```

After executing the above line you can use the RND function as normal, but this time you can be confident of a "random" random number. One final note, I have found that by PEEKing location 17011 while in the text mode, you can get a random seed from 1 to 12. If you peek this location right after calling TEXT, it will always contain the same number. It seems that calling TEXT re-initializes this location. If you can replace TEXT with HOME then this location won't be re-initialized. For short applications it may be worthwhile skipping the machine code routine in favor of using this location for your initial seed.

Many text files that are sent between Adams have mysteriously added blank spaces inserted between words. If you compare the original with the copied version of the file, you'll notice that the blanks are inserted according to a discernable pattern. The blanks are always added after the last word on a line of a file created with SmartWriter. If you look a little closer, you'll discover that the additions are not present if the last character of the line does not have a return marker after it. This is all related to the W/P wrap around feature that allows you to type away without worrying about when you reach the end of a line. When Adam goes to store your file - chock full of automatically wrapped around sentences - it pads the end of the line with blank spaces.

It is this padding that you see when you transmit text files under normal conditions. When you use AdamLink 2 you are told to change 3 parameter settings - full duplex to half, auto wrap around from off to on, and auto line feed from off to on. For normal conversation (terminal mode) these settings are fine. However, when you want to send a text file to another Adam it is not. You must return the auto wrap around and auto line feed selections to off. This way the smaller horizontal line setting for the modem program does not interfere with the previously stored settings of the file you want to send or receive. Without auto wrap around and auto line feed, the blanks are not added and the new copy is perfectly reproduced when accessed under Smart-Writer. But be prepared for some odd effects on the screen while you are transmitting without the auto line feed. Some lines will appear to overwrite previously displayed lines. This is nothing to worry about. After the transmission, if you want to resume your conversation, return the 2 settings to on.

CP/M modem users have a unique problem to overcome when using text files. Many text files (especially documentation files) seem to be created with the "WordStar" word processing program. When you attempt to print them out to the screen or printer, you often get a lot of weird junk and inverse characters.

However, these text files are not junked. What we are seeing is what I'll refer to as the WordStar Syndrome. This is the result of attempting to PIP or TYPE a WordStar file to the screen or printer. The weird characters seem to be some sort of WordStar control codes or something. This explanation may not be terribly accurate, but the solution to this problem is. Simply take the problem file and PIP it to another file that you create with the [Z] option on. This option zeros out all the high order bits for each ASCII character. What this means is that after the new file is created you can type it out normally. Problem solved. An additional point to note is that this problem is not a CP/M modem problem per se. I only mention it in this context because as CP/M modem users, many of you will experience this effect in the day to day experience of downloading files and then attempting to print them out. Since the vast majority of you do not own WordStar, I think it's only fair to warn you of this problem.

SmartBASIC has the ability to "draw" figures in it's High resolution modes. This article will show how to use this feature.

Using either the DRAW or XDRAW commands will produce a shape at a specific location on the screen. The shape's color will be the current HCOLOR and the shape is referred to by a shape index number. The index number is 1 for the first shape in the shape table. A shape table holds the draw instructions and is placed in a reserved part of memory.

I'll give a detailed example and show how the shape table is organized. We will define a shape table with 3 entries, ie. 3 different shapes. First we need to reserve a part of memory to place the table in. A good place is in high memory at location 51456 since it gives us plenty of room to work with. Therefore we call HIMEM: 51455. We now want to poke the shape table bytes into memory. The first two bytes poked in are the number of shapes in the table. The number of shapes is assumed to be a 16 bit number which is split into 2 bytes in memory, least significant byte first. The next two bytes is the byte offset to the first shape list. We have 3 2-byte offset entries in the table, one for each shape list. Following the 3rd 2-byte offset is the bytes in the first shape list. A shape list ends with a zero byte. The next shape list follows directly after this zero byte. Each byte in a shape list can hold 3 draw codes (instructions). Draw codes are read right to left in a byte. The draw codes are as follows:

X00	move up
X01	move right
X10	move down
X11	move left

The X in each code may be a zero or a one. If a zero then the computer will move to the new pixel without plotting anything. If a one then the computer will plot before moving on. To place a 3-bit code in an 8-bit byte requires one of the codes to be only 2 bits long. The 3rd code (leftmost in the byte) has an assumed zero as its X. Since we don't always want to move up every 3 instructions, a 00 in the 3rd code means to ignore that code. This means we can not move up in the 3rd code of a byte. For example.
00 111 010 means to move down without plotting, plot and move left, and ignore the 3rd code.

Before using the shape table you must poke its starting address at locations 16766 and 16767. In our case we poke 51456 into these locations (after breaking it up into it's two byte components). You will find an accompanying program which uses our 3 entry shape table in this issue. The program will show a bird flying accross the screen. I used the XDRAW command to erase the shape after plotting. If you have SmartBASIC 2, you will find that XDRAW doesn't work as well in reDRAWing over the shape in the background color. There are also some problems with XDRAW in

SmartBASIC 1, but we'll discuss these and other problems later.

For more information on shape tables I refer you to your SmartBASIC manual (especially pages C-17 to C-21). I hope to include a shape table editor for next issue.

```
 90 &  SHAPE TABLE DEFINITION
100 HIMEM :51455
110 &  place shape bytes in reserved memory
120 FOR i = 0 TO 49: READ sb: POKE 51456+i, sb: NEXT
130 POKE 16766, 0: &  poke least significant byte of 51456
140 POKE 16767, 201: &  poke most significant byte of start of shape table
150 POKE 25471, 7: POKE 25431, 0: HGR: HCOLOR  = 3: SCALE  = 1
155 PRINT "  Press control C to quit"
160 k = 0: x = 20: y = 100
170 ft = 0: FOR i = 1 TO 3 STEP 1: &  cycle through 3 shapes
172 DRAW i AT x, y: &  put bird on screen
174 FOR j = 1 TO 50: NEXT: &  delay a bit
176 XDRAW i AT x, y: &  erase bird from screen
178 IF i = 3 AND ft = 0 THEN  ft = 1: i = 2
180 x = x+1: IF x > 230 THEN  x = 20
182 y = y+k-1: k = 2-k: &  make bird bob up and down
184 NEXT: &  next wing flap shape
186 GOTO 170
190 DATA   3,0: &  3 shapes in shape table
191 DATA   8,0: &  byte offset to shape 0
192 DATA   23,0: &  byte offset to shape 1
193 DATA   35,0: &  byte offset to shape 2
200 DATA   64,40,45,21,21,21,14,36,21,12,12,12,45,45,0: &  shape 1
210 DATA   45,45,45,45,46,192,21,45,45,45,45,0: &  shape 2
220 DATA   146,41,45,12,12,12,12,54,44,21,21,21,45,45,0: &  shape 3
```

FIGURE 1;EXPLANATION OF LINE 100

BYTE	DATA	EXPLANATION
0	1	Tells ADAM that there is one shape
1	0	Unused
2	4	tells ADAM that shape starts at byte 4
3	0	Tells ADAM that shape ends at DATA "0"
4	5	Plots one pixel tothe right
5	17	moves right one and down one
6	5	Same as 5 above
7	17	Same as 17 above
8	5	Same as 5 above
9	17	Same as 17 above
10	5	Same as 5 above
11	17	Same as 17 above
12	5	Same a. 5 above
13	0	Tells ADAM that this is the end of the first tape

(see the following article on the next page)

ADAM HI-RES Custom Shapes:Part 1

To make high resolution shapes on ADAM you must first draw your shape on graph paper, convert the shape into vectors (arrows indicating direction), code the vectors into bytes, take the hexadecimal values of the bytes and poke the hex values into ADAM's memory. The hexadecimal values poked into memory are instructions to ADAM on how to make the shape. Unfortunatley, we were never told which values when poked into memory did what. To find out what the different values do, I wrote a program that will test the unknown values with known values. I experimented with different values and came up with certain values that I was certain about. For example, I found that 5 plotted one pixel (picture element) or dot, to the right. I then made a program that would help me find out what the more difficult values did. I used the following program to help me.

```
10 HIMEM :51455
20 FOR t = 0 to 13
30 READ a: POKE 51455 + t, a: NEXT
40 POKE 16766, 0: POKE 16767, 201
50 HGR: HCOLOR = 6
60 SCALE = 10: ROT = 0
70 DRAW 1 AT 128,79
80 END
100 DATA 1,0,4,0,5,x,5,x,5,x,5,x,5,0
```

In line 10 the program sets a boundary where the program will not go beyond, this protects the memory beyond the boundary. Lines 20-30 READ the DATA and poke the data into the protected memory area. Line 40 tells the computer where the shape table location is. The format is low byte first high byte last. Line 50 sets the screen for hi-res graphics and sets the color of the shape to medium blue. Line 60 sets the size of the shape at 10 times the default size. If default was one pixel, then 10 times that amount would be 10 pixels (using SCALE = n where "n" is a whole number). Line 70 draws the shape (called SHAPE 1) at screen location 128,79 (the centre of the screen). Line 80 ENDs the program. Line 100 contains the data for the shape (the DATA statement can be anywhere in the program even after the END line). Line 100 contains the codes for the shape. When I say "shape" I mean, at this point, the figure that will be drawn on the screen with our known values and our unknown values. In our shape there are 14 values but only the last 10 define the actual shape, the first 4 apply to all shapes. The X's in line 100 stand for any value you are testing. What line 100 actually does and its explanation can be best described in chart form. Figure 1 explains line 100, figure 2 shows what it does on the screen (I have used the number 17 for X). And figure 3 is a chart of the codes for the shape table.

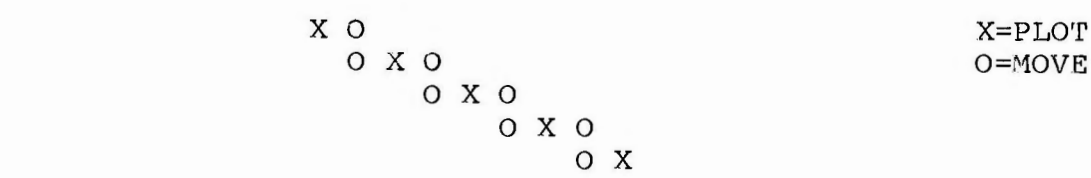
In part 2 I will show you how to make a shape using the information given here. Although I have tested the values of the codes for the shape table very carefully, I may have made a mistake or two. Any comments, suggestions and corrections should be sent to SYNTAX so that all members can be informed.

CODES FOR SHAPE TABLE
U-UP L-LEFT D-DOWN R-RIGHT P-PLOT M-MOVE

#	CODE	#	CODE	#	CODE	#	CODE	#	CODE
0	END SHAPE	51	ML/PD	102	PD/PU/MR	153	=2	205	=5
1	MR	52	PU/PD	103	PL/PU/MR	154	MD/ML/MD	206	=6
2	MD	53	PR/PD	104	MU/PR/MR	155	ML2/MD	207	=7
3	ML	54	PD2	105	MR/PR/MR	156	PU/ML/MD	208	=3
4	PU	55	PL/PD	106	MD/PR/MR	157	PR/ML/MD	209	=2
5	PR	56	MU/PL	107	ML/PR/MR	158	PD/ML/MD	210	ML/MD2
6	PD	57	MR/PL	108	PU/PR/MR	159	PL/ML/MD	211	ML2/MD
7	PL	58	MD/PL	109	PR2/MR	160	MU/PU/PD	212	PU/MD/ML
8	MU/MR	59	ML/PL	110	PD/PR/MR	161	MR/PU/PD	213	PR/MD/ML
9	MR2	60	PU/PL	111	=45	162	=6	214	PD/MD/ML
10	MD/MR	61	PR/PL	112	MU/PD/MR	163	ML/PU/PD	215	PL/MD/ML
11	ML/MR	62	PD/PL	113	MR/PD/MR	164	PU2/MD	216	MU/ML2
12	PU/MR	63	PL2	114	MD/PD/MR	165	PR/PU/PD	217	=3
13	PR/MR	64	MR/MU2	115	ML/PD/MR	166	=6	218	=11
14	PD/MR	65	MR2/MU	116	PU/PD/MR	167	PL/PU/PD	219	ML3
15	PL/MR	66	=1	117	PR/PD/MR	168	MU/PR/MD	220	PU/ML2
16	MU/MD	67	MU	118	PD2/MR	169	MR/PR/MD	221	PR/ML2
17	MR/MD	68	PU/MU/MR	119	PL/PD/MR	170	MD/PR/MD	222	PD/ML2
18	MD2	69	PR/MU/MR	120	MU/PL/MR	171	ML/PR/MD	223	PL/ML2
19	ML/MD	70	PD/MU/MR	121	=5	172	PU/PR/MD	224	MU/PU/ML
20	PU/MD	71	PL/MU/MR	122	MD/PL/MR	173	PR2/MD	225	MR/PU/ML
21	PR/MD	72	MU/MR2	123	ML/PL/MR	174	PD/PR/MD	226	MD/PU/ML
22	PD/MD	73	MR3	124	PU/PL/MR	175	PL/PR/MD	227	ML/PU/ML
23	PL/MD	74	MD/MR2	125	=5	176	MU/PD/MD	228	PU2/ML
24	MU/ML	75	=1	126	PD/PL/MR	177	MR/PD/MD	229	PR/PU/ML
25	MR/ML	76	PU/MR	127	PL2/MR	178	MD/PD/MD	230	PD/PU/ML
26	MD/ML	77	PR/MR	128	=67	179	ML/PD/MD	231	PL/PU/ML
27	ML2	78	PD/MR2	129	=1	180	PU/MD2	232	MU/PR/ML
28	PU/ML	79	PL/MR2	130	=2	181	PR/PD/MD	233	MR/PR/ML
29	PR/ML	80	=1	131	=3	182	PD2/MD	234	MD/PR/ML
30	PD/ML	81	MD/MR2	132	=4	183	PL/PD/MD	235	=2
31	PL/ML	82	MR/MR2	133	=5	184	MU/PL/MD	236	PU/PR/ML
32	MU/PU	83	=2	134	=6	185	MR/PL/MD	237	PR2/ML
33	MR/PU	84	PU/MD/MR	135	=7	186	MD/PL/MD	238	PD/PR/ML
34	MD/PU	85	PR/MD/MR	136	=1	187	ML/PL/MD	239	=7
35	ML/PU	86	PD/MD/MR	137	MR2/MD	188	PU/PL/MD	240	MU/PD/ML
36	PU2	87	PL/MD/MR	138	MD2/ IR	189	PR/PL/MD	241	MR/PD/ML
37	PR/PU	88	=67	139	=2	190	PD/PL/MD	242	MD/PD/ML
38	PD/PU	89	=1	140	PU/MR/MD	191	PL2/MD	243	ML/PD/ML
39	PL/PU	90	=2	141	PR/MR/MD	192	MU2/ML	244	PU/PD/ML
40	MU/PR	91	=3	142	PD/MR/MD	193	=67	245	PR/PD/ML
41	MR/PR	92	=4	143	PL/MR/MD	194	=3	246	PD2/ML
42	MD/PR	93	=5	144	=2	195	ML2/MU	247	PL/PD/ML
43	ML/PR	94	=6	145	=138	196	PU/ML/MU	248	MU/PL/ML
44	PU/PR	95	=7	146	MD3	197	PR/ML/MU	249	MR/PL/ML
45	PR2	96	MU/PU/MR	147	=19	198	PD/ML/MU	250	MD/PL/ML
46	PD/PR	97	MR/PU/MR	148	PU/MD2	199	PL/ML/MU	251	ML/PL/ML
47	PL/PR	98	MD/PU/MR	149	PR/MD2	200	=67	252	PU/PL/ML
49	MU/PD	99	ML/PU/MR	150	PD/MD2	201	=1	253	=31
50	MR/PD	100	PU2/MR	151	PL/MD2	202	=2	254	PD/PL/ML
51	MD/PD	101	PR/PU/MR	152	=3	203	=3	255	PL2/ML
						204	=4		

fig. 3 (see previous page)

fig.2:screen representation
one square = ten pixels in length (scale=10)



First I'd like to thank Sami Cokar of Calgary, Alta. for the above article and its accompanying figures. With this article and the one before it we've covered a lot of ground in regard to "shape tables". Sami, you're quite right in pointing out that the manual is vague on what the specific numbers do in terms of plotting points once they are poked in. As the article previous to yours pointed out, each number poked in actually represents 2 and sometimes 3 movements. So this number is actually a composite of the types of moves the programmer wishes to make. Your figure 3 is an attempt to list all the possible move combinations by decimal number. Bravo! As far as we can tell your figures are correct and we hope that this chart will help people with their shape table programming.

We've noticed a few minor problems with your submission that I thought I'd like to go through. First, number 0 is actually a move up. Only under certain condions is it an end shape (see article previous to yours). In your line 30 you start your first poke at the address that you set your HIMEM at, 51455. This must be a typo error because this would throw the data statements out of synch, thereby giving you rather bizarre resuts. They are sometimes more interesting to look at! But for predictable results you should change the number that you start poking at in line 30 to 51456.

SOME MORE LOCAL USERS' GROUPS

1) Kitimat Adam Users' Group
c/o Wendy Flegel
Kitimat, B.C.
V8C 1Z5

2) Edmonton Adam Users' Group
c/o Kevin Patzer
Edmonton, Alta
T6L 5K6

Dave McIntosh 7 Monsarrat Crescent, London, Ontario N5Y 4Y7 (1-519-679-0578) would like to hear from Adam users from anywhere.

And now the bad news. Most of the POKEs that we have told you about in this and past issues do not work in 2.0. Thus we have to first find the equivelent POKEs in 2.0 and then convert any programs that use these POKEs in 1.0 so that they'll run in 2.0. Explorations of this and some of the new things that I've touched on above will be seen in future editions of SYNTAX, so stay tuned.