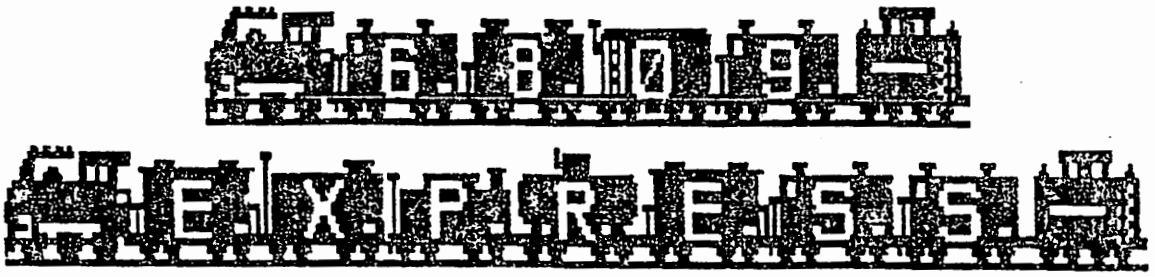


The Official Publication of The

PENN-JERSEY COLOR COMPUTER CLUB

MAVERICK BBS

215-760-0456

President's Report

This month we will be discussing telecommunications at our PJCCC get together. Since we are operating a club BBS, it is to all members advantage to know how to connect to the system. Two Coco's will be in operation, one as a host or the BBS and the other as a calling system. If you own a modem equipped Coco, be sure to attend! The BBS is the fastest way to get answers to Coco related questions or to obtain new programs without leaving the comforts of home.

Clyde Gano reported some progress with his contacts of the Glenside Coco Club in Illinois. We will be exchanging newsletters and info with this club. In reading the first received issue of the Glenside newsletter we see that their club is quite a lot like ours and I feel a relationship with both user's groups can only be beneficial to all involved!

Also a reminder to all club officers, and these days that's almost everyone. (heheh) Try to have your reports and newsletter items uploaded to the system by the 10th of the month so our editor can have the 6809 put together in time each month.

Sysop Report

Maverick has logged over 640 calls to the system. Currently the message base is small due to a recent squeezeing of the message disk. Our message base has passed some 340 messages since being initialized. Also some 200+ programs have been passed through the system.

Rick Hengevld - Sysop

IMPORTANT NEWS ABOUT

MAY 29 COCO CLUB MEETING

Richard Kravitz, our contact with the Northampton County Community College, just learned that our regular meeting room(s) at the college are being converted into lounge areas and will no longer be available to us.

The new room will be Room 190 in the College Center.

BUT ,the new room, like most of the school rooms, has no phone outlet and, realizing that we will need two outlets to properly demonstrate bulletin boards and modems at the next meeting, Richard investigated further and believes that there is a good chance that for this month's meeting construction might not have begun and if not, we can use them one more time.

If the old rooms are no longer available Al and Rick will come up with an other good idea for a program.

So, be aware of the problem, come to the meeting on May 29, and look for us first in the old room and if that is closed, use the map.

The Executive Committee

TREASURER'S REPORT

Statement date 4/27/92

Balance on hand 3/27/92		$219.55
Receipts:		
		0.00
Total receipts		$219.55
Disbursements:		
Bell of PA	ck. #265	$17.00
Clyde Gano (April stamps)	ck. #266	2.38
Total disbursed		$ 19.38
Balance on hand 4/27/92		$200.17

Clyde Gano,
Treasurer

Tech-tip POKE

To change to lower case from within a Basic program POKE 282,0. To change to all capitals, POKE 282,2. This works on Coco 1,2, & 3! It also works on the Coco 3's Hi-Res screens as well as the 32 column screens of all three Cocos.

The Library Car

For April 1992

Welcome once again to the PJCCC Library Car. It seems I ran amuck slightly last month. First there was an error that crept into the program listing due to technical problems. About two thirds down the listing there is a line with two PRINT statements seperated in the listing by a copyright symbol. That symbol should be a back or reverse slash. Secondly I made an error in assuming everyone knew how to get all the "strange" symbols used in the listing and in Basic09 in general.

I'll start this month by going over the keystrokes necessary to create these symbols. Please keep in mind that the keystrokes I am about to discuss are for OS9. Some may work in RSDOS or may require a totally different set of keys to accomplish the same thing.

■ Since we've already mentioned the backslash, this is made by using the control and slash keys. The slash key is in the lower right of the keyboard and the control key is on the extreme left. When using the control key, it is much like using the shift key. Press the control key and while holding it pressed, press the other key in the sequence, in this case the slash key.

■ Notice that in the second line of last month's program, there are square brackets around the 20. These are made with the sequence control eight and control nine. Noticed that these keys (eight and nine) when shifted give parentheses. When used with the control key they give the brackets.

■ I will try to remember from hence forth to give instructions for symbols that don't appear on the key faces when I introduce them. The backslash may not be seen in this column again as its only function in Basic09 is to seperate multiple statements on a single line.

As mentioned last month, this practice is frowned upon as single statement lines are easier to follow.

So now that we are done with the corrections and appologies, let's move on to the good stuff.

■ Last month I introduced a data type that may have been new to you. That was the integer. Remember it can store numbers in the range from -32768 to +32767 and it uses only two bytes to accomplish this. The computer can work with these numbers very quickly as the whole number can be loaded into a cpu register and worked on without having to juggle memory locations or make conversions to manipulate it. This month we will go on to discuss BYTE, REAL, and BOOLEAN data types.

■ To understand these data types properly we have to look at the way our computer is built. Don't worry. We won't be getting into this very deeply, just enough to understand what's going on. Our Cocos store data in memory in eight bit bytes. Bits are the smallest unit of data. They are either on or off, similar to light bulbs. If the bit is on it is usually represented by a 1. If it is off, a 0 will do nicely. If you've ever dabbled in binary math, this is the level on which bits work. A byte is eight of these bits taken as a unit. If you call a memory location, the computer retrives a byte's worth of data. A single byte is capable of

holding numbers from 0 to 255. Like an integer discussed earlier, a byte can be manipulated very quickly. In addition a byte takes up only one memory location. Thus if you have data that can be stored in a number 0 to 255, this makes for a very compact data storage package. Note that even an integer takes TWO bytes to store its data. I'm not going to go into it now, but there is a trick where you can store some kinds of information at the bit level. This allows you to store as much as eight different pieces of information in one byte.

■ We've already discussed an integer, so the next data package up is the REAL number. It takes five bytes to store a real number. How big a number do you think we can store in a real data type? Would you believe: 170,141,183,500,000,000,000,000,000,000,000,000 OR as small as .00000000000000000000000000000000000000293 8735877 !!! Suffice it to say these are larger and smaller than you are likely to need. A couple more things to remember about data type real. First if you notice, there are a lot of zeros in those numbers. That is because the computer actually stores the numbers in what is called scientific notation. This means that only so many actual digits are stored and the rest is stored as a note as to how many zeros go before or after the digits actully stored. In each of the numbers above, ten actual digits were stored and the rest of the number is zeroes! This means that if you exceed a number that can be stored in ten digits, some precision will be lost. This is usually not a problem, but must be accounted for if you do a comparision of two numbers and expect them to be the same. Due to the round off that occurrs in the larger numbers, what you get may not be what you expect. Since exactly what will happen will depend on exactly how the program is written, I can't anticipate how to correct a particular problem in advance. Just be aware that problems can come from this somewhat unexpected source. Real numbers are also the only numerical data type that can deal with decimal points.

■ I've got lots of memory so the fact that real numbers take five bytes doesn't bother me. Why shouldn't I use them for everything? How 'bout that it takes appoximately fives times longer to do a multiplication of real numbers than integers and division takes even longer!

■ The last data type we will discus this month is the boolean. Some of you may have heard of a branch of mathematics called boolean algebra. OOPs, I see some of you are turning the page. Get back here! We are not going to teach you some new math algebra. Boolean algebra is a way of figuring out a logical problem and ending up with either a true or false answer. A boolean data type is either true or false. If you ask your program to print the contents of a boolean type variable, it will print either the word TRUE or the word FALSE. Boolean data types are used to store the result of a test of some sort. An example would be: bool := A > B. If A is in fact greater than B then bool would contain the word TRUE.

See there wasn't any reason for you to be scared of boolean algebra now was there?

■ There is just one more thing we must touch on to make these items usable. As was mentioned last month, ALL variables MUST be declared before they can be used in Basic09. Similarly to RSDOS Basic, the DIM statement is used to declare the variables, but in Basic09 in addition to just declaring them we must declare the data type as well. The following lines are samples of how you could declare the types we've discussed this month.

■

■DIM age:BYTE
■DIM counter:INTEGER
■DIM bignumber:REAL
■DIM test:BOOLEAN

■

■ My plans for next month are to discuss strings, arrays, and creating your own data type. Yes, you read that correctly, your own data type!

You may be thinking, "Why do I need yet another data type?" Believe me once I show you what you can do with this, you'll wonder how you got along without it. Well, 'til next time, get your garden started, fertlize the lawn, tune up the mower, dig out the Bar-B-Que, clean the rain gutters, paint the house, do the spring cleaning, get the fishing gear in order, take the snow tires off the car, get the summer stuff out of mothballs, pack away the winter gear, enjoy the spring weather, and try to work in a little time to blow the dust off the COCO. Happy computing!

The Library Car

For May 1992

Welcome to the PJCCC library car. Last time we discussed several new data types, BYTE, INTEGER, REAL, and BOOLEAN. This time as promised we will be looking at a couple more, Strings, Arrays, and how you can arrange these data types in a data type of your own construction.

■ Strings are groups of characters stored in consecutive locations in memory. A string can be as short as one byte or as long as there is memory to hold it. There is NO limit as there is in RSDOS of 256 characters. Even though there is no limit to the length of a string, if you don't declare the length in the DIM statement, Basic09 will set aside just 32 bytes. If you try to enter more string data than there is memory set aside, the string will be truncated to the number of bytes set aside. Remember it takes one byte for each character. To DIMension a string you would type: DIM name:STRING[40]. Three things of note here. First, you don't NEED a dollar sign to declare a string. Second, note the type of symbols around the number. These are square brackets and are formed by the squence control 8 and control 9.

Third, the number inside those brackets can be any number from one to some number that taxes the amount of memory you can access in your computer! Remember back a couple of sessions when I spoke of how to assign memory to Basic09? It is this memory that is actually the limiting factor. Instead of assigning just a memory loacation when declaring a string, you can also declare the string you wish to store

there. To do this, type: name$:= "This is the string.". Note that this time the dollar sign IS required. We also are not really declaring a variable, but a name that is associated with that particular string location. The string length is determined by the number of characters between the quotes, not including the quotes. There are several string manipulating commands that we will cover as we get to working with strings. For those of you familiar with RSDOS Basic, they are very similar to the string manipulation commands found there.

■ The next item is the array. An array is not really a data type in quite the same sense as the previously discussed items. An array is a way of grouping a collection of like data in a form that can be accessed by an index. An array can be one, two, three or more dimensional. A one dimensional array is like a list with each line numbered. You can access any line mearly by using its number with the variable that names the array. The exact way that this is done will be shown once we discuss arrays a bit more. A two dimensional array can be thought of as a grid. This grid is numbered on each line and there is data stored at each intersection. A three dimensional array can be thought of as a cube divided into little boxes. The cube is so many boxes high, by so many wide by so many deep. Each box is numbered by its postion in each axis, as in third over, second up and fourth back. Each box of course contains data unique to itself. In arrays of more dimensions than three, you're on your own in the visualization.

■ Realize though that arrays have a way of eating memory exponentially. Suppose we wanted to declare an array of 10 strings 15 characters long. This might be done with a one dimensional array this way: DIM demo1(10):STRING[15]. This would require approximately 150 bytes to store the data. (Actually a little more is needed to store the array name and a few other facts the computer needs to know about the array.) Let's say that we now needed to store 5 more sets of this data. We could now go to a two dimensional array thusly: DIM demo2(10,5):STRING[15]. This now uses approximately 750 bytes. Now we want to store 10 more of these sets of data in a three dimensional array like this: DIM demo3(10,5,10):STRING[15]. This now uses 7,500+ bytes! Each one of these 500 strings contains 15 bytes of space whether they hold data or not. Now there is one problem with arrays. All of the data MUST be of the same type and size. It can be any of the data types discussed previously but each array can only hold one and only one type. This brings us to the next and neatest data "type". This is the user defined complex data type. For instance:

```
TYPE item=name,address(2):STRING[40];
zip:REAL.
```

This declares a data type "item" to contain a 40 character string called name, a one dimensional array called address containing two index locations each containing a 40 character string, and a variable of type real called zip.

Now we must declare some variable to be of type "item": DIM record:item. If we wish to access this data to print it out, we might do something like this:

■

■PRINT record.name
■PRINT record.address(1)
■PRINT record.address(2)
■PRINT record.zip

■

■Complex data types can contain other complex data types, but keep your wits about you if you try that one as the call to a particular piece of data can get quite complex! Complex data types can also contain any combination of data types and/or arrays of those data types in any combination your program should require. Complex data type can be used in arrays. Suppose we want to use an array of items called address_file. We could say: DIM address_file(50):item. This would create a one dimensional array of data type item that was 50 items long. To access the name in the fifth record, we could make a call this way: PRINT address_file(5).name. To access the second address index in that same record: PRINT address_file(5).address(2).

■ There is just one more thing to remember when DIMensioning data types. Simply setting aside memory for a data type DOES NOT initialize its value! What this means is that you can't assume a variable will be zero the first time it is used. If some other program has used the location now set aside for your variable, it may contain garbage. More than one programer has assumed a zero or null to be in a variable only to be tripped up by garbage left over from some previous computations. This means that before a variable is checked for the first time, a known value should be placed in the variable. For example: address_file(5).name := "". This sets address_file #5 variable name to a null string.

■Now that we have covered the basic data types, it is time to start applying them. I had asked for suggestions as to what kind of program those in the club would like to see. Pete suggested a desktop publishing program but I think this is a little complex to use as a beginning tutorial. Clyde suggested an inventory type of data base program. One that you could use to inventory household items for insurance purposes. If you're thinking ahead of me a little, you might see some possibilities with the complex data type just discussed to store the data. Since we are going to write a program, let's discuss what is involved in preparing for such an undertaking. There are five fundamental steps to creating a program. They are:

■

■1. Define the problem.
■2. Design a solution.
■3. Write the program.
■4. Compile(if the language requires it), debug, and test the program.
■5. Document the program.

■

■To be sure the actual execution of these steps are not usually as sharply defined as the above list suggests. For instance, if you wait until the program is actually

working to begin the documentation, you probably won't want to be bothered with what at that time seems extraneous. BUT, six months from now when you decide that the routine to gather the data isn't working out as well as you had hoped and you decide to tweek it a bit, you'd give your eye teeth to have good documentation. You decide that your program has wide enough appeal that you want to give it away (public domain) or maybe even sell it. How are you going to tell someone else that you have to load the print module before you can add that new data. You do it out of habit, but noone else even knows that there is a print module.

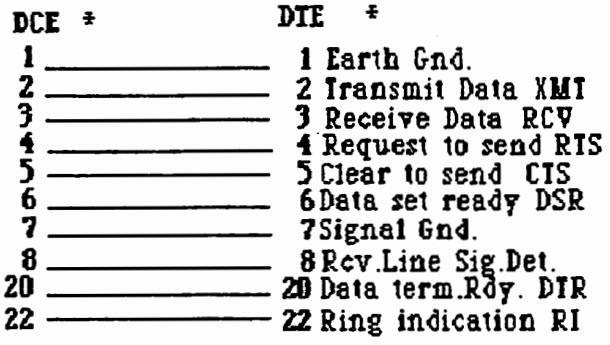
■ Let's try to define our problem. To define the problem we have to look at three parts of the problem: output, input, and the processing that gets us from one to the other. The output is what we need the program to give us, perhaps in the form of a printed page or maybe just a presentation on the computer screen. The input is what we have to give the program to work with. The processing may be the storage, sorting, and recall routines.

■ The output is often considered first as it governs what we need to collect and how we have to manipulate it to get the output desired. What do we need in our output? We need to name the item, a description (mfgr, model No., serial No.), guesstimate as to the value, possibly date of purchase, where purchased, how much, on credit, which card, where in the house (world) does this item normally reside. It might be nice to be able to output this to a printer in a format that fits on a 3x5 card so a card file can be made for the insurance company or anyone else that may not have access to a COCO, as well as output to the screen and to standard printer paper. The outputs to the screen and standard paper can be a variation of the 3x5 format or just take the 3x5 and put it to the different medium. Maybe we want to search the file by location, perhaps by value in excess of some number. We definitely need to be able to search by name, such as, camera 35mm.

■ How do we want to do the input? We could use the same screen layout as for the output. This means that we have to design the routine to be a fill-in-the-blank style screen. For input, the keyboard does the filling, for output, the computer does the filling from the file. What to input is easy as we have already decided what we need out, so that is what needs to be put in.

■ What processing needs to be done? Do we want this menu driven? Not a bad idea as menu driven programs are usually easier to run if you're into it for the first time or haven't used it for a while. We need to create a file to store the main bulk of the data. We need to check if this file exists, create it if it doesn't and access it for additions or corrections if it does. We need some kind of sort so the data comes out in logical order. Perhaps we need to make several index files so that we don't need to create a different sorted file of ALL the data for each access mode. This file should contain only the

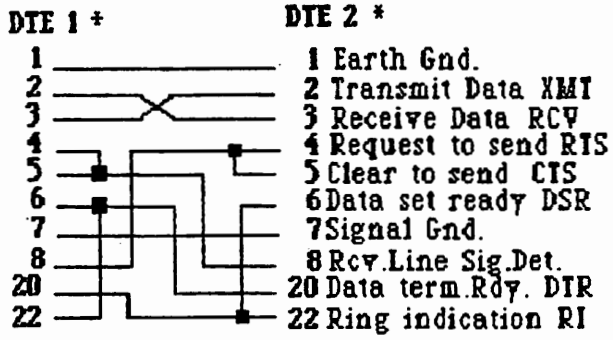
Technical Corner



*DTE= Data Terminal Equipment
DCE= Data Communications Equip.

Standard RS-232-C

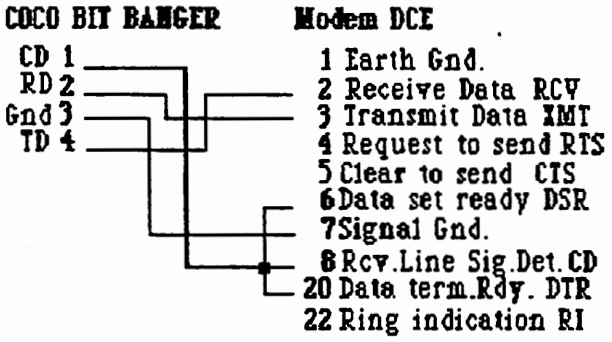
The above is how the RS-232-C standard was originally conceived. It was meant to go from data termial equipment (a computer or terminal) to a piece of comunications equipment such as a modem.



*DTE= Data Terminal Equipment
DCE= Data Communications Equip.

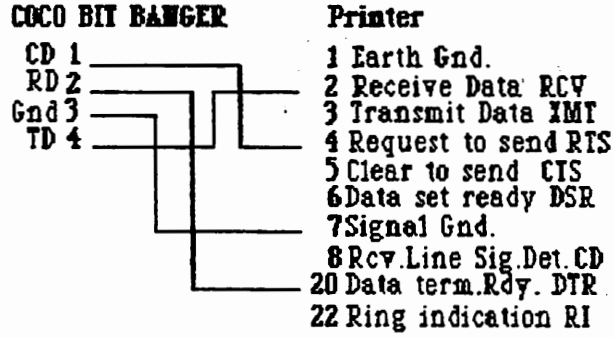
Standard RS-232-C Null Modem

The standard null modem is used between two pieces of data terminal equipment such as two computers having a DB25 connector outlet for serial communications



Coco Bit Banger Port
RS-232 Modem Cable

The above diagram shows a modem cable for the Coco's 4 pin DIN connector. Note that both the standard null and the cable for the Coco have some pins tied together. This is to cause signals to appear with fewer wires and for equipment that might not generate all the signals required for the RS-232 standard, as in the case of the Coco bit banger port.



Coco Bit Banger Port
RS-232 Printer Cable

The above is the cable for a Coco bit banger port to a printer serial port. Note that Coco pin 1 is connected to RTS on the printer and pin 2 on the Coco is connected DTR on the printer. This is because of the way a Coco senses printer busy.

record numbers in the order required for this search method and maybe some kind of notes as to where the breaks in the file are for the various sorts, i.e, where the end of the "A"s is. It should probably contain some kind of flag to denote if the index file has been updated since the last update of the main file. Maybe it would be better to include this in the main file as this file will be open whenever a sort is done and it would be open whenever an update is done. Above all, through all this we need to be considering error trapping.

■ Think about solutions to the above. You don't have to actually write code for the solutions. As a matter of fact, you don't even want to. Just write out in english the steps you think you'll need to get the job done. This is what the second step is all about. I'll leave you here. Have a good month and happy computing.

Due to the recent health problems of Peter Unks, I (Al Wagner) and Clyde Gano undertook to get out the 6809 Express. I would like to thank Clyde for his support both physical and the mental problem solving that made this issue possible. The experience has made me appreciate all the more the effort that Pete puts into our club news letter each month. I want to take this space to tell Pete we all wish him a speedy recovery and look forward to his more professional efforts at publishing The 6809 Express. I wish also to appologize for the lack of pictures. Please feel free to use the blank spaces for notes, scribbling, or any other purpose you think it deserves.

Tech-tip PEEK

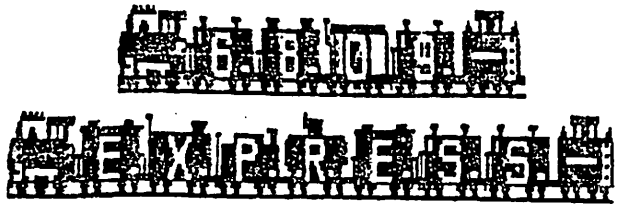
Did you ever wonder how to find out, from within a Basic program, if the arrow, ALT, CTRL, F1, and F2 keys have been pressed. One good way to do this is to PEEK at the keyboard rollover table. Though the table is larger than we will cover here, memory locations 341 through 344 contain the information we seek.

	191	247
341	ALT	up arrow
342	CTRL	down arrow
343	F1	left arrow
344	F2	right arrow

To check for the F1 key,

```
IF PEEK(343)=191 THEN 200
```

or something similar. This would mean that if the F1 key were pressed, the program would goto lin 200 and continue execution there You may find that you need to slow the keyboard response. If you do, experiment with a FOR/NEXT delay loop to find an apropriate delay.



The Official Publication of The

PENN-JERSEY COLOR COMPUTER CLUB

H. Peter Unks, Editor

FIRST CLASS MAIL