P 8

the AARDVARK JOURNAL

april 1980 vol.1 , no.1

WHAT ARE WE GONNA DO????

Basically, we are going to use this Journal to share information. While the information included will assist the "serious programmer, we are not going to emphasize only practical and utilitarian programs as I firmly believe that it is a mistake to allow yourself to be bullied into stating or believing that your hobby must be "practical". While I now make my living exclusively with computers, I still view them as fascinating and fun things. When the fun leaves, so will I.

I must admit that the "Utilitarian Fixation" among computerists confuses me. Most people seem to have a set of golf clubs and a TV or two around the house. They frequently cost more than a modern computer and strangely, no one asks "What are they good for?" Personally, I can't picture anything with less intrinsic value than a set of sticks that let me hit a ball a long way away so that I can chase it and hit it a long way away and chase it and.....

We are not going to ignore the fact that computers are a little better than abacus for some things. We are even setting up computerized inventory control and bookkeeping here at AARDVARK. However, when we do publish utility programs, we will insist that they actually be handier to use than your TI calculator and not just a way of telling everyone that your computer has some "practical" use.

We will publish at least one program per issue and probably more. As of this writing, we have planned to publish a Mastermind game in the first issue primarily to demonstrate the uses of string functions in doing digital manipulation. In the second issue we will publish the first checkbook balancing program that I have ever seen that really is handier than using your TI calculator. We picked it mainly to demonstrate a method of storing DATA on tape. We will not often publish complete programs for running your bookstore or gas station. The Basic idea of this Journal is to show you how to program, not program for you. We will, of course, make exceptions to publish any fun programs that come along.

This also seems like an appropriate place to clear up the connection between AARDVARK as a software house and AARDVARK as a Journal publisher. One of my pet peeves over the past few years has been semi-anonymous persons who publish OSI journals and then turn out to be software houses. When I subscribe to one, I get the damnedest feeling that I just got took by someone looking for a mailing list or a sneaky way to publicize their products.

It may not be a perfect solution, but we are going to try to handle that by announcing up front that we <u>are</u> a software house. We make money publishing programs and data sheets for OSI machines. We <u>will</u> announce new products in the Journal and keep you up to date on what we are doing, but we will not pretend to be neutral bystanders.

Now anyone with half a brain or more has got to be asking himself why a professional software house wants to cooperate in giving away its secrets. I admit that it doesn't seem rational at first glance. However, I have a firm belief that whatever improves the general quality of OSI programs will increase computer sales and eventually benefit my business. I want to see the quality of programs for OSI systems be as good as the best on the market (probably APPLE at this point). I also don't like TRS-80 software houses and do not want to see their attitude become common among OSI software houses.

I recently bought a TRS-80 to see what the competition was doing. I ordered about $100 worth of software recommended by various magazines. With one exception, it was dismal. It was undocumented, often ran poorly and was gimmicked so that it could not be read, changed, copied or corrected.

Anyone who knew anything went to great lengths to keep it secret. One graphics keyboard program ($19.95 - turn your TRS-80 into a PET-like graphics keyboard) was the equivalent of a three line BASIC program to take apart strings, increase the value of each character and put the strings back together again.

The music and sound effects tapes were rigged so that they couldn't be read - so you wouldn't find out the magic secret that POKEing letters to the cassette port caused sound output.

I can't help but think that such an attitude hurts everyone in the hobby. It probably accounts for the generally poor quality of available TRS-80 software. We have learned techniques at AARDVARK that will prevent the honest hobbyist from copying or listing a program in any normal manner (no, we won't tell you how!) What we found is that you can't stop thieves. Anyone crooked enough to steal, can find two tape recorders and do tape-to-tape stealing (as they do with TRS-80) so you can't stop them. You do stop the honest hobbyist from learning from and improving on the programs.

If we are right, we will continue to offer state-of-the-art programs for OSI, share all the information we can get and make a substantial amount of money. If we are wrong, we will continue to offer state-of-the-art programs for OSI, share all the information we get, and go out of business and stop offering any programs.

AARDVARK TUTORIAL #1

- STRINGS -

I would guess from the programs that are submitted to AARDVARK for publication, that the average computerist rarely uses strings to do anything but ask the question: INPUT" DO YOU WANT INSTRUCTIONS (Y/N)";A$. Strings, however, have many uses in data transfer and storage and, properly used, are one of the major secrets to writing compact and efficient BASIC programs.

Before we go on to discuss the uses of strings, we are going to review what the commands are. If you are already comfortable with A=ASC(MID$(A$,LEN(A$)-1,2)), I would suggest that you skip to the next section.

Let's start with the simple ones.

ASC(A$) - A=ASC(A$) makes A equal to the ASCII value of the first letter in A$ or the first letter in any group of characters you have specified with any of the other string functions. It is important to remember that no matter how many characters are in the string, ASC tests only the leftmost character. Whether A$ is "ABCD" or "AXXXY", the ASC of that string is 65, the ASCII value of the leftmost character.

RIGHT$(A$,X) - peels off X characters starting from the right side. If A$="ABCD"THENRIGHT$(A$,2)="CD".

LEFT$(A$,X) - is the same but works from the other side. If A$="ABCD", LEFT$(A$,3)="ABC".

MID$(A$,X,Y) - is the first one that may take a little concentration. As the neumonic suggests, it is a way to separate out a string from the middle of another string. In order to do that, the system needs to know three things - Which string do I separate, where do I start taking letters and how many do I take? The first element in the parenthesis tells the system which string, the second (X in the example) tells it which character (counting from theleft) to start with, and the third element in the parenthesis tells the system how many characters to take. Thus, if we go back to our famous "ABCD",MID$(A$,2,1)="B",

The most common error in using any of these functions is the FC error you get if you try to specify a string that does not exist, such as the rightmost six characters in a 5 character string.

The AARDVARK JOURNAL is published six times a year by Rodger Olsen, AARDVARK TECHNICAL SERVICES, 1690 Bolton, Walled Lake, MI 48088. Subscription rate: $9.00 a year. First class postage paid at Walled Lake, MI 48088.

A=LEN(A$) makes A equal to the length of A$. The principal uses of this function are to make sure there is an A$ (IFLEN(A$) > 0), and to write code that will manipulate several strings during a program where the strings may vary in length. For instance, if you are going to drop the last digit off of several strings of varying lengths (i.e. scores or names of players), you would write a subroutine like this: 100A$=LEFT$(A$,LEN(A$)-1) and it would automatically adjust for the length of the strings. It also saves you from the most common error that we mentioned in the last section. If you have written the code in terms of LEN(A$), you won't often try to treat digits that don't exist.

STR$(A) turns a number into a string. It is a particularly valuable function that all BASIC programmers should be comfortable with as it allows individual digit manipulation of numerical values. (We'll explain that later). The function does have one peculiarity that you have to get used to. It figures that the sign of a number is the first character of the string even if it is not printed. Therefore, if A=12 then STR$(A)=" 12", a three character string with the unprinted "+" as the first character.

VAL(A$) goes the other way. It turns a string into a variable type value. If A$="1324" then VAL(A$)=1324. This one is simple and uncomplicated.

CHR$(A) is another function that everyone needs to know. It allows you to print characters that you cannot access with the keyboard such as control codes and graphics. CHR$(A) is whatever character would be printed if you could print the value in parenthesis. For instance, CHR$(65) is "A". CHR$(254) is a tank. CHR$(13) is a tree on an OSI system and will give you a carriage return if you print it to a printer.

(Before you go on, be certain that you understand the difference between CHR$(A) and VAL (A). VAL returns a numeric value while CHR$ returns a character.)

Most new computerists who discover the CHR$ function tend to use it to print gaming characters during instructions, but the real value of it lies in it's ability to build and store strings that never existed and possibly could not even be printed. That allows us to build strings out of PEEKed values and store data in string format. Until we cover that, I suppose that it is sufficient to remember that if you execute PRINTCHR$(17), you will send to the printer or screen the ASCII symbol 17, which you can't access with a keystroke.

As a minor note, it is also the only easy way to get the system to print ". You know that if you try to include quotation marks in the middle of a print statement, the system will figure that is the end of the print and quit. However, if you execute PRINTCHR$(34) the system will print the ".

- MATH FUNCTIONS -

I have always felt that the OSI BASIC manual was a little flip in their bland assurance that strings could be compared, added, concentrated and so on... Actually, you have only three math functions that can be used directly.

You can use "+" to add strings together. They will appear in the final string in the same order they appear in text.

You can use = and < > to compare two strings to see if they are identical.

There are several things that you cannot do. You cannot directly subtract one string from another. A$=B$-C$ gives an error. You cannot assign a value to a character directly. MID$(A$,3,1)=65 returns an error.

To subtract part of a string, you have to define the string that is to be left and extract that instead. For instance, if you want to eliminate the last two characters of a string, you have to execute A$=LEFT$(A$,LEN(A$)-2)

To insert or change a character in the middle of a string, you have to break the string into pieces and reassemble a new string. If, for instance, we have the string "ABZDE" and want to change it to "ABCDE", we have to execute A$=LEFT$(A$,2)+"C"+RIGHT$(A$,2).

The difficulty in doing direct manipulations is what makes writing word processors so much fun and sells so much Anacin.

GETTING DOWN TO WORK
INPUT"DO YOU WANT INSTRUCTIONS (Y?N)";A$; IF A$="Y"THEN (go to instructions)

I have always wondered why computerists have this desire to tell everyone how to answer questions. How often does any English speaking user answer a yes or no question with "Si, Senor"? Most common negatives in English begin with "N" and most common affirmatives begin with "Y". I really think that the most we need to do is ask the question and then check for a "Y" or "N" in the answer. As it is trivially simple to strip out the first letter in a string, there is also no reason to specify that the user must limit himself to a one letter answer. We can check the first character of the answer and if it is a "Y", assume that the user said Yes, Ya, Yep, Yessiree or YOU bet. So we get: 100INPUT"DO YOU WANT INSTRUCTIONS";A$:IFLEFT$(A$,1)="Y"THEN (go to instructions).

As ASC(A$) automatically strips out the first character, we can even save a few bytes with IFASC(A$)=89THEN (go to instructions). The single character tests and the avoided explanations (Y/N) save a lot of typing and make the system seem a lot friendlier to use.

If you do any extensive programming, particularly business and utility programming, you will hit the situation where you do not want to choose between a numerical and string input statement. For instance, if the user is inputing a series of check amounts, you want to know when the last check has been entered without asking the user to count them all ahead of time and tell you how many he is going to do. (USERs who have bought computers do not appreciate doing things like counting for the computer!!). I have seen several ledger and check book programs done by asking "DO YOU WISH TO INPUT ANOTHER (Y/N)";A$; If the system gets a "Y", it then executes INPUT"CHECK AMOUNT";CH. If it gets a "N" after the first input, it goes on to process the check.

I suppose that works, but asking a user to press several more keys and wait for his answer to be processed before every entry is made, is very time consuming and damned irritating if you are doing several dozen entries.

The VAL function is the way to simplify the inputs. In the instructions specify a keyword such as "END" to signify no more checks and then input strings, check for the keyword, and, assuming the keyword is not found, assume that the user input an amount and convert it to a number with VAL(A$) - like this:

```
100 INPUT"CHECK AMOUNT";A$;
110 IFA$="END"THEN(go to next section of program)
120 A=VAL(A$)
130.....enter amount in record and jump back to 100
```

Now you don't have to choose between inputing A and inputing A$, the user doesn't have to answer a bunch of questions and the program is easier to use.

The CHR$ function and a few tricks of OSI BASIC allow you to input strings and stuff without carriage returns and to input strings that BASIC cannot normally handle.

Anyone who has read our catalog in the last year has probably seen the section entitled "INPUTS WITHOUT CARRIAGE RETURNS". However, for those of you who may have missed it, we'll go over it here. Every BASIC has a machine code section that handles the actual detection and decoding of keypresses. In OSI ROM BASIC, that routine is at FDØØ. In OS65D, the routine sits at $252B. The ROM routine stores the ASCII of the input character in location 532 (decimal), while)OS65D stores the character in location 9815 (9804 for C1P). Using it looks like this:

```
100 POKE11,0:POKE12,253          set up USR function for $FDØØ
110 X=USR(X)                     get character
120 P=PEEK(531)                  get ASCII
130 P$=P$+CHR$(P)                build a string
```

Disk BASIC is similar except for the location used. We use that setup for BLACKJACK and AWARI both because it does not disturb the displays and because it is relatively machine independent.

It is also handy for stuff like work processors because the input is not limited to BASICs 72 character input buffer and you can input stuff like commas

and periods without jumping out of the line. You can input a 128 character line complete with commas and even quotation marks.

Now we are going to cover complex choice input to a dimensioned array (impressed yet)? This is actually a fairly common situation dressed up with fancy polysylabolic terminology. This is the situation where the users next input is a choice between a fairly large number of alternatives. For instance, if you are doing a bookkeeping program, you might display 15 choices where the next check could be posted, such as "CAR", "rent" or "sales receipts". If you are sharp at all, you will number the choices and have the user input a number to call a catagory. That's not bad, but it requires about 3 keystrokes per entry (2 digits and a carraige return) and the keystrokes are up on the numbers row where they are irritating to make anyway.

The solution for neat programming is to letter the choices rather than number them, and input one character. Subtract 64 from the ASCII code that you get to get the number of the choices. If the user puts in an A and you subtract 64 from the ASCII, you get choice #1, B gives you two and so on. The nice thing is that one key can make any choice up to 30. There are no keys designated 22, 23 and 24, but X,Y,Z and + are on the keyboard and accessible with a single keystroke.

If you feel fancy, use the input without scrolls we discussed earlier, but we are going to assume a normal input routine. Take the ASC of the input character and subtract 65 to get a numbered choice that you can use for an ON __ GOTO or subscripted array manipulation. It looks like this: 100 INPUT"CHOICE";CH$:CH=ASC(CH$)-65. We used a similar routine in BACKGAMMON to allow one letter choices of all the 26 points that a player might want to move to or from.

CONVERTING STRINGS TO POKE VALUES

As OSI BASIC does not have a PRINT AT statement, it is necessary to POKE up names and scores and things that you want to appear anywhere but scrolled off the print line. Thank God, the string functions make that simple. We've covered how to do a PRINT at statement in most of our catalogs, but we'll review it here for those few demented souls who don't read our catalogs. Add this subroutine to your program: 5000FORY=1TOLEN(D$):POKED+Y,ASC(MID$(D$,Y,1)):NEXT:RETURN

To POKE up any name, work or even sentence on the screen, simply set the name equal to D$ and make D=equal the starting address on the screen. i.e. 300D$="WINNER IS":D=5404Ø:GOSUB5000

Scores should be done just a little differently. You start at the second digit because the BASIC thinks the sign is the first digit in the string and can set you over one space from where you planned. You may also want to blank the digit after the string to allow for the possibility that the score may decrease (say from three to two digits). To use it you set the score equal to D$ and the final product looks like this:

```
300D$=STR$(SCORE):D=5404Ø:GOSUB5000
5000FORY=2TOLEN(D$):POKED+Y,ASC(MID$(D$,Y,1)):NEXT
5010POKED+Y,32:RETURN
```

I might point out for the adventurous of you that a similar technique is used by some TRS-80 users to get low speed animation in pictures. They set up two or more character strings that overlayed provide cartoon action and then use print ats to alternate the strings (remember strings can be non-alphabetic characters). One of the most popular TRS-80 games, ANDROID NIM was done that way. It is a little slow, but does allow complex animation with a small memory.

STRINGS FOR DIGITAL MANIPULATION

Setting a number up as a string can make certain kinds of digital manipulations much simpler. For instance, we included the Mastermind game in this issue mainly to demonstrate how much simpler strings can make programming. The usual thing you do when you try to break down a number into its digits is to do a lot of fancy divisions and integers to get the individual digits. For you lucky computerists who are too new to remember this one, it went like this. Assuming you wanted to break down a 4 digit number into separate digits, you divide the number by 1000 and take the integer of the result.

To get the hundreds digits, you multiply the first result by 1000, subtract it from the original number, divide the result of that operation by 100 and took the integer of that number and so on for four digits. It is a lot of figuring. Most computerists chicken out and input the digits separately by asking the user to insert commas between the digits - a rather unnatural and unhandy process - who feels right inputing 1234 by entering 1,2,3,4 return. Using strings, the problem becomes trivial. If the number has four digits, the first digit is VAL(MID$(A$,1,1)). The second digit is VAL(MID$(A$,2,1)) and so on for all the digits. Of course, pairs of digits can be picked off the same way, which is handy for inputing dates without requiring a lot of special formatting. If the user inputs the date in a fairly standard 6 digit format (month, day, year), VAL(RIGHT$(2)) will give you the year, the same thing with LEFT$ will give you the month and a MID$ will pick out the day without the user having to put in commas, slashes or any other special character.

DIMENSIONED STRINGS AND DATA STORAGE

((**BOREDOM WARNING**)) THIS GETS A LITTLE THICK

To really understand where the advantages of string data storage are, you have to take a fresh look at what a string really is. It is not a collection of alphabetic characters. That is merely what you see when you print out the data from most strings. What a string really is is a set of integers from Ø to 255 which have a common name and which can be added to or examined individually with the use of string functions. Thought of that way, it has some things in common with subscripted arrays (individually addressable data bits, callable with a common name, each data bit can be changed without effecting the others.) It also has a few advantages over a subscripted array. Those nasty arrays have to be set up ahead of time with DIM statements and memory is partitioned off for every possible element whether you need it or not. This is even more wasteful in two dimensional arrays (editors note: Despite what OSI tells you in the manual, you do have 2,3,4 ... N dimensional arrays in your BASIC). You have to dimension out the same number of columns for every row even if some rows are going to be shorter than others.

The first time the problem really hit me was when I did my first Variable Table Maker program. The program searches the host program and lists each variable and each line the variable appears in. The program is mainly useful for documenting and sorting out large programs. The usual way to do one is to set up a 2 dimensional array. You set up an (X,Y) matric with X being the variable designation and Y storing the line numbers the variable appears in. Unfortunately, it requires that you make the same provision for storage space for each and every variable in the matrix. Therefore, the variable WH which appears once, gets as much space reserved for it as "X" the temporary counter that appears in every second line. Darned wasteful! On the other hand, if you set up a dimensional string, you need dimension only on one axis and the system sets aside only about 5 bytes for each string until the string gets longer. If one of the strings is rarely used, little space is allocated for it and if one gets very long, space is automatically allocated for that.

Here's how it worked in practice. To make the VTM, we set up a dimensioned string A(X) that was dimensioned for the maximum number of variables that we were going to handle. Notice that we only have to have a one dimensional array - length takes care of itself. When a variable name is encountered in the program, a string is initialized with the first three letters of the string being the variable name and a blank or a $ depending on what kind of variable it is. From then on when the variable name is encountered in text, it is matched up with the original string by searching all the variable names then on file. (i.e. FORX=1TO (number of variables on file):IFLEFT$(A$,3)=(Variable name) THEN... If a match is made, the line number is turned into a string and added to the string naming the variable. i.e. A$(X)=A$(X)+STR$(LINE NUMBER)+" ". (" " spaces the numbers out so they may be more easily read). When we have finished scanning the program, we simply print out all the strings. The first three letters are the variable name and the rest of the string contains the numbers of the line in which it appears. The nice thing is that if we only ran into a variable name once, we only set aside enough memory to store the one line number.

It doesn't work quite as well in practice as it does in theory as there are some bugs in the OSI BASIC, but it does work fairly well.

A second set of characteristics for strings and variable arrays can sometimes be important. Everytime you set up an element in an array (or even the space for a potential element) the system reserves space to store a six digit number, a decimal point and a pointer for that element. That's very wasteful if you are storing small integer numbers one byte long.

We ran into a problem recently doing an anagram program for a psychology experiment. (An anagram is a puzzle where a work has been scrambled and you have to unscramble it). For the purpose of the experiment, all words are five letters long and had to be scrambled in a sequence randomly chosen from 15 preset sequences. What you actually store for a pattern is five digits representing the order the letters will appear in. For instance, if you were going to reverse the last two letters and leave the first three alone, you store 1,2,3,5,4. The original FORTRAN program used a 15x5 array to store the numbers. We couldn't afford that on a mini computer. At 6 bytes an element, we were blowing about 450 bytes for the one array. What we did was to store the array as one long string. The first five characters in the string were the first pattern, the second five the second pattern and so on. To pick out a pattern, we used the MID$ function to pick five characters out of the center. We then broke the characters down in a manner similar to that used with the Mastermind game in this issue and used the individual values to build the anagram string. The nice thing was that we didn't have to store decimal points, headers and all the other stuff that goes into an array so we ended up using just about 100 bytes.

In a similar manner, we are using string to store the objects in an adventure game that we are currently developing. There are about 40 objects in the game that can be picked up and carried from place to place. We needed to know which objects were in which room and which one the hero was carrying. That entails having about 21 places where you can store up to 40 integer numbers from one to 40 without wasting a lot of space. We did it by setting up 21 strings. One for each room and one for the hero. When the hero picks up an object, we add its number to the Hero string and when he drops it, we take it off his string and add it to the string for that room. For instance, if the hero picks up a sword (object 15) we execute HERO$=HERE$+CHR$(15). To find out if he has the sword, we execute the code: 100FORX=1TOLEN(HERE$):IFASC(MID$(HERO$,X,1)=15THEN HE HAS THE SWORD. If he drops it, we subtract the sword from his string and add it to the room that he drops it in.

If we did the garbage collection routine, we can store all the possible combinations of places and objects and still only be out the 5 bytes per string overhead and the 40 bytes it takes to store the current locations of the objects.

It's a long way from the question "DO YOU WANT INSTRUCTIONS (Y/N)".

USING THE $219 PRINTER

Before we discuss the printer, we should talk about what most hobbyists, particularly C1-P users need a printer for. Despite all of our dreams of work processing, very few of us will use our systems for writing many letters. Most of us don't even write many letters to begin with. The lowest cost typewriter print quality printers are still in the $2,300 - $3,000 range. (I don't include used Selectrics - a good choice for a hardware handy masochist). What most of us need a printer for 90% of the time is to list and troubleshoot programs. We need to be able to look at SUBROUTINE2000 while we look at the line 200 that calls it without continually scrolling 4 or 5 lines across the screen.

We need to be able to look over a whole program at one time for errors, wasted space and documentation.

We use a printer at AARDVARK that we purchased for $219.00 brand new. It has recently been advertised for $179.00. It has one moving part and has never given us a minute's trouble - except for the six weeks of figuring out how to use it caused by its faulty design - and we can save you that six weeks.

We use the RADIO SHACK QUIKPRINTER II for program listings. It you have purchased a program from AARDVARK, you probably received a listing made on this

printer. It has some limitations as it prints on aluminized paper about 3 inches wide, but it gets 32 characters per line, (which is 7 better than the C1 display) and has an option for double width characters.

Installation requires that you populate the RS-232 port. That involves about $2.00 worth of parts and a helf hour with SAMS, your dealer, or the data sheet from AARDVARK. You also need to wire in the 600 BAUD conversion. That is supersimple on the C1. It requires a switch and some wire and another trip to your dealer or an order for a data sheet. The C2/4 is a little more complex. It took me about 2 hours on my C2 and I never did write up the instructions. Any handy type should be able to figure it out as OSI has pads for installation of other baud rates already on the board.

You only need to wire in the transmit and ground connections to the printer - that's where that six weeks of problem comes in. RADIO SHACK tells one little fib. The RS-232 does not work on the printer. It receives and prints data properly, but the CTS signal latches high or low at random. We received calls from the local computer center, the regional computer center and two people who purported to be from engineering centers in Houston. We got evasions, excuses, nonsense words (i.e. "It's just a software problem"), and promises. However, at last count, the CTS line still did not work.

To use the printer, you have to sync in without handshake. The wrap around buffer they advertise is one character long and simply adds to the confusion rather than helping.

You have to avoid carriage return (POKE 15,31), space out the characters with the baud rate simulater (POKE 518,190) and add some extra nulls at the end of each line (POKE 13,9). We have used three different C1 and C2's with the printer and all seem to use almost identical values to work properly. On one system we had to space the characters out a little more to avoid the loss of the first character on every line. We used POKE 518,200 on that one.

With all the delays, the printer ends up running at a little less than 300 BAUD effective rate, but that's a lot faster than I can type a program. You may have to fiddle the POKE values up and down a little for your system.

POKE 13,9 (add nine nulls)
POKE 15,31 (set width to 31 characters)
POKE 518,190 (delay set between each character)
CAUTION : BEFORE MAKING TAPES, YOU MUST RETURN TO THE NORMAL VALUES
POKE 13.0
POKE 15,72
POKE 518,0

WANT TO WRITE FOR THE JOURNAL??

We do want programs and informative paragraphs for this JOURNAL. We do not, however, feel that anyone should work for free. Any article worth publishing is going to save others countless hours or many frustrations with their systems. The good feeling of offering to help other computerists is nice and should be appreciated, and the minor fame of publishing a little piece is nice, but there should also be some reimbursement.

For paragraphs and short equipment and software review, we offer gift certificates for AARDVARK software; A $15.00 certificate for a paragraph sized tip (and we put your name in the JOURNAL) or a short product review. We will publish software reviews from other sources, but don't plan to do any ourselves as we want everyone to be able to trust the reviews. We also sell software and we don't want to get into conflict of interest problems.

Programs and articles will be offered gift certificates from $25.00 (short programs) to $100.00 for "how to do it" articles. We will also pay cash for exceptional "how to do it" articles.

If you send us a data sheet or article, put your name on the back of each sheet and your phone number or address. If you send a program tape, include your name, address and phone as program lines 1,2 and 3 in BASIC programs. Be sure to specify if the information is for the JOURNAL or for publication by AARDVARK software house. The requirements for the two are very different.

HOW ABOUT THIS FOR A LETTER COLUMN???

Send in questions. I'll answer all I can, and my staff will tackle the rest. We'll publish the answers in the next issues column. If you fool us and your question has public merit, we'll publish the question and scream for help. We can't promise to answer everything in one JOURNAL, but at least it will give us an idea of what you need or want to know.

TID-BYTES

This is a simple one, but the problems it causes if you don't know the trick are maddening. In order to LIST out a segment of program, type in LIST (1st line number you want to look at) - (minus sign signifies "to") (last line number you want to look at). (i.e. LIST 230-260 will cause all lines between 230 and 260 to be printed on the screen).

There are an unknown number of C1-P's out there with a hardware problem built into the cassette interface. (I know I spent several hours tracing down why our original C1 wouldn't load tapes as well as it should - in fact, it loaded better through the output port). If your C1 has these symptoms, the problem may be a bad foil run between diodes 9 and 10 and pin 2 of U66. Use a VOM set for resistance to check out the trace. If the trace is broken, a small jumper wire tacked inbetween the two points on the bottom of the board will solve the problem and have your cassette interface back in working condition.

Sometimes when you use a POKE statement as a direct command, as we do in order to run the QUICKPRINT II - the system will come up with an OM (out of memory) ERROR the first time you execute the POKE. Ignore it! Just repeat the POKE. The system will usually accept it the second time, if not keep POKEing.

If you have purchased our C1 CURSOR CONTROL, have we got a surprise for you. It is possible to do mid-line insertion with the CURSOR. Just CONTROL from the beginning of the line to the space where the insert should go. CONTROL the number of spaces needed for the insert. Then type in whatever you want to add (it will seem to wipe out part of the line as you type it in, so don't panic). At the end of the insert CONTROL to the end of the line, hit RETURN and list your line, complete with inserted material. (It may take a little practice to get the spacing just right).

Good News!! OSI fibbed to you. For some reason OSI still publishes manuals that claim that OSI BASIC is limited to one dimentional arrays. In actual fact, OSI BASIC handles N dimentional arrays. (i.e. You can dimention A for DIMA (5,5,5). Thus, your dimentions are limited only by the size of the machines memory.

BEGINNER'S CORNER

WE BELIEVE YOU!! At least once a week, we receive the following information from customers. For those of you who may have missed it:

(1) In ROM BASIC, POKEing a Ø into location 15 will make the system double space.
(2) It is not necessary to do a cold start everytime you hit the break key. That W that comes up in D/D/W/M means warm start and will restart your system without killing your program.
(3) On C2 and C4 machines, it is not necessary to POKE an exact Ø or 1 into location 569ØØ to change the type size. The system checks only the last bit, and therefore, any even or any odd number will work.

NEW FROM AARDVARK

SUPERDISK for C2/4/8 Contains a complete BASIC text editor and allows midline insertion, deletion and correction of BASIC lines. Also has BEXEC*, RENUMBERER, SEARCH and VARIABLE TABLE MAKER.
5¼" disk $24.95 8" disk $26.95

THE FIRST BOOK OF OSI It finally got here!! Written by Jim Williams and George Dorner, it is a 65 page expansion on the ROM BASIC DATA SHEET. While it is definately not for beginners (it presupposes a working knowledge of BASIC), it goes a long way toward telling you everything you wanted to know about OSI BASIC and couldn't pry out of the manuals. $15.95

TIME TREK A real time STARTREK - runs in 8K. $9.95

C1 TAPE CONTROL Puts your tape recorder under software control. Includes instructions for hardware modifications. Data Sheet. $3.00

THE C1 BEEPER Add a software controlled Beeper to your C1-P. Data Sheet $3

ADAPTING THE BASE 2 PRINTER FOR THE OSI 7 pgs. $4.00

ADVERTISING??

Yep, we will accept ads for the JOURNAL, but there are some restrictions on what we will advertise. We will accept ads for hardware, software and information sheets on OSI equipment. However, as we are a software house, we cannot afford to be associated in any way with second class software or impractical data sheets. Therefore, software ads and data sheet ads must be accompanied with a sample for evaluation.

Ads for hardware are $5.00 for up to 50 words. Ads for software and data sheets are $12.00 for 50 words and are subject to the rules above.

MASTERMIND

```
5 REM THIS IS A MORE DIFFICULT VERSION OF MASTERMIND
7 REM IT DOES NOT REPEAT NUMBERS, BUT USES MORE DIGITS (0-9)
10 PRINT:PRINT:PRINT:PRINT
20 PRINT"MASTERMIND,COPYRIGHT 1978,RODGER OLSEN"
30 PRINT"HIT SHIFT TO START"
35 IFPEEK(57088)=254ORPEEK(57088)=1THENR=RND(8):GOTO35
40 POKE56900,1
50 FORS=1TO34:PRINT:NEXT
60 PRINT"MASTERMIND FOR THE CHALLENGER"
70 PRINT:PRINT:PRINT:PRINT
80 INPUT"DO YOU WANT INSTRUCTIONS";A$
90 IFASC(A$)=89THEN540
120 FORO=1TOM:RF=RND(1):NEXTO
130 W=INT(10*RND(1))
140 X=INT(10*RND(1))
150 Y=INT(10*RND(1))
160 Z=INT(10*RND(1))
170 POKE56900,0
180 IFW=XTHENGOTO140
190 IFW=YTHEN140
200 IFY=XTHENGOTO150
210 IFW=ZTHENGOTO160
220 IFX=ZTHENGOTO160
230 IFY=ZTHENGOTO160
240 PRINT:PRINT:PRINT:PRINT:PRINT
```

```
250 PRINT"A MYSTERY NUMBER IS READY"
260 T=0
270 PRINT:PRINT:PRINT:PRINT:PRINT
280 PRINT"    CORRECT   EXACT
290 T=T+1:R=0:E=0
300 INPUTG$:FORX=1TO4:A(X)=VAL(MID$(G$,X,1)):NEXT
305 REM DO NOT ENTER LINES 310-340-THEY JUST DEMONSTRATE THE WAY YOU
307 REM HAVE TO DO IT WITHOUT STRINGS
308 GOTO350
309 REM 310-340 IS MENTIONED IN TEXT AS OLD WAY OF SEPARATING DIGITS
310 A(1)=INT(G/1000)
320 A(2)=INT(G/100)-A(1)*10
330 A(3)=INT(G/10)-A(1)*100-A(2)*10
340 A(4)=G-1000*A(1)-100*A(2)-10*A(3)
350 IFA(1)=WTHENR=R+1
360 IFA(2)=XTHENR=R+1
370 IFA(3)=YTHENR=R+1
380 IFA(4)=ZTHENR=R+1
390 FORL=1TO4
400 IFA(L)=WTHENE=E+1
410 IFA(L)=XTHENE=E+1
420 IFA(L)=YTHENE=E+1
430 IFA(L)=ZTHENE=E+1
440 NEXTL
450 PRINT"--------"E,R
460 IFT>30ANDR<4THENGOTO500
470 IFR<4ANDT=15THENGOTO280
480 IFR<4THENGOTO290
490 PRINT"CONGRATULATIONS":PRINT"YOU GOT IT IN"T"TRIES":GOTO510
500 PRINT"GOTCHA"
510 Q=1000*W+100*X+10*Y+Z
520 PRINT"THE NUMBER WAS"Q
530 PRINT:PRINT:PRINT:PRINT"CARE TO TRY AGAIN":GOTO130
540 PRINT"I WILL THINK OF A
550 PRINT"FOUR DIGIT NUMBER
560 PRINT"NO TWO DIGITS WILL BE THE SAME
570 PRINT"BUT THE FIRST ONE CAN BE A 0
580 PRINT"YOU GUESS THE NUMBER
590 PRINT"I WILL TELL YOU HOW MANY OF
600 PRINT"THE DIGITS IN YOUR NUMBER I USED
610 PRINT"AND HOW MANY I USED IN THE SAME PLACES
620 PRINT"FOR INSTANCE-IF MY NUMBER  WAS 1234
630 PRINT"AND YOU GUESSED 1478
640 PRINT"I WOULD SAY THAT YOU HAD 2 DIGITS
650 PRINT"CORRECT (1 AND 4)
660 PRINT"AND ONE EXACT (1)"
670 PRINT
680 INPUT"READY FOR MORE";A$
690 PRINT"TO GIVE UP-TYPE
700 PRINT"A RETURN WITH NO GUESS
710 PRINT" AND TYPE GOTO 500
715 PRINT:PRINT:PRINT
720 INPUT"READY TO START";A$
730 GOTO130OK
OK
```

RACE COURSE 600

```
5 REM THIS PROGRAM DEMONSTRATES JUST ABOUT EVERY WAY YOU CAN
7 REM CONTROL A GRAHICS CHARACTER
10 PRINT:PRINT:PRINT:PRINT"RACE COURSE 600"
15 PRINT:PRINT
20 INPUT"DO YOU WANT INSTRUCTIONS";A$:IFLEFT$(A$,1)="Y"THEN600
30 PRINT:PRINT"(1)KEYBOARD (2)FULL PADDLE (3)STEERABLE PADDLE
40 INPUT"PICK A GAME BY NUMBER";G
50 FORX=1TO8:READM(X),T(X):NEXT
60 INPUT"DEGREE OF DIFFICULTY";D:TI=30*(10-D)
70 LO=0:CR=.99:TP=54024
80 IFG>1THENPOKE530,1:POKE57088,128
90 FORX=1TO30:PRINT:NEXT
95 REM DATA IS MOVEMENT FACTOR (HOW FAR TO NEXT SQUARE IN THAT DIRECTION
97 REM ) AND GRAPHICS CHARACTER USED IN THAT DIRECTION
100 DATA-32,248,-31,249,1,250,33,251,32,252,31,253,-1,254,-33,255
105 REM THIS PRINTS A RACE COURSE- IT'S MORE FUN TO DO YOUR OWN
107 REM BUT UNTIL YOU DO, USE MINE
110 FORX=1TO2:PRINT"XXXXXXXXXXXXXXXXXXXXX":NEXT
120 FORX=1TO3:PRINT"XXX";:PRINTTAB(20)"XXX"
130 NEXT
140 FORX=1TO5:PRINT"XXX    XXXXXXXXXX    XXX":NEXT
150 FORX=1TO3:PRINT"XXX       XXXXXXXXX  XXX":NEXT
160 FORX=1TO4:PRINT"XXXXXX    XXXXXXXXX  XXX":NEXT
170 FORX=1TO4
180 PRINT"XXX      XXXXXXXXX    XXX":NEXT
190 PRINT"XXX    XXXXXXXXXXX  XXX"
200 FORX=1TO4:PRINT"XXX";:PRINTTAB(20)"XXX":NEXT
210 PRINT"XXXXXXXXXXXXXXXXXXXXX"
220 I=3
230 PRINT"CRASHES       LAPS
240 FORX=1TOTI:NEXT
250 POKE530,1:POKETP,T(I)
255 REM 270 IS A KEYBOARD ROUTINE/ 390=FULL JOYSITCK
257 REM 500 IS PADDLE TYPE USE OF JOYSTICK
260 ONGGOTO270,390,500
270 POKE57088,254:P=255-PEEK(57088)
280 IFP=129THENPOKE530,0:STOP
290 IFP=3THENI=I+1:IFI=9THENI=1
300 IFP=0THENTI=TI+30
310 IFP=5THENI=I-1:IFI=0THENI=8
320 IFP=7THENTI=TI-30
330 P=PEEK(TP+MK(I)):IFP=88THENGOSUB560
340 POKETP,32:TP=TP+MK(I):POKETP,T(I)
350 IFTP>53609ANDTP<53613THENL=L+1:POKE54103,L+48:IFL>9,9THEN760
370 FORX=1TOTI:NEXT
380 GOTO260
390 POKE57088,127:P=255-PEEK(57088)
400 IFP=64THENI=1
410 IFP=96THENI=2
420 IFP=32THENI=3
430 IFP=48THENI=4
440 IFP=16THENI=5
450 IFP=24THENI=6
460 IFP=8THENI=7
470 IFP=72THENI=8
480 IFP=128THENPOKE530,0:STOP
490 GOTO330
500 POKE57088,127:P=255-PEEK(57088)
510 IFP=32THENI=I-1:IFI=0THENI=8
520 IFP=16THENTI=TI+50
530 IFP=8THENI=I+1:IFI=9THENI=1
540 IFP=128THEN60
550 GOTO330
560 FORX=1TO100:POKETP,42:POKETP,32:NEXT
570 TP=53993:I=3:TIME=20*(10-D)
580 CR=CR+1:POKE54093,CR+48:IFCR>9,9THEN760
590 RETURN
600 PRINT:PRINT:PRINT"THERE ARE 3 GAMES AVAILABLE"
610 PRINT"THE JOYSTICK GAMES CAN BE PLAYED ON KEYS
620 PRINT"IF YOU ARE USING THE KEYBOARD
630 PRINT"(1)IS TRIGGER (2)UP (3)RIGHT (4)DOWN (5)LEFT
640 PRINT"FULL PADDLE GIVES THE 8 STANDARD DIRECTIONS
650 PRINT"STEERABLE MEANS ONLY RIGHT AND LEFT WORK AND
660 PRINT"YOU MUST STEER TANK WITH 2 TREADS
670 PRINT:INPUT"READY FOR MORE";A$:PRINT:PRINT
680 PRINT"TO PLAY STANDARD KEYBOARD
690 PRINT"USE (R.SHIFT) AND (L.SHIFT) FOR DIRECTION
700 PRINT"(REPT) END GAME EARLY
710 PRINT"ON PADDLE GAMES, THE TRIGGER ENDS GAME EARLY
720 PRINT"YOU GET 10 CRASHES PER GAME
730 PRINT"YOUR RATING DEPENDS ON THE LAPS YOU MAKE
740 PRINT"HIGHER DIFFICULTY RATING MEANS HIGHER SPEEDS
750 GOTO30
760 PRINT:PRINT:PRINT"  RATING   "100*(L/CR):PRINT:PRINT
770 CR=.5:L=0
780 GOTO60
```

RACE COURSE 540

```
5 REM THIS IS A 540 (C2/4/8)VERSION OF THE 600 RACE CAR PROGRAM
10 INPUT"DO YOU WANT INSTRUCTIONS";A$:IFASC(A$)=89THEN1100
15 INPUT"NAME OF DRIVER";A$
17 PRINT:PRINT"(1)KEYBOARD (2)FULL PADDLE (3)STEERABLE PADDLE
20 INPUT"PICK A GAME BY NUMBER";G
25 FORX=1TO8:READM(X),T(X):NEXT
30 INPUT"DEGREE OF DIFFICULTY";D:TI=20*(10-D)
35 LO=0:CR=.99:TP=54404:I=5
80 IFG>1THENPOKE530,1:POKE57088,128
90 FORX=1TO30:PRINT:NEXT
93 PRINTTAB(23)"DRIVER  "A$:PRINT:PRINT
95 PRINTTAB(23)"LAPS":PRINT:PRINTTAB(23)"CRASHES"
97 FORX=1TO10:PRINT:NEXT
98 REM DRAW A RACE TRACK--IT'S MORE FUN TO DRAW YOUR OWN
99 REM BUT YOU RE ALLOWED TO USE MINE
100 FORX=53248TO53503:POKEX,161:NEXT
110 FORX=55104TO55295:POKEX,161:NEXT
130 FORX=1TO45:POKE53705+X,161:POKE54729+X,161:NEXT
140 FORX=1TO18:POKE53715+X*54,161
150 POKE53740+X*64,161:NEXT
155 POKE53960,161
160 FORX=1TO10:POKE54204+X,161:POKE53952-X,161:NEXT
170 FORX=1TO3:POKE55134-64*X,161:NEXT
270 FORX=1TOTI:NEXT
300 POKE530,1:POKETP,T(I)
310 ONGGOTO313,500,700
313 POKE57088,1:P=PEEK(57088)
315 IFP=129THENPOKE530,0:STOP
320 IFP=3THENI=I+1:IFI=9THENI=1
325 IFP=0THENTI=TI+30
330 IFP=5THENI=I-1:IFI=0THENI=8
335 IFP=7THENTI=TI-30
340 P=PEEK(TP+MK(I)):IFP=161THENGOSUB1000
350 POKETP,32:TP=TP+MK(I):POKETP,T(I)
360 IFTP>54217ANDTP<54229THENL=L+1:POKE54300,L+48:IFL>9THEN2000
370 FORX=1TOTI:NEXT
400 GOTO310
500 POKE57088,128:P=PEEK(57088)
510 IFP=64THENI=1
515 IFP=96THENI=2
520 IFP=32THENI=3
525 IFP=48THENI=4
530 IFP=16THENI=5
535 IFP=24THENI=6
540 IFP=8THENI=7
545 IFP=72THENI=8
550 IFP=128THENPOKE530,0:STOP
560 GOTO340
700 POKE57088,128:P=PEEK(57088)
710 IFP=32THENI=I-1:IFI=0THENI=8
720 IFP=16THENTI=TI+50
730 IFP=8THENI=I+1:IFI=9THENI=1
740 IFP=128THEN30
780 GOTO340
1000 FORX=1TO100:POKETP,42:POKETP,32:NEXT
1020 TP=54404:I=5:TIME=20*(10-D)
1030 CR=CR+1:POKE54432,CR+48:IFCR>9THEN2000
1040 RETURN
1100 REM INSTRUCTIONS ARE THE SAME AS THE 600 VRSION
1300 GOTO15
2000 PRINT:PRINT:PRINT"  RATING   "100*(L/CR):PRINT:PRINT
2010 CR=.5:L=0
2020 GOTO30
2030 REM DATA ARE MOVEMENT FACTOR FOLLOWED BY DISPLAY CHARACTER
2035 REM FOR MOVE IN THAT DIRECTION
10000 DATA-64,248,-63,249,1,250,65,251,64,252,63,253,-1,254,-65,255
```