MICROPOLIS USERS GROUP

MUG Newsletter # 10 - May 1981

****

GRAPHICS PRIMER for VECTOR GRAPHIC

by Burks A. Smith of DATASMITH
Box 8036, Shawnee Mission, KS 66208

The first thing everyone should understand about computer graphics is that graphics capability is primarily a function of HARDWARE. Almost all "Home Computers" have some kind of graphics and a library of games because no one would buy them if they didn't. Business computers, on the other hand, almost never have fancy graphics. They are supposed to work, not play, and purchasers are more concerned with a crisp, readable display. I suspect that most MUG members own computers that both work and play. Our systems are more expensive and powerful than the "home" variety, but we still like to get some recreation out of them.

To use graphics at all (other than plotting letters of the alphabet), you need a video display with graphics capability. There are numerous "video boards" on the market that will plug into your computer and produce graphics. However, the VECTOR FLASHWRITER II is probably the most common among MUG members because it is standard equipment on MICROPOLIS equipped VECTOR GRAPHIC computers. The FLASHWRITER II is the business end of VECTOR'S "MINDLESS TERMINAL", which is nothing more than a black-and-white TV monitor. The FLASHWRITER generates the video signal that makes the patterns of little dots in the shape of letters appear on your screen. It will also generate 64 different "graphic characters" for special purposes.

All of the characters displayed on your terminal are stored in your computer in the form of number codes. The code used is the "American Standard Code for Information Interchange" or "ASCII" (pronounced Ask-ee) for short. These numbers are translated into a picture in the following way: First, the ASCII code is sent to the video board which uses the code to access its own Read Only Memory (ROM), which contains information on the pattern of dots that make up that character. Next, the video board changes the memory image into a TV signal that will make dots appear on your terminal screen in the appropriate shape.

Computers that have graphics capability use a type of display storage called "memory-mapped video", which means that the codes for all the characters currently appearing on the console screen are stored in your computer's own memory and your computer's microprocessor time is "stolen" to update the display. This is opposed to "smart" terminals (such as Hazeltine) which have their own memory and processor for these tasks and don't add any overhead to the host system. A memory-mapped video system such as VECTOR's with an 80 X 24 display uses one byte for each character displayed or 1920 bytes of the host system's memory just to store the screen information.

On the FLASHWRITER II video board, the 64 graphics characters consist of all the possible arrangements of six small near-squares that fit in the space normally occupied by a single letter on the screen. These squares are arranged two-across and three-down for each displayed position on the 80 X 24 screen, making a grid of 160 X 72 small squares (11840 positions) possible. By arranging the graphics characters on the screen in clever ways, a picture with crude-to-fair resolution is possible. A key to the graphics produced with the FLASHWRITER II can be found in its manual, which is included with all VECTOR systems. The codes used for the grapics characters are in the range 0 to 31 for the basic set, and 128 to 159 for the reverse of the basic set, giving all possible combinations of 6 squares set in a rectangle.

The problem in using graphics with MICROPOLIS-based systems, is that BASIC doesn't support any graphics commands. Actually, there is no way it could since graphics depend on hardware and MICROPOLIS BASIC is a general-purpose language that can be used on a variety of machines. Only specialized single machine Basics like APPLE BASIC can possibly include commands to create graphic displays. Therefore, all graphics using MICROPOLIS BASIC must be programmed by the user. Furthermore, the PRINT command can not be used to generate graphics because the codes for the graphics characters overlap standard ASCII control codes such as carriage return, control-C, etc.

The solution is to put the graphic codes directly into the area of memory used by the video display with the POKE command. The syntax of this command is:

```
POKE(address)=byte
```

The command stores a number in the range 0-255 (a byte) in the memory location designated by the address (0-65535). On most VECTOR GRAPHIC computers using the FLASHWRITER II board, video memory starts at address F000 Hex. This is written in Basic as 16RF000 or by its decimal equivalent of 61440. Therefore, the command POKE(61440)=12 will write the graphic character represented by the code 12 to the upper left-hand corner of the display screen. Other screen locations can be calculated from this value. Add 80 for each line down you want to travel and then the amount you want to travel left to right. For example, the middle of the screen is 12 lines down and 40 characters across (61440+12*80+40=62440). POKE(62440)=27 will write the character represented by code 27 to the middle of the screen.

Using the POKE command to write graphic characters to the screen from Basic is a slow and difficult process which discourages most would-be computer artists right away. It is certainly no good for real-time shoot-em-up games like you see in arcades. What is needed is a fast assembly language utility that can take care of the details of writing graphic characters and let Basic do the number crunching. You might want to experiment with POKEing some graphic characters, however, just to familiarize yourself with the available graphic characters.

NEXT MONTH we'll discuss some real programs that use graphics and an assembly language plot utility.

.....

PROGRAMMING WITH A SUBROUTINE LIBRARY

by Joel Shapiro, of Bonjoel Enterprises
P.O. Box 2180, Des Plaines IL 60018

There are many ways of generating an application program. Some require the use of rather rigid structures and others defy the use of structure altogether. The exponents of both these extremes can make a good case for their concepts. One suggests the rigid approach is necessary for further development of the program and optimum efficiency of its operation. The other makes a case for a program that works well regardless of how it's put together.

BASIC lends itself to the use of subroutines for functions that are used several times in a program. This allows the use of a modular concept in the structure of the program and the economies of saving some core. I tend to take the concept a few steps further and consider the use of a duplicate set of subroutines in all application programs that I write for myself and my clients.

The thought behind this is the simple fact we will probably establish a good means of performing a function that we wish to incorporate into other programs. Instead of picking these subroutines out of several programs, why not put them into a library so they're readily available. A further step is to give each subroutine its own set of line numbers so you will remember the lines for your subroutine call from the main body of your new program. The subroutines can then become the beginning of your new program and used as a base for the rest.

Each of the subroutines can be optimized for efficient operation and reduction of code. Subroutines not needed for the application can be deleted from the program after it is completed. Structuring around a defined set of subroutines decreases the probability of error and duplicate code.

The efficiency of the program is certainly affected by the use of subroutines and in many cases will be improved. The important fact supporting the use of subroutines is it's an effective means of generating an application program in a short period of time. By using our subroutine library properly the loss of efficiency can been reduced as well as the time required for writing, testing and debugging the programs.

The subroutines in your library should be those you're most likely to use in more than one program. Subroutines for; CLEAR SCREEN, CENTER TEXT, OPEN FILES, SEARCH DRIVES FOR A FILE, CONTINUE OPERATION, ERROR MESSAGE PRINTING, FORMAT DATES, CALCULATE DATES, PRINT PAGE HEADINGS, CHECK FOR SPECIAL CHARACTERS, etc., are all worthy of consideration for your library. As you develop your own style of programming, additional subroutines will become apparent and these can be added.

By now, you should be able to define quite a few subroutines you would like to have for your next programming endeavor.

There are a few rules and tricks I use in the library concept that may be helpful to you. First, make each subroutine self-contained. If you're feeding data to the subroutine, place the data into the proper variable BEFORE calling the subroutine. When the subroutine is going to feed back data to the body of the program, move the data from the subroutine's variables to the program's variables AFTER returning to the program. This way the integrity of the subroutine is protected.

Define a set of variables for use in the subroutine library and DON'T use them for the main body. DON'T use the subroutine's variables for holding data beyond the step before, or the step after the subroutine is called. By following these rules the same variables can be used in each of the subroutines without problems. Just make sure a value, a zero or a null is forced into the variables used in the subroutine by either the subroutine itself or the main program.

Take care in not calling too many subroutines from other subroutines. If you find yourself doing this it probably indicates you're trying to customize your subroutines to the application and that really defeats the purpose of the library. If this is the case it may be better to include the code in the main program or write a larger, more specific subroutine.

Your subroutines should be toward the beginning of the program. This helps increase the speed of their operation and groups them in one area of your listing. My own preference is to use lines 110 through 1999 for my general subroutines.

Don't be afraid to make improvements. If you find a better way of doing the job, rewrite the subroutine. I seem to keep learning as I write programs and like to incorporate my latest ideas into the library as time permits. Don't be surprised if you find yourself adding and replacing subroutines with each new program you write.

Remember, programming with a subroutine library is only another concept used in generating programs. I feel all programmers have a library of sorts, whether in their minds or readily available on disk. In either case, time and sometimes frustration can be eliminated with its application in your programs.

.....

SAVING ACROSS LOADS

by Bob Zale, of Systemation
PO Box 75, Richton Pk. IL 60471

We have noticed somewhat of an ongoing discussion in the MUG Newsletter regarding the possibility (or impossibility!) of saving a large block of variables in a single disk operation. We have put together two small BASIC programs to illustrate that this concept is not only feasible, but relatively straight-forward.

As a review of the listings will show, these programs have not been optimized, nor do they include appropriate limit-checking on the parameters passed to the main subroutine at line 1000. I'll leave this to the creativity of MUG members.

This concept can be valuable for several reasons. It can elimiate a multitude of long PUT and GET statements, as well as avoiding the need to block and deblock data records. Additionally, it utilizes all 256 bytes of each record, not just 250. In the case of numeric variables, efficiency is further increased as they are stored in a compressed BCD format, rather than in ASCII.

The first program "P/DIM" creates an array in high memory and saves it to disk as an object file. The second program "RETRIEVE" loads the data into an array in high memory.

These programs use the object file version of the LOAD and save commands to read and write variables to disk. However, since the LOAD command does not allow specification of a load address, the array must always be stored at the same absolute memory address. As BASIC allocates variable space dynamically, we must "fool" BASIC into thinking it has dimensioned an array in high memory above an address specified by a MEMEND.

BASIC maintains three 26-entry DW tables for pointers to active arrays. Any time an entry is found to be non-zero, it is assumed to be the absolute address of the array data. The sub-routine at line 1480 plugs the address of our "high-memory array" into the appropriate position of one of the tables.

Sufficient space must be allowed for the entire array as follows: A single dimension array requires a four byte header plus (# elements * element length) bytes. The element length for a numeric array is that specified by RSIZE or ISIZE in a sizes statement. For strings, it is the specified length +2, as a max length and current length byte is maintained for each element.

A final word of caution -- please do not attempt to utilize this concept on "live data" until you understand it throughly. Since this sub-routine alters BASIC, a minor error could "bomb" the system. Also, as this concept is not supported by Micropolis, non-standard or future releases of the BASIC interpreter might render this technique totally unusable. We do hope, however, that this concept will help to provide some insight into the "inner workings" of your system software.

Of course, if any specific questions should arise, we will be more than happy to provide any possible assistance to MUG members.

.....

Title: P/DIM

```
  30       !            Pseudo-Dimensioning Subroutine
  40       !
  50       !                by Systemation, inc.
  60       !
 100       MEMEND 16RCFFF : !  Set aside room for th
           e array
 120       Z0 = 16RD000 : ! Define the base address
           for the array
 140       Z$ = "A$" : ! Define the array name - Don
           't execute a DIM !
 160       Z1 = 11 : ! Define the number of elements
           in the array
 170       !             Don't forget element zero !
 180       !             i.e.  A$(0) through A$(10) =
           11 elements
 200       Z2 = 20 : ! If a string array, define the
           length
 220       GOSUB 1000 : ! Do the 'pseudo-dimension'
 240       FOR Z% = 0 TO Z1-1 : ! Fill up the array
           with data
 250           A$(Z%) = REPEAT$(CHAR$(65+Z%),Z2)
 260       NEXT Z%
 280       SAVE "N:ARRAY/FILE" Z0, Z0+Z3
 300       END
 970       !
1000 >*    ON INDEX("$%",RIGHT$(Z$,1))+1 GOTO 1040,
           1130, 1230
1010  *    !
1020  *    !      Real variable arrays
1030  *    !
1040 *>    Z2 = PEEK(16R4C5) :  ! Get the length from
           RSIZE
1050  *    GOSUB 1320 : ! Fill the block with binary
           0's
1060  *    GOSUB 1400 : ! Fill in the array 'header'
1070  *    Z4 = 16R3385 : ! Base address of real arr
           ay table
1080  *    GOSUB 1480 :   ! Tell BASIC where the arr
           ay is located
1090 <*    RETURN
1100       !
1110       !      String variable arrays
1120       !
1130   >   Z2 = Z2 + 2 : ! Strings require two lengt
           h bytes
1140       GOSUB 1320
1150       GOSUB 1400
1160       GOSUB 1550 : ! Plug a max length byte int
           o each element
1170       Z4 = 16R33B9 : ! Base address of string a
           rray table
1180       GOSUB 1480
1190 <*    RETURN
1200       !
1210       !      Integer variable arrays
1220       !
1230   >   Z2 = PEEK(16R4C6) :  ! Get the length from
           ISIZE
1240       GOSUB 1320
1250       GOSUB 1400
1260       Z4 = 16R33ED : ! Base address of integer
           array table
1270       GOSUB 1480
1280 <*    RETURN
1290       !
1300       !      Fill the block with binary 0's
1310       !
1320 >*    Z3 = Z1 * Z2 + 4 :   ! Calculate memory re
           quirement
1330  *    FOR Z% = Z0 TO Z0+Z3
1340  *        POKE(Z%) = 0
1350  *    NEXT Z%
1360 <*    RETURN
1370       !
1380       !      Fill in the array 'header'
1390       !
1400 >*    POKE(Z0) = 1 : ! A single dimension array
1410  *    POKE(Z0+1) = MOD(Z1,256) : ! Size in DW f
           ormat
1420  *    POKE(Z0+2) = Z1 \ 256
1430  *    POKE(Z0+3) = Z2 : ! Number of bytes per e
           lement
1440 <*    RETURN
1450       !
1460       !      Tell BASIC where the array is locat
           ed
1470       !
1480 >*    Z4 = Z4 + ((ASC(Z$)-65)*2) : ! Index into
           the table
1490  *    POKE(Z4) = MOD(Z0,256) : ! Array address
           in DW format
1500  *    POKE(Z4+1) = Z0 \ 256
1510 <*    RETURN
1520       !
1530       !      Plug a max length into each string
           element
1540       !
1550 >*    Z2 = Z2 - 2 : ! The length bytes aren't c
           ounted
1560  *    FOR Z% = Z0+4 TO Z0+4+((Z1-1)*(Z2+2)) STE
           P Z2+2
1570  *        POKE(Z%) = Z2
1580  *    NEXT Z%
1590 <*    RETURN
```

Title: RETRIEVE

```
  30       !            Array Retrieval Subroutine
  40       !
  50       !                by Systemation, inc.
  60       !
 100       MEMEND 16RCFFF : !  Set aside room for th
           e array
 120       Z0 = 16RD000 : ! Define the base address
           for the array
 140       Z$ = "A$" : ! Define the array name - Don
           't execute a DIM !
 160       Z1 = 11 : ! Define the number of elements
           in the array
 170       !             Don't forget element zero !
 180       !             i.e.  A$(0) through A$(10) =
           11 elements
 200       Z2 = 20 : ! If a string array, define the
           length
 220       GOSUB 1000 : ! Do the 'pseudo-dimension'
 240       LOAD "ARRAY/FILE"
 260       FOR Z% = 0 TO Z1-1
 270           PRINT A$(Z%)
 280       NEXT Z%
 300       END
 970       !
1000 >*    ON INDEX("$%",RIGHT$(Z$,1))+1 GOTO 1040,
           1130, 1230
1010  *    !
1020  *    !      Real variable arrays
1030  *    !
1040 *>    Z2 = PEEK(16R4C5) :  ! Get the length from
           RSIZE
1050  *    GOSUB 1320 : ! Fill the block with binary
           0's
1060  *    GOSUB 1400 : ! Fill in the array 'header'
1070  *    Z4 = 16R3385 : ! Base address of real arr
           ay table
1080  *    GOSUB 1480 :   ! Tell BASIC where the arr
           ay is located
1090 <*    RETURN
1100       !
1110       !      String variable arrays
1120       !
1130   >   Z2 = Z2 + 2 : ! Strings require two lengt
           h bytes
1140       GOSUB 1320
1150       GOSUB 1400
1160       GOSUB 1550 : ! Plug a max length byte int
           o each element
1170       Z4 = 16R33B9 : ! Base address of string a
           rray table
1180       GOSUB 1480
1190 <*    RETURN
1200       !
1210       !      Integer variable arrays
1220       !
1230   >   Z2 = PEEK(16R4C6) :  ! Get the length from
           ISIZE
1240       GOSUB 1320
1250       GOSUB 1400
1260       Z4 = 16R33ED : ! Base address of integer
           array table
1270       GOSUB 1480
1280 <*    RETURN
1290       !
1300       !      Fill the block with binary 0's
1310       !
1320 >*    Z3 = Z1 * Z2 + 4 :   ! Calculate memory re
           quirement
1330  *    FOR Z% = Z0 TO Z0+Z3
1340  *        POKE(Z%) = 0
```

```
1350  *  NEXT Z%
1360 <*  RETURN
1370     !
1380     !     Fill in the array 'header'
1390     !
1400 >*  POKE(Z0) = 1 : ! A single dimension array
1410  *  POKE(Z0+1) = MOD(Z1,256) : ! Size in DW f
         ormat
1420  *  POKE(Z0+2) = Z1 \ 256
1430  *  POKE(Z0+3) = Z2 : ! Number of bytes per e
         lement
1440 <*  RETURN
1450     !
1460     !     Tell BASIC where the array is locat
         ed
1470     !
1480 >*  Z4 = Z4 + ((ASC(Z$)-65)*2) : ! Index into
         the table
1490  *  POKE(Z4) = MOD(Z0,256) : ! Array address
         in DW format
1500  *  POKE(Z4+1) = Z0 \ 256
1510 <*  RETURN
1520     !
1530     !     Plug a max length into each string
         element
1540     !
1550 >*  Z2 = Z2 - 2 : ! The length bytes aren't c
         ounted
1560  *  FOR Z% = Z0+4 TO Z0+4+((Z1-1)*(Z2+2)) STE
         P Z2+2
1570  *      POKE(Z%) = Z2
1580  *  NEXT Z%
1590 <*  RETURN
```

DISK RUDIMENTS

The disk is divided into tracks - circles. Visually, the disk looks like a phonograph record. The physical reality is more like a magnetic tape on a tape recorder. The disk's tracks are not grooved like a record; they are each a separate entity. A phonograph record's grooves are not separate, but part of one long spiral. Reading and writing to a track is not accomplished therefore, by placing the read head in a "groove". There are no physical features on the disks to specify "this is a track". Any part of the disk surface can be written to and read from.

Information transfer is accomplished at whatever point the head is positioned. What portion of the disk to be used for information storage is therefore determined by head position, rather than anything on the disk. Head position is specified by commanding a stepping motor. Stepping motors move in "jumps", rather than smoothly. Just as you can't walk up a half a stair, a stepping motor can't move half a step. The movement is limited to the internal mechanics of the drive which are physically adjusted to some standard.

MOD I's and MOD II's read and write to different portions of a disk. That is, the step size of the stepping motors are different. That's why data written by one can not be read by the other. Physically, through, the same disk can be initialized and used on either type drive.

VARIATIONS

It is confusing to see all the different drives manufactured and all the various qualities - hard sector, soft sector, ten hole, sixteen hole, single, double, quad density, IBM compatible. Is everybody doing something unique? More on this next month.

.....

RENEWAL TIME

A year has almost gone by since the start of the MUG. The rates for next year will be $18 for the U.S., Canada, and Mexico; $25 airmailed elsewhere. I suppose I could go into a long dissertation in an attempt to justify the fee, but I'm not going to. From your point of view, the information is worth the cost, or it isn't. From my point of view, if I'm going to do this at all, I'm going to do it as well as I can. That translates to a lot of time and sizable monetary costs for phone calls, production and distribution expenses. I have to break even, both in terms of reimbursement of expenses and in justification of time.

I won't be sending back issues with mid-year subscriptions anymore. Any particular month's newsletter will be printed in quantities sufficient for the current membership. Back issues will probably be obtainable, but at an increased cost.

Your mailing label now contains the expiration date of your subscription. I truely hope you have enjoyed the newsletter and the other MUG benefits, and will renew for another year.

.....

MEMBERSHIP DIRECTORY

Many of you have asked for the membership directory. It will be available next month, (MOD I and MOD II). One problem is that many of you have not returned your information forms. I will not include the name of a member who has not specifically checked "NO" to the question "Would you object to ...."

Look at the second label which is affixed under the end of the text on page 6. The letter in the upper left-hand corner is either "Y", "N" or "U". If it's "U", I haven't received any information form. If it's "N", you told me not to release your name. "Y", of course, means you're on the distributable membership list.

Other items on the label are:

```
MFM = Manufacturer of your computer
DPY = Type of video display - If you have a
      terminal, its name should be listed here,
      i.e., HAZL-1400
      If memory mapped, the designation is
      vague but is meant to be the software
      (firmware) used to drive the display.
      VDM = Proc. Tech. Video Display (SOL)
      MT  = Vector Graphic Mindless Terminal
      EVD = Exidy Video Display
      VTI = Poly-88 Video Display
MEM = Amount of contiguous memory
TYP = Type of drives, "I" or "II"
HPN = Home Phone
PER = Peripherals
```

IMPORTANT NOTICE

It has been brought to my attention that the list can be used by a thief as a shopping list. With that thought in mind, if you want to be taken off, drop me a postcard.

YES AND NO?

PLEASE NOTE: Some of you have said "Yes" to releasing data to members, "No" to releasing data to vendors. You have been entered as "NO". For the record, I won't be releasing the data to non-members. However, that doesn't mean that members can't release the data to non-members. Nor does it mean that I won't use the availability of such data as an incentive to get vendors to join the MUG. If you read this newsletter, you are certainly aware that we already have vendors in the MUG. They are contributing the bulk of the material.

Never-the-less, the point is not to pick on those of you who prefer not to be on mailing lists. The point is, -NOTE!!- if you're on the list, anyone might get access to it, though, as I said, I will release it only to members.

The Membership Directory Disk will be available in the same manner as a Library Disk. Contribution of software or an article plus $3 ($5 overseas), or $15 ($17 overseas), will get you the disk. The same bartering rules also apply. See last month's

MUG LIBRARY story for details.

Finally, I welcome any suggestins on what additional information should be contained in the data base. Perhaps an INTEREST field, SOURCE number, HAM call letters - what do you want?

.....

CP/M TECHNICAL TIPS

by S. Tattersall, of ITT
London Rd., Harlow, Essex England CM17 9NA

A BACK-SPACE RUBOUT MODIFICATION

One feature of CP/M that is particularly annoying is the way it handles deletions. As distibuted, CP/M echos the deleted character on the console device. As this simple feature is buried within BDOS it is not easy to alter. A simple method of implementating this feature is to ammend CBIOS.

The first step is to ammend the console keyboard routine, i.e.,

```
......  Normal input routine (character in regis-
        ter A)
        ....
        CPI   7FH      ;IS IT RUBOUT
        JZ    RBOUT    ;IF SO JUMP TO ROUTINE
        CPI   5FH      ;IS IT AN UNSHIFTED RUBOUT
        RNZ            ;IF NONE OF THE ABOVE RETURN
                       ;TO BDOS
RBOUT:  MVI   A,01H    ;SET DELETE FLAG (CURSOR
                       ;LEFT)
        STA   DELF     ;STORE IT
        MVI   A,7FH    ;RESTORE RUBOUT CHARACTER
        RET            ;RETURN TO BDOS
DELF:   DB    00       ;DELETE FLAG STORE LOCATION
```

The above routine will detect the shifted and unshifted rubout key.

The next stage is to alter the console output routine, i.e.,

```
VIDOUT: EQU  E01B     ;SORCERER VIDEO OUTPUT
CON1:   LDA  DELF     ;LOAD DELETE FLAG
        CPI  01H      ;IS IT SET?
        JNZ  CON1A    ;IF NOT GO TO NORMAL ROUTINE
        CALL VIDOUT   ;SEND DELETE FLAG (LEFT
                      ;CURSOR) TO CONSOL
        MVI  A,20H    ;PUT A SPACE IN REGISTER A
        CALL VIDOUT   ;SEND TO CONSOL
        MVI  A,01H    ;PUT CURSOR LEFT CHARACTER
                      ;INTO A
        CALL VIDOUT   ;SEND TO CONSOL
        MVI  A,00H    ;RESET DELETE FLAG
        STA  DELF     ;STORE
        RET           ;RETURN TO BDOS
CON1A:  .........     ;NORMAL OUTPUT ROUTINE
```

The above program will backspace, space, then backspace again to achieve rubout on the screen.

As you can see from the above code, if you press the rubout key when there is nothing in the input buffer, the next character will not be displayed.

.....

HIGH-SPEED SORT FOR CP/M

Systemation has released SORT/B, an assembly language sort callable from Microsoft BASIC-80 (Rev. 5.0 or later). SORT/B, except for the CP/M & BASIC-80 linkage, is the same as SORT/A, which has been extensively discussed in previous newsletters. Available on MOD I & II for $75 ($67.50 to MUG members, at least for the month of May) from Systemation, Inc., PO Box 75, Richton Park IL 60471.

.....

FREE CLASSIFIED ADS

Enclosed is a card from Computer Shopper which enables you to place a free classifed ad in their publication. Please note, though, that the only allowable ads in the classified section are for used equipment, software (no software trading) and equipment wanted. New equipment, books and services must run as display ads. Those rates are available on request.

.....

LETTERS

CP/M LIBRARY; COMMUNICATIONS

Buzz,
We should consider buying the entire CP/M library which is now about 48 volumes and put them on Micropolis Mod II discs. Doing this as a group would be very inexpensive.

We should also consider buying as a group a driver for a common phone modem which would allow us to communicate and save programs and files. If we could do this, it would save us a lot of money and provide inexpensive back up with friends that have large amounts of data storage.

NEED FOR DOCUMENTATION

We should also consider publishing, in most elementary form, exactly how to determine the memory map of our systems.

Finally we should consider some way of determining the various configurations in which the Micropolis systems are used. This would allow us to exchange drivers that we have developed very simply.

Peter R. Senn
1121 Hinman Avenue, Evanston IL 60202

.....

Peter,
Several people have mentioned the CP/M library. The CP/M Users Group will soon have the library on Micropolis disks. I think we should wait for that.

I'd love to get a common driver for modems. I'll get to work on the problem and let you know the progress next month.

I agree with your documentation ideas. Several people have also made this suggestion. We'll work on it (along with 111 other things, so don't look for it next month).

.....

ACCESS TO 'DIR'

Buzz,
I have a couple of questions for the MUG members. It would ease my task of keeping track of all disk files if it were possible to read the disk directory itself as a BASIC data file, and thus access the information you get from a FILES command in MDOS. All I've been doing so far to create my SYSDISK file is entering the data, field by field, from the keyboard to create the record for each disk file. A few updates to the disk, and I'm way behind again. Perhaps Systemation would be able to help here as their programs have a nice directory display.

MOVE MDOS?

Also, would it be possible to re-locate MDOS so that the shared subroutines, especially the disk access ones, could be called as assembler functions from BASIC. This would allow a lot of flexibility and avoid conflicts with some older programs I have for the SOL which run in low memory where MDOS is. Perhaps Micropolis could write and sell a program allowing the user to specify where RES and MDOS would reside and execute. I would be willing to pay for such a program.

MORE DOCUMENTAION

I'll finish with one suggestion for the long run. This involves the creation of an extended documentation manual for MDOS, BASIC, etc., from all the information which the MUG gathers from its members and software sellers such as Systemation and DATA-SMITH, and whatever help the Micropolis Software Engineering Group could provide. I'm thinking something along the lines of organizing the material by software product, i.e., extended notes on MDOS (a perfect example is the @RFILESECTOR and @WFILESECTOR routines mentioned in the April '81 Newsletter). There would be other sections on LINEEDIT, ASSM, BASIC, etc., which would be a supplement and further explanation of those areas not covered or not fully explained in the Micropolis manuals. These items would only be published when they had been fully tested and clearly documented, so that the MUG could offer a reasonable guarantee as to their accuracy. To make such a manual easily maintainable, the data could be released by program and category on separate loose-leaf sheets, much as IBM issues updates to their manuals. This would mean that if the notes for one section changed, it would only be necessary to re-issue that section's pages, not the rest of the manual. All pages could be dated, and every so often all the updates consolidated into a completely new issue of the whole manual.

MUG INDEX

In regard to the above, I've been thinking of making a disk file index of the articles and information in the MUG newsletters (I have all of the issues so far). I should have a little more time to devote to this around the middle of May, and perhaps I could use this project to trade for Volume 2 of the MUG Disk Library. I find myself remembering a helpful hint from a past newsletter, and then end up looking through them page by page to find it. If you could send me an idea of what type of informaion would be most useful to a general index before mid-May, I'll go ahead and work on it.

Anyway, keep up the excellent work. The MUG has certainly been a great help to my Micropolis programming so far, and I can see it becoming even more valuable in the future. Thanks.

Ken Findlay
937 Briar Hill Ave, Tornto, Ont., Canada M6B 1M1

.....

Ken,
I'd be most happy to take you up on your trade. An index of Topics, S/W reviews, BASIC statement exiplainations, etc., would be very useful. Perhaps the members will write you directly with suggestions on categories that would make the index useful to them.

Again, I certainly agree on the need for documentation. I haven't found time to do it yet.

.....

SYSTEMATION'S COMPILER

Yup. I made you wait 'till last for the best. I have been GUARANTEED that the compiler will be released in May. We will certainly have a review of the specifications next month. With but a few execptions (you have to have THENs, you can't use EXECs), you can run your current programs after compiling with what we hope is a dramatic increase in speed and decrease in memory utilization. Can't wait to see. Cost? $345.

......

05/01/81

FIRST CLASS MAIL

FIRST CLASS MAIL

Subscription (August through July) rates:
U.S., Canada, Mexico; $12/year: Other, $25/year
Mid-year subscribers receive current year's back issues.
Published monthly by the MUG

MICROPOLIS USERS GROUP
BUZZ RUDOW, EDITOR
604 SPRINGWOOD CIRCLE
HUNTSVILLE AL 35803
(205) 883-2621

FIRST CLASS MAIL