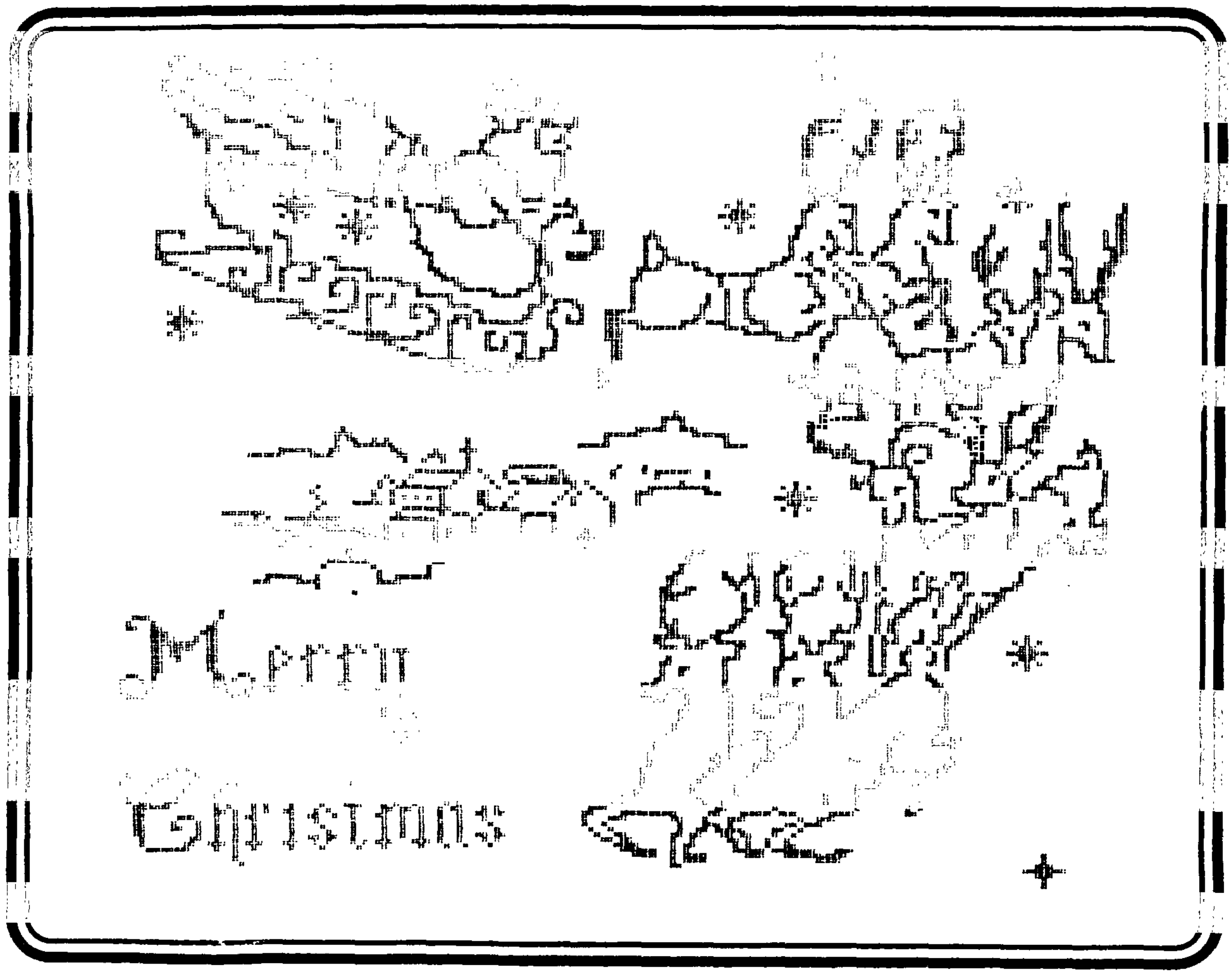
5612 OTTAWA (065

The Ottawa T.I.99/4 Users' Group

NEWSLETTER

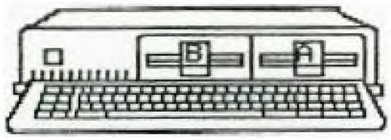
VOLUME 5 NUMBER 10......DECEMBER 1986



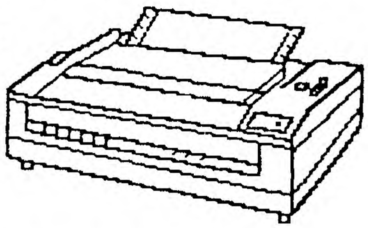
P.O. BOX 2144, STATION D, OTTAWA
*** ONTARIO, CANADA K1P 5W3 ***

Need a second Disk Drive? We have Panosonics on at a special price of:

$169.00



Looking For A Printer?



The Siekosha SP-1000 A is a Near Letter Quality printer that can't be beat. It sports a draft mode at 100 cps and a NLQ mode of 22 cps. With both tractor and friction feed the price can't be beat at

$369.00

Bulk DS/DD Diskettes

$7.95

DD-100L Diskette Box

$14.95

CHAIRMAN'S TWO CENTS WORTH

by Berry Minuk

Well those of you who braved the storm and made it to the December meeting were lucky enough to have your first look at the Geneve. The birth pangs were interesting but not astounding. We must all remember that what we had was only a prototype and the final product should be much superior. I don't think that we will be able to get a newer board for a while but we may be able to get a newer operating system for our next Geneve show. In the meantime we will just have to wait a while longer. There was also a report on the faire at the last meeting mostly given by the ever amiable Bob Boone with addendas by Jane Laflamme, Mike Taylor and Lucie Dorais.

In the last newsletter I mentioned that the tutorials had been cancelled until further notice but that workshops would be arranged instead. We are happy to announce that an Assembly Workshop led by Art Green has started. We will meet once a month on Saturday mornings. If you are interested please phone Art. The first project is to write an Archiver program that will compress files as well. If there is interest in other workshops they can also be arranged but you must let us know.

I would like to take the oppurtunity at this time on behalf of the executive and myself personally to wish you all a Merry Christmas, Happy Chanukah, Joyeux Noel, Bonne Annee and Happy New Year. Let us hope for new advances for humanity in 1987 and also continued growth for our poor orphan.

One thing that I would like to stress is that it is time to renew your membership. Early renewal helps the Membership Person keep her lists straight and gives us a good indication of how many members we will have in 1987 so that we can plan properly. If you have not yet renewed just mail in your cheque for $20.00 and let us have any changes in address, etc.

At the next meeting we should have demos of the long-awaited programs from Databiotics. We hope that our new multiple monitor system will be available so will all be able to see the demos properly. Also there is a good possibility of a demo of the new diagnostic package that TI has put into the public domain. This package which includes extensive documentation will be available as a special disk of the month for $4.00.

Remember that we still have a cassette library and tapes are available by contacting Jack McAllister.

I will see you all at the January meeting and let's have a super turnout for it.

FROM THE EDITOR

By Marg O'Connor

A special thanks to Bob Lanoy this month's guest Editor for all his typing and assembling of the bulk of the Newsletter.

An appology to David Caron, his second article on Basic Inputs for the TI console will have to appear in next month's Newsletter, we ran into several mutipage items this month.

GETTING STARTED WITH C

by Hal Tonkin

At a recent meeting there was a request for some form of turorial or beginner articles to be written for the newsletter. The intent of this article is to outline a method of naming and organizing the files necessary to create, and run a C program, then with an example, create, compile, load and run a simple C program.

In order to run C you will need the following minimum configuration. Console, expansion memory, one disk, and the editor/assembler (or TI writer). More disks, printer, etc are nice, but not necessary.

The C disks contain several types of files, and one problem is sorting out just what you need, and where it is. In order to keep track of what is in each file I have renamed some of the files, and all the files I create, according to the following simple file naming convention. The last two characters of each file name are a suffix consisting of a semi-colon (;) and a file type identifier. This still leaves eight characters for a meaningful file name. The file type identifiers that I use are:

;A	Assembler Source File
;C	C Source File
;I	C Include File
;L	Assembler Listing File
;O	Assembler Object File

The one exception is RUN PROGRAM FILES which have no suffix, since if there is more than one, there is an automatic increment of the last character.

The C disks which you have may contain run program files, source files, include files, documentation files, and object files. The following procedure will help you select the necessary files to get started.

1. Put WRITE PROTECT TABS on all your original C disks!

2. Load up DM1000 and get a catalog of all the disks.

3. Format and initialize 3 or 4 disks to become your working set.

4. Printout and/or read the files C99MAN1, C99MAN2, C99MAN3. These describe the C compiler.

5. Copy to a new disk the following files.

C99C	The C compiler (three files)
C99D	
C99E	
CSUP;O	The object support library
STDIO;I	The standard I/O include file
TCIO;I	The enhanced I/O include file
FLOAT;I	The floating pointinclude file

I also like to put the editor (EDIT1) and the assembler (ASSM1, ASSM2) on this disk as well.

This disk will be the basis for program development. Put a write protect tab on it!

6. Copy to the same disk, if double sided, or to a separate disk if single sided the following files:

C99PFF;O	The final file loaded when using SAVE to produce run program file
C99PFI;O	The initial file for SAVE
CFIO;O	The file I/O object file
CSUP;O	The object support library
TCIO;O	The enhanced I/O object file
FLOAT;O	The floating point object file
SAVE;O	From your Editor/Assembler disk
PRINTF;O	The printf object file
FPRINTF;O	The fprintf object file

This will be your object library disk. Put a write protect tab on it also!

7. · Copy your program development disk to another disk. This is the one that we will use. Every time I start on a new program I make a copy the development disk to use as a starting point, then add or delete files from it as required.

Later on you will want to add more to your development disk, and your object disk, but this should be enough to get you started.

We are now ready to create and run a C program. For the following example, I will assume that everything is being done on DSK1.

I will outline the procedure using the C program described below. This program will fill the screen with asterisks. The corresponding EXTENDED BASIC program is described at the end of this article, so that you can compare the language statements, and the difference in execution times between them.

The process of producing a running C program involves the following five steps.

```
1.  Create a file containing the C program
   source statements.                                    STARS;C
2.  Compile the C source statements to create
   an assembler source file.                             STARS;A
3.  Assemble the assembler source statements
   to produce an object file.                            STARS;O
4.  Load the assembler object file, and any library
   object files required.
5.  Run the program.
```

The first step is to use the EDITOR to type in the following C program. The blank lines are not necessary, but make it easier to read the program. Similarly everything between /* and the next */ is a comment, and is ignored by the compiler, but makes it easier to follow what is going on in the program.

```
/* FILL SCREEN WITH STARS */

#include DSK1.STDIO;I

main()            /* all programs are called main */
 {                /* start of program */
 int row, col ;   /* define integer variables */
 char star ;      /* define character variable */

/* start of program

 putchar(FF) ; /* clear the screen */

 star = '*' ; /* character to put on screen */

 for ( row = 1 ; row = 24 ; row++ )   /* rows 1 to 24 */

  {

  for ( col = 1 ; col = 40 ; col++ ) /* columns 1 to 40 */

   {

   locate(row,col) ;    /* position cursor */

   putchar(star) ;      /* write * character to screen */

   }                    /* end of col loop */

  }                     /* end of row loop */

 }                      /* end of program */
```

Save this program in a file called DSK1.STARS;C and exit the editor.

For step two we return to the main menu, and using option 5, load and run the C compiler file DSK1.C99C.

The compiler will ask the following questions, with the default answers in square brackets. The answer column indicates what you should type in. ´enter´ means pressing the enter key.

question	answer
include c-text? [n]	y if you wish to see how each C statement is converted to assembler code, otherwise n, followed by ´enter´.
inline push code? [n]	´enter´ to select the default no.
input filename?	DSK1.STARS;C
output filename?	DSK1.STARS;A

At this point the compiler will begin to compile the source file and create the assembler input file and produce an assembler source output file. If there are no errors, then you will get the message:

```
    compilation complete
```

If you have made a typing error, then the compiler will stop at the line in error, and give some indication of the type of error. Press ´enter´ to continue compiling.

When the compiler has finished it will indicate the number of errors, if any, and ask the following questions.

question	answer
c99 exit-rerun (y/n)	n
PRESS ENTER TO CONTINUE	´enter´

There should be no errors in this program, however if you made a typing error, go back to the editor, correct it, and recompile the program. When there are no errors, continue on with the next step in the process.

The third step is to assemble the output file from the compiler DSK1.STARS;A. The assembler will produce an object file for the loader, and optionally a listing file for us to look at.

From the main menu, select 2, the assembler.
The assembler will ask the following questions. The answer columns indicate what you should type in.

question	answer
SOURCE FILE NAME?	DSK1.STARS;A
OBJECT FILE NAME?	DSK1.STARS;O
LIST FILE NAME?	DSK1.STARS;L for a listing,otherwise ´enter´
OPTIONS	LC if you requested a listing, otherwise just C

If all goes well, then there will be no errors, and you are ready to continue on with the next step.

The fourth and fifth steps are to load your object file, and the C library file that contains the routines your program has called, and then run your program. Return to the main menu and select option 3, LOAD AND RUN

The questions and answers are as follows:

question	answer
FILE NAME	DSK1.STARS;O
FILE NAME	DSK1.CSUP;O
FILE NAME	´enter´
PROGRAM NAME	START

At this point the screen should clear, and then fill with asterisks. The program ends by asking c99 exit-rerun (y/n). Enter y to rerun the program, n to exit.

Congratulations!! You have now created and run a C language program! This is just a beginning. To learn more about C, read one or more of the books that are available in the library. One book which I have read and can recommend is C PROGRAMMING GUIDE by Jack Purdum. Also, don´t be afraid to ask any for help from any of the club members that are familiar with C.

Here is the equivalent EXTENDED BASIC program to fill the screen with stars. Note that C works in 40 column mode, and XB only has 28 columns when using the DISPLAY AT statement.

```
100 REM FILL SCREEN WITH STARS
110 STAR$="*"
120 CALL CLEAR
130 FOR ROW=1 TO 24
140 FOR COL=1 TO 28
150 DISPLAY AT(ROW,COL):STAR$
160 NEXT COL
170 NEXT ROW
```

PANASONIC FLOPPY DRIVE 149.00

COPAL 1200L 329.00

EXCELTRONIX COMPUTING INC.

217 BANK STREET

230-9000

MAR'S BYTES

by Marilyn Boone

First, let me say how nice it is to get some positive feedback from people who read and like my articles.

Now for a lighter look at Chicago!!

I packed very carefully and got everything required into a small suitcase for Bob and I - only to find out that I could take a larger one. Oh,joy!!! Room to bring back more! Jane and Lucie arrived shortly after this and their three suitcases (not each!!) were arranged in the trunk. Mike Taylor arrived on the scene and there was still room for his suitcase! So with several hundred km. ahead, we started our journey (Wed. afternoon) and arrived near the U.S. border about 10 pm. The task then was to find a motel and have something to eat. We arrived at the motel and unpacked the trunk.

Next morning, after packing five suitcases back in with everything else, we had an almost uneventful trip. I say ALMOST uneventful- most turns are made at stop lights....WE made a U turn on a four lane highway and were finally off in the right direction. We arrived in Chicago none the worse for wear at supper time. After unloading the trunk AGAIN of 5 suitcases, boxes of diskettes, books, (well, you fellows who help Bob unload the trunk for meetings know what I am saying!), a video camera, recorder, tripod..........

Friday was a beautiful day. We planned an early breakfast and a tour of Chicago. Our tour guide was Lucie Dorais who did a fantastic job. We toured the art museum. I was impressed by the paintings. I've been to the Louvre in Paris, but this was totally different. There were sections on Mexican art, Medieval costumes of many eras, gun cases, glass and porcelain displays. Also, in the same building was the famed Chicago stock exchange.

I lost Bob in this building, but soon caught up with Jane and Lucie. We explored the impression era of Rembrante, Dante and Picasso to name a few. Then we did Madison Avenue. Lucie pointed out various styles of architecture in the city and we all went around(with our noses in the air) looking at the various buildings. One thing I did enjoy was the sculpture by Picasso in the down town section but the piece de resistance was our trip up to the top of the Sears Tower. The building was secure but if I stood still- I swayed -only slightly. The view was fantastic! We could see for 20 miles (so the brouchure said) on each side. The expressways were like trails of red lava with all the car lights exiting the city. Everyone enjoyed their day!

Saturday was the day of the Chicago TI faire - the only day it rained. We were a bit sluggush getting started but after breakfast we were full of anticipation at seeing the new computer. We arrived at Triton College and set up. As in New Jersey, everyone came to greet these Canadians from Ottawa. We, however, were not the only Canadians. There was a group from Hamilton, Bruce Ryan (of Ryte Data) and his family , and a few other individuals. We had an extremely busy morning. Hardly anyone from our booth had time to leave for any reason. If they did, it was hurried. Later in the afternoon, things slowed down and I left to do some shopping. No use travelling for two days if I can't go shopping. The shopping plaza was easy to find this time and I spent two hours and a fortune. The problem was; how do I get it all into the suitcase.(I was not the only one with this dilema.) I arrived back at Triton in time to see the new computer by Myarc. I was a bit disappointed as I was expecting a whole, new system- not just a board and disk. Oh well!!!!!

Jane had a chat with Lou Phillips about importing the new computer. We even got to bring a couple of boards home. Mike bought a printer at a fantastic low price. "Where do we put it?"(A2B) We were thinking about a car top carrier for Mike and his printer......

Everyone bid a fond farewell to new and old friends with promises to see them at the buffet supper. We relaxed a little too long and missed out on seeing everyone . So we ajourned for a late supper. Most of us were too tired to consider Hank Ellerman's gracious invitation to visit his home after the buffet. Everyone crashed.

Next morning, Mike was at his task again - that of finding room for everything. His printer was neatly tucked away in his suitcase. I wore most of the clothes I bought and everything was packed in like a jig-saw puzzle with the fervent hope that it would only have to be disturbed ONE more time. Late that night we stopped and very carefully undid Mike's work. Mike was so good at this by now, everyone had great confidence that he could tackle the task once more in the morning. We weren't dissappointed. Monday morning Mike did it again, but this was to be a day unlike any other. We came back south of the Great Lakes. Later on, after covering about 500 km the car started to sway. I thought it was the road because it was a bit slanted but it was - a flat tire. Out came all the boxes, suitcases,clothes, etc. to get to the '50 km' tire. Then everything was put back , not as carefuly this time. We found a service centre and hauled everything out only to find out they had no spare to match. So......back in again. We continued on to a tiny town and found a reasonable facsimile of our original tire to get us home. Now in the trunk, in addition to all we had before is the flat tire. All the purchases at the duty free shop now had to travel as 'passengers'. The time at this point is 10pm. Everyone breathed a sigh of relief when we breezed through customs. The trunk was repacked a total of 8 times - 5 of which happened on our way home. Mike has proved to be a very valuable asset on a trip to Chicago!

One other funny item.....the car was so heavy in the back that the oncoming traffic and transport trucks we were behind all kept flashing their lights at us. It took a while for us to realize that our low beams were hitting their mirrors and they thought we were using high beams.

TI-99/4(a) Disk Format

The following is a complete and, to the best of my knowledge, accurate description of the Disk Directory format and file storage allocation used by the TI-99/4(A) computer.

SECTOR 0 - Volume Information Block

ADDRESS	CONTENTS
0000-0009	Disk name - up to 10 characters
000A-000B	Total number sectors on disk (0168=360, 02D0=720, 05A0=1440)
000C	09 (# of sectors/trk)
000D-000F	'DSK' (44534B)
0010	50 = Disk backup protected, 20 = not protected
0011	# of tracks per side (28=40, 23=35)
0012-0013	# of sides/density (0101=SS/SD, 0201=DS/SD, 0202=DS/DD)
0038-end	Sector allocation bit map. See note below

NOTE on 0038-end:
This is a sector-by-sector bit map of sector use; 1=sector used, 0=sector available. The first byte is for sectors 0 through 7, the second for sectors 8 through 15, and so on. Within each byte, the bits correspond to the sectors from right to left. For example, if byte 0038 contained CF00 then the first byte equals 1100 1111. This means that sectors 0 through 3 are used, sectors 4 and 5 unused and sectors 6 and 7 used. Information for the 2nd side of a DS/SD disk starts at byte 0065 and ends at byte 0091.

SECTOR 1 - Directory Link

Each 16-bit word lists the sector number of the File Descriptor Record for an allocated file, in alphabetical order of the file names. The list is terminated by a word containing 0000; therefore, the maximum number of files per disk is 127 [(256/2)-1]. If the alphabetical order is corrupted (by a system crash during name change, for instance), the binary search method used to locate files will be effected and files may become unavailable.

SECTOR 2 TO 21 - File Descriptor Records

ADDRESS	CONTENTS
0000-0009	File name - up to 10 characters
000C	Filetype: 01=Program(memory-image) 00=DIS/FIX 02=INT/FIX 80=DIS/VAR 82=INT/VAR File deletion protection invoked by Disk Manager 2 will be shown by 08 added to the above.
000D	# of (MAXRECSIZE) records/sector
000E-000F	Number of sectors allocated to the file. (Disk Manager 2 will list one more than this number, thereby including this sector in the sector count)
0010	For memory-image program files and variable-length data files, this contains the number of bytes used in the last disk sector. This is used to determine end-of-file.
0011	MAXRECSIZE of data file.
0012-0013	File record count, but with the second byte being the high-order byte of the value.
001C - end	Block Link (see note)

Note on file storage:
Files are placed on the disk in first-come / first-served manner. The first file written will start at sector 0022, and each subsequent file will be placed after it. If the first file is deleted, a newer file will be written in the space it occupied. If this space isn´t big enough, the file will be ´fractured´, and the remainder will be placed in the next available block of sectors. The block link map keeps track of this fracturing. Each block link is 3 bytes long. The value of the 2nd digit of the second byte followed by the 2 digits of the first byte is the address of the first sector of this extent. The value of the 3rd byte followed by the 1st digit of the 2nd byte is the number of additional sectors within this extent.

Sectors 2 through 21 are reserved for File Descriptor Records and are allocated for file data only if no other available sectors exist. If more than 32 files are stored on a disk, additional File Descriptor Records will be allocated as needed, one sector at a time, from the general available sector pool.

Earl Hall

1987 RENEWAL
$ 20.00

If you have not done it yet, send
to: The OTTAWA TI99/4A Users' Group
P.O. BOX 2144, STATION 'D'
OTTAWA, ONT. K1P 5W3
or: bring it to next meeting

TIMP TIPS _TECHNIQUES

by

Steve Zimmerman

Many thanks to those of you who left questions for me on the TI SIG and SCCG boards! I'll try to answer some of them here this month.

One of the most useful features of Multiplan (and one that I miss on my Tandy 200 portable!) is WINDOWS! Windows are a BIG THING now--just read the magazine reviews and ads for programs such as Framework and Symphony. Windows enable you to see two parts of the screen, in different areas of the worksheet, at the same time. Sounds simple enough, but what good is it? Well, let's say that you have a worksheet with about 60 rows of labels in Column 1, and that you enter data for each day in a month in columns 2-32. Since your basic column width shows only 4 columns, you can see the labels only while entering the first 3 days data. If you make the columns narrower, of course, you can see more--if the numbers aren't too long! If the numbers are too long, and you make the columns narrow, you just see "#####", an error symbol which means that you have too many numbers to display in that column width. To enable you to enter later columns of data in the proper rows, you need to create a window.

To do this, position your cursor (the cell pointer) in R1C2 (one column to the right of your labels), and hit the W key (for Windows). On the next menu, you want to key S (for Split). On the next menu, key V for vertical, and Multiplan will respond with "at column 2", Linked yes no, with the NO highlighted. Use the Tab (Ctrl 2) to move the command cursor from the 2 in "at column " to the Linked field, key Y to link the windows, and then key Enter. With practice, this command sequence becomes W S V Tab Y Enter. You now have a window, linked vertically. As you move the cell pointer down in the active window (the one you just created), the labels move along. As you move the cell pointer to the right, columns will disappear on the left side of the second window, but the labels will remain in view.

To 'uncreate' this window, key W (for Window) C (for Close); Multiplan will respond with #2, key Enter, and the window closes.

You can do the same thing with horizontal windows, if you have labels across the top and you want to see them as you move up and down. One thing you cannot do, however, is to link windows which intersect in the same cells. R1C1 cannot be in 2 linked windows, one for R1 (horizontal) and one for C1 (vertical).

You can move the cell pointer between windows by using Crtl 6 (change window), which makes a different window the active window. Ctrl 6 again will take you back to the original active window.

I hope that this 'raises the shades' for those of you with windowing questions! To cover some other questions, I don't know of any way to load the recalc routines when you start out in Multiplan. This will, of course, cause those with only one drive to have to swap disks, starting with the Multiplan disk, then your data disk, and then back to the Multiplan disk to load the recalc routines. The program will also have to access the system disk to set up Names, to do Xternal copies, to show you your disk directory, and to load the routines for logical operators. It appears that these are handled by some type of overlay, which is loaded from the system disk only when needed. I don't know for sure, but it is possible that one or more of these routines uses the same (or overlapping) addresses in memory. Oh, yes, one more thing which requires access to the Multiplan system disk is the Help files. These can be quite handy, though! To get Help, just move your command line cursor to the command you need Help with, and key Fctn ?. To get Help with Help, move the command cursor to Help and hit Enter.

One more handy hint on recalculation--to recalculate a single cell, move your cell pointer to that cell, key E (for Edit), DON'T CHANGE ANYTHING, and hit Enter. This will cause the contents of THAT CELL ONLY to be recalculated. Of course, if you left recalc turned on....

Other handy hints--to Un-Name Names, enter the Name, and delete the reference in the "to refer to" field--this deletes the name by making it refer to nothing. To delete an Xternal reference or link, the same procedure is used. You enter X (for Xternal), C (for Copy), from sheet (sheetname), Name (enter the name), to: (delete anything in this field), Linked (yes), Enter. This redefines the Xternal copy link to refer to nothing, and the value placed in the cell by the Xternal Copy command will disappear, and the cell will now be unlocked. To change the target of an Xternal copy, use the same procedure, but specify the new cell or cells in the "to:" field. Since each cell or range on the supporting sheet can have only one target (on the dependent sheet), the old link will be replaced by the new.

Well, I suspect that I've written more than I should, so I'll cut it short for now. There are a few more questions that have been sent in, and I'll try to cover those next month. Until then, have fun with MulTIplan!

WRITER-WROUTES

by Jane Laflamme

As a personal preference, I like to work with a screen only of text by setting my left (L) and right (R) margins by entering "T" for Tabs, at 1 and 39. I then get rid of the line numbers by using Fctn. 0. (BUT never do this for a variable file for mail listings. The letter can be typed with these margin settings, but not the variable file; the left margin must be set at "0" or the printing will stop when it accesses the variable file and you will be returned to the menu screen. Took me a long time to figure THAT one out!). Also, remember the finished document's margins will be set through the Formatter with the .LM n and .RM n.

Of course, I only work with a screen of text while in word-wrap (Solid cursor, or power-up) mode. I can usually work around all problems in that mode but sometimes I need to work in Fixed mode (Hollow cursor or toggle with Ctrl. 0). If within my document I need to use fixed mode to align my data, I set up a separate file and include it in my document at the appropriate place, with the .IF DSK2.ADDLDATA. Within the fixed mode file, I set my left and right margins in Tabs to 0 and 79, or appropriate settings, and needed Tabs for use with Fctn. 7. That way, I only need to window over in Fixed Mode. I then continue with the original document in word wrap and a screen only of text with no need to window over 80 columns.

When reworking your file in fixed mode, be sure to toggle on Ctrl. 0, for if you inadvertantly use Reformat, Ctrl. 2, you'll have one "heckava" jumbled mess!

If you have questions please don't hesitate to contact me, either by phone or writing to me in care of the Newsletter. I will attempt to answer your questions in future columns. If you have a tip of your own, or a better way of doing things, I'd also like to hear from you! Until next time...keep on TI'ing!

FORTH TO YOU, TOO! - SESSION 2

by

Lutz Winkler

You have determined which of the editors suits you and found a display color you like. They could be entered from the key board each time FORTH is booted. But there is a better method: Let the disk do it for you! To begin with we'll use the simple - and later on a more elegant - way.

(If you haven't made up an overlay yet, better do it now, else editing isn't going to be easy. Programming in FORTH is done by editing SCREENS and the various editing functions are made a lot easier if you can refer to the overlay.)

So boot your FORTH disk again and when the MENU shows up, enter either -EDITOR or -64SUPPORT. Now get out your manual and go to Appendix I (Contents of the Disk) and look at SCREEN 3. This is the one that gives you the first inkling that something is going on by displaying "BOOTING". So you get an idea of the way FORTH works, let's scan its content before going on :

Line 0: The parenthesis ( ) act like a REM in Basic, so we see that it is called the Welcome Screen. GOTOXY is like DISPLAY AT, note the coordinates 0 0 preceding it.

Line 1: Forget the BASE-R for now, but let's do someting with HEX. From your keyboard enter
HEX 83C2 DECIMAL .

Don't forget the period, actually a FORTH WORD called DOT. (Look up each word in the GLOSSARY) What did you get ? -31806 is correct. In plain English line 1 states: Switch to BASE 16, put 10 (16) on the stack, and C! (C-STORE, see page 17, Glossary) it at 83C2. This is how FORTH does the CALL LOAD for FUNCT-Quit Off. (You have seen that one before!)

Line 2: DECIMAL returns us to Base 10, ignore the (84 LOAD), 20 LOAD loads SCREEN 20 (look at scr # 20 and you'll see that it's the menu which appears at boot time. 16 SYSTEM is CALL CLEAR (more about System Calls later) and finally MENU displays the menu.

Take a moment to digest this, as it gives some idea as to how FORTH works. The command 20 LOAD booted scr # 20 at which time a new FORTH WORD was compiled (see scr 20, line 1). MENU is now part of the DICTIONARY. Anytime MENU is invoked, FORTH looks it up and executes it. Try it, enter MENU. You get the menu and 'ok'. If you enter something FORTH can't find you'll see a '?', sometimes followed by an error message (see Appendix H). Most mistakes made by beginners are simple ones, such as missing spaces, colons or semicolons or a LOAD OPTION not booted.

OK, back to the Welcome Screen. But now let's put it on display. Enter 3 EDIT and watch it come up. Skip to line 4 and note that here we have the menu words defined, i.e. : -EDITOR 34 LOAD ; etc. The first word after a ':' is the new word being added to the dictionary. Any words that follow will be executed, provided FORTH can find them in the dictionary. The definition ends at ';'. Now move the cursor down to line 12 and type

-EDITOR (or -64 SUPPORT) enter. Are you surprised that nothing happened (except the cursor moved to the start of the next line)? That's because you are in the EDIT mode. If you are sticking with the normal Editor type in the number which you selected with the SEE experiment as you display color followed by 7 VWTR. If you chose the 64-column Editor dont' bother, type : COLD TEXT COLD ; instead.

Hit the ESCAPE key (F-9) to get out of EDIT. Your additions to scr 3 are NOT actually on the SCREEN but in a buffer and you must enter FLUSH before going on.

Remember that every time you EDIT a SCREEN you must FLUSH, otherwise all your efforts will be for nought.

So let's check if your edit was successful. Enter COLD. This word is like NEW except you don't have to do anything else, FORTH will re-boot. (It'll take longer now because you are booting the editor also.)

Now let's recap:

You have 'edited' SCREEN 3 so it boots your editor and sets up the screen color for you. This was done while in the EDIT mode. You have also worked in the 'interactive' mode when you defined the word SEE to determine your color choice. In this mode you can try out your definitions before you use them in a program. You'll find this to be tremendously helpful because unlike BASIC there is no need to go to RUN and see what happens and then finding the line which needs to be changed.

Having worked my way into TI-FORTH the hard way, I will leave you with a few suggestions which I feel will be helpful:

Look up each new word in Appen. D of the manual. See how it is defined.

Mark the chapters and appendices in your TI-FORTH manual for easier access to them. You'll be using it frequently because - even though it may not seem so at first - it DOES contain a lot of information.

Get a FORTH book, preferrably Leo Brodie's STARTING FORTH. It is sold in many bookstores/software houses. The manual (Appendix C) explains the differences between fig-FORTH, which Brodie uses, and TI's implementation of it.

Though it may read like Greek, scan through the manual. As we go along you might just remember having seen something that rings a bell. (Finding it again may be something else!)

If you have any problems, feel free to call me at 277-4437. I am usually

TI TUTORIAL #2

by ROBERT COFFEE

```
+--------------------------------+
|Communications Register Unit 8K|
+--------------------------------+
```

Let's run down the CRU again.

>0000-03FE	CRU TMS 9901 space, required.
>0404-10FE	For test equipment use on production line.
>1100-11FE	Disk Controller.
>1200-12FE	Modem.
>1300-13FE	Primary RS232, serial ports 1 & 2 and parellel port #1.
>1400-14FE	Unassigned.
>1500-15FE	Secondary RS232, serial ports 3 & 4 and parellel port #2.
>1600-16FE	Unassigned
>1700-17FE	Hex-bus (tm).
>1800-18FE	Thermal printer.
>1900-19FE	EPROM programmer, something that TI planned but never came out wi
>1A00-1AFE	Unassigned
>1B00-1BFE	Unassigned
>1C00-1CFE	Video Controller Card.
>1D00-1DFE	IEEE 448 Controller Card,apparently something else that TI didn't release.
>1E00-1EFE	Unassigned
>1F00-1FFE	P-Code Card.

```
+-------------+
| VDP RAM 16K |
+-------------+
```

>0000-02FF SCREEN IMAGE TABLE (.75K)
This portion of VDP Ram contains the characters that you see on your screen. Hex 0000 is the character in the top-left corner of the screen. The ascII values have offset value of >60.

>0300-036F SPRITE ATTRIBUTE TABLE (.1K)
This table holds the information for all 28 sprites.
eg. position(dot row, dot column), character number, and its color.

>0370-077F PATTERN DESCRIPTOR SPRITE PATTERN TABLE (1K)
Contains the patterns for characters sprites.
eg. address for the space is (768+8=1024).

>0780-07FF SPRITE MOTION TABLE (.12K)
This holds row and column velocities for all 28 sprites and it used by the Interrupt routine in console ROM. The routine executes 60 times a second(or 60 Hertz)and since it is interrupt driven it will use the values i this table to update the Sprite Attribute Table. Each sprite uses 4 bytes. One for row velocity, one for column velocity, and 2 for the system to use.

>0800-081F COLOR TABLE (.03K)
This portion contains the foreground and backround color information for each character set. The definition for each color uses one byte, bytes 0-3 for foreground and 4-7 for backround. there are 32 bytes in the table.(Sets 1-32). Sets 1-3 aren't used by the 'COLOR' statement. Set 4(in table) is character set 0, set 5 is 1, etc. up to set 18(for table) 14 for character set. Sets 19-32 aren't used by the 'COLOR' statement in Extended BASIC.

>0820-35D7 DYNAMIC MEMORY SPACE (11.5K)
This holds your program and other things like PAB(Peripheral Access Block), strings, symbol table, numeric value table, the line number table(for finding the lines of your program thats in the crunched format).Your BASIC program is loaded from >35D7(bottom) and up.Lines appear as they as typed in, not in the order of line numbers (like 100,110,120).

>35D8-3FFF FILE BUFFERS (2.5K)
CALL FILES(n) will change this starting address but with CALL FILES (3) it start repectively at >35D8. If the power up routine finds a disk controller then the computer will automatically reserve this this space for drive control, file allocation, and data buffering.

```
+-----------------+
| Console GROM 18K |
+-----------------+
```

There are 3 GROM chips in our consoles. Each has 8K of space but only 6K is used. The difference between ROM and GROM is that GROM automaticallt increments itself everytime it is accessed.GROM is also written in GPL(Graphics Programming Language),which TI wrote themselves. Here are those 3 GROM chips:

GROM 0 >0000-17FF The title screen power up routine, title screen character set, standard character set(Upper Lower casd), cassette DSR messages and the trigonometric functions.

GROM 1 >2000-37FF Vector tables for BASIC, the error messages, and part of the BAISC interpreter.

GROM 2 >4000-57FF Part of the BASIC interpreter,the reserved word list and their associated token values.

GROM chips 3-6 (24K)are in the Extended BAISC cartridge and contain the following:

GROM 3 >6000-77FF X/BASIC vector tables, the error statements for X/BASIC and part of the X/BASIC interpreter.

GROM 4 >8000-97FF Part of the X/BASIC interpreter.

GROM 5 >A000-B7FF Part of the X/BASIC interpreter.

GROM 6 >C000-D7FF Part of the X/BASIC interpreter, the reserved word list and their associated token values.

```
        +-------------------------------------------------+
        |    Video Display Processor RAM for Extended BASIC |
        |                                                 |
        |                VDP , a complete look.           |
+--------------------------------------------------------------------+
|>0000          VDP SCREEN IMAGE TABLE                     768 bytes |
|                                                                    |
|     each screen location takes up 1 byte, the character            |
|     value at each location is offset by >60.                       |
|                                                                    |
|     LOCATION=COL+32*(ROW-1)                                        |
|                                                              >02FF |
+--------------------------------------------------------------------+
|>0300          SPRITE ATTRIBUTE TABLE                     112 bytes |
|                                                                    |
|     Each sprite takes up 4 bytes. (room enough for only 28)        |
|     These for bytes consist of vertical postion -1, horizontal     |
```

>0300 SPRITE ATTRIBUTE TABLE 112 bytes

Each sprite takes up 4 bytes. (room enough for only 28)
These for bytes consist of vertical postion -1, horizontal position, character # + >60, clock bit, color.

>036F

>0370 EXTENDED BASIC SYSTEM BLOCK

>0371	Auto Boot (needed flag)
>0372	Line to start execution at
>0376	Saved symbol table "GLOBAL" pointer (used with subprograms).
>0378	Used for CHR$
>0379	Sound blocks
>0382	Saved program pointer for continue and text pointer for break
>0384	Saved buffer level for continue
>0386	Saved expansion memory for continue
>0388	Saved value stack pointer for continue
>038A	ON ERROR line pointer
>038C	Edit recall start address
>038E	Edit recall end address
>0390	Used as temporary storage place
>0392	Saved main symbol table pointer
>0394	Auto load temp for inside error
>0396	Saved last subprogram pointer for continue
>0398	Saved ON WARNING/BREAK bits for continue
>039A	Temp to save subprogram table
>039C	Same as above but used in subprograms
>039E	Merged temp for PAB (Peripheral Access Block) pointer
>03A0	Random number generator seed 2
>03A5	Random number generator seed 1
>03AA	Input temp for pointer to prompt
>03AC	Accept temp pointer
>03AE	Try again (used when you input a string instead of a number)
>03B0	Pointer to standard string in VALIDATE
>03B2	Length of standard string in VALIDATE
>03B6	Size temp for record length. Also temp in relocating program
>03B7	Accept "TRY AGAIN" flag
>03B8	Saved pointer in SIZE when "TRY AGAIN"
>03BA	Used as temp storage place
>03BC	Old top of memory for relocating program / temp for INPUT
>03BE	New top of memory for relocating program
>03C0	Roll out area for scratch pad RAM when certain operations are performed
>03DC	Floating point sign

>03EF

>03F0 PATTERN DESCRIPTOR TABLE / SPRITE DESCRIPTOR TABLE 912 bytes

Each character take up 8 bytes. There are 114 characters here. They are numbered from 30 to 143.

>077F

>0780 SPRITE MOTION TABLE 128 bytes

Each sprite takes up 4 bytes. These 4 bytes contain the vertical velocity, horizontal velocity, & the last 2 are for system use.

>07FF

>0800 COLOR TABLE 32 bytes

Each character set requires only 1 byte. This byte is broken up into the foreground & backround.

>081F

>0820 CRUNCH BUFFER 160 bytes

This area of VDP is used when the system needs to crunch ASCII values into token codes.

>08BE

>08C0 EDIT / RECALL BUFFER 152 bytes

What you type in at the command line is stored here.

>0957

>0958 VALUE STACK 16 bytes

Used by these ROM routines : SADD, SSUB, SMUL, SDIV, & SCOMP

>0967

```
+--------------------------------------------------------------------------+
|>0968                                                         11888 bytes |
|        The items in this area move according to the size of the         |
|        crunched program & the system always reserves 48 bytes of        |
|        area.                                                             |
|                                                                          |
|        The SYMBOL TABLES are generated during the pre-scan peroid        |
|        after you type RUN. The strings are placed into memory when      |
|        they are assigned.                                                |
|                                                                          |
|        WITHOUT MEMORY EXPANSION:                                         |
|                                  -STRINGS                                |
|                                  -DYNAMIC SYMBOL TABLE & PABS            |
|                                  -STATIC SYMBOL TABLE                    |
|                                  -LINE NUMBER TABLE                      |
|                                  -PROGRAM SPACE(crunched program)        |
|                                                                          |
|       WITH MEMORY EXPANSION:                                             |
|                              -STRINGS                                    |
|                              -DYNAMIC SYMBOL TABLE & PABS                |
|                              -STATIC SYMBOL TABLE                        |
|                              -Numeric values, line number table,         |
|                               & program space are moved into             |
|                               High-memory expansion( >A000 )             |
|                                                                   >37D7  |
+--------------------------------------------------------------------------+
|>37D8          DISK BUFFER AREA [ default 'CALL FILES(3)' ]    5 bytes    |
|                                                                          |
|          >37D8 Validation code for the disk controller DSR ( >AA )      |
|          >37D9 Points to TOP of VDP memory  ( >3FFF )                    |
|          >37DB CRU base identification                                   |
|          >37DC Maximum number of OPENed files  ( >03 default )           |
|      -------------------------------------------------------------       |
|             File Control Block for 1st file OPENed         518 bytes     |
|                                                                          |
|          >37DD Current Logical record offset                             |
|          >37DF Sector number location of File Descriptor Record          |
|          >37E1 Logical Record Offset(used woth VARIABLE files only)      |
|          >37E2 Drive number(using the high order bit)                    |
|        File Descriptor Record(brought from the disk 256 bytes)           |
|          >37E3 File name                                                 |
|          >37ED Reserved  ( >0000 )                                       |
|          >37EF File status flags(file type & write protection)           |
|          >37F0 Max number of records per Allocation Unit(1 AU=1 Sector)  |
|          >37F1 number of sectors currently allocated (256 byte blocks)   |
|          >37F3 End of File offset within the last used sector            |
|          >37F4 Logical record length                                     |
|          >37F5 # of FIXED lenght records OR # of sectors for VARIABLE    |
|                length                                                    |
|          >37F7 Reserved (>0000 >0000 >0000 >0000)                        |
|          >37FF Pointer blocks                                            |
|          >38E3 Data Buffer area(256 bytes)                               |
|      -------------------------------------------------------------       |
|        >39E3 File Control Block for 2nd file OPENed (6 b)   s 518 bytes  |
|                                                                          |
|          >39E9 File Desriptor record (256 bytes)                         |
|          >3AE9 Data Buffer area  (256 bytes)                             |
|      -------------------------------------------------------------       |
|        >3BE9 File Control Block for 3rd file OPENed (6 b)     518 bytes  |
|                                                                          |
|          >3BEF File Descriptor record (256 bytes)                        |
|          >3CEF Data Buffer area  (256 bytes)                             |
|                                                                   >3DEE  |
+--------------------------------------------------------------------------+
|>3DEF               VDP STACK AREA                            252 bytes   |
|                                                                          |
|                                                                   >3EEA  |
+--------------------------------------------------------------------------+
|>3EEB              DISK DRIVE INFO                              4 bytes   |
|                                                                          |
|          >3EEB Last drive number accessed                                |
|          >3EEC Last track access on drive #1                             |
|          >3EED Last track access on drive #2                             |
|          >3EEE Last track access on drive #3                             |
|                                                                   >3EEE  |
+--------------------------------------------------------------------------+
|>3EEF         not used by the 4A , it might have been used      6 bytes   |
|              by the 4  (?)                                               |
|                                                                   >3EF4  |
+--------------------------------------------------------------------------+
|>3EF5              VOLUME INFORMATION BLOCK                   256 bytes   |
|                                                                          |
|       An exact copy of sector >0 from the disk last accessed.            |
|                                                                   >3FF4  |
+--------------------------------------------------------------------------+
|>3FF5              FILE NAME COMPARE BUFFER                    11 bytes   |
|                                                                          |
|       Contains disk number & 10 character file name from last           |
|       access.                                                            |
|                                                                   >3FFF  |
+----------------------------------     -----------------------------------+
```

A LOOK AT TI BASIC

by R. A. Green

As you may already know, TI BASIC programs can be executed from GROM/GRAM as well as from VDP RAM. Several TI modules (ie Personal Record Keeping) contain TI BASIC programs. Now that there are some GRAM devices such as MAXIMEM and GRAM KRACKER you have the capebility of putting your favorite TI BASIC program (not Extended BASIC) into GRAM for execution. TI BASIC programs executed from GRAM have the whole VDP RAM available for data storage thus allowing larger arrays, more variables, etc.

There is one little problem -- there has to be, otherwise this article wouldn't be interesting. I have found that there is a bug in TI BASIC when processing a RESTORE statement that refers to the line number of a DATA statement. TI BASIC forgets, in this instance, to check if the program is in GROM/GRAM and usually ends up doing a restore to the first DATA statement in the program.

This is the case on my console at least. Heiner Martin in his execellent book, "TI 99/4A INTERN" says he knows of no variations in the TI BASIC GROMS, so I imagine it is the case on your console also. The following is a short TI BASIC program which will show the bug. The program works correctly when executed normally from VDP RAM. It fails when executed from GRAM.

```
1 PRINT "SHOULD PRINT 220,110,100":::
100 DATA 100,"ONE HUNDRED"
110 DATA 110,"ONE ONE OH"
130 RESTORE
140 READ N,N$,N,N$,N,N$
150 PRINT N,N$
160 RESTORE 110
170 READ N,N$
180 PRINT N,N$
190 RESTORE 100
200 READ N,N$
210 PRINT N,N$
220 DATA 220,"TWO TWO OH"
230 FOR I=1 TO 2000
240 NEXT I
```

There is a fix -- there has to be, otherwise this article would only be a little interesting. The GRAM device I have is Miller's Graphics GRAM KRACKER so I'll give the fix using it. Miller's Graphics supplies with the GK routines to put a TI BASIC program into GRAM. I will not duplicate their directions for doing it here. Just remember to turn the LOADER switch to OFF when executing a TI BASIC program.

In order to apply this fix you must have a GK that has GRAM 0 and GRAM 1-2 (ie for the console GROMS). The steps to fix the bug are listed below.

1. Copy the console GROMS 1 and 2 to disk.
2. Load the console GROMS 1 and 2 into the GK.
3. Locate the code with the bug.
4. Patch a branch to the fix over top of the bug.
5. Patch in the fix.
6. Test the fix.
7. Save the fixed GRAM 2 to disk.

Steps 1, 2 and 7 are fully described in the GK manual. Steps 3, 4 and 5 are done using option 5 "Edit Memory" from the GK menu.

When the Edit Memory screen comes up, switch the LOADER switch to OFF, the GRAM 1-2 switch up and the W/P switch to BANK 1 (ie allow writing). Press FCTN = to change the display to HEX. View memory at address G420E. The memory display should then begin with:

```
C5 4A B0 36 42 1F D5 ...
```

The first four bytes are the instruction in error. They represent the GPL instruction:

```
DCH    VDP*8336,@834A
```

Which is comparing the line number in the line number table to the line number in the RESTORE statement. This instruction assumes the line number table is always in VDP RAM.

Now we will patch a branch over this instruction to our fix. Press FCTN 9 to put the cursor in the memory window and type the following three bytes begining at address G420E.

```
     05 58 00
```

Which is a GPL branch instruction to address G5800 which is where we will put our fix.

Press FCTN 9 again to exit from the memory window and view address G5800.

The memory window will display a lot of junk because TI GROMS are 6K bytes and GK GRAMS are 8K bytes. If we needed it we have 2K bytes for fixes. Again press FCTN 9 to enter the memory window and begin typing the fix at address G5800. The fix is shown below.

```
     8E 80 89
     58 0E
     C5 4A B0 36
     42 1F
     05 42 14
     33 00 02 00 00 00 36
     C5 4A 00
     42 1F
     05 42 14
```

Which are the GPL instructions for:

```
          CZ     @8389               Test if pgm in GROM
          BR     GRAM                 Branch if yes
     * Program in VDP RAM
          DCH    VDP*8336,@834A  Original instruction
          BR     421F                 Instruction after original
          B      4214                Continue original code
     * Program in GRAM
     GRAM MOVE   2,G@0(@8336),@8300 Get line # from GRAM
          DCH    @8300,@834A     Compare line numbers
          BR     421F                 Instruction after original
          B      4214                Continue with original
```

Now test the fix by running the sample program normally in VDP RAM and then running it from GRAM.

BROWSING THE LIBRARY
--with STEPHEN BRIDGETT

Disk sales have been brisk since we re-introduced the 'DISK OF THE MONTH', and except for a batch of bad copies in Nov., things are running real smooth.

At the monthly meeting, the members voted to purchase 500 disks to back up the software disk library and a further $500.00 to have disk jackets made up with the club name and insignia embossed on them. When new disks are to be purchased the embossed jackets will be sent to the manufacturer. Members are encouraged to promote pride and notoriety of our club by purchasing their disks with the embossed jackets. The extra charge is very minimal.

Many offers to help with the LIBRARY have been gratefully received. Most activities will take place early in '87 and people will be contacted. One job that will need to be done is typing in programs gleamed from various sources, Micropendium etc. There are some excellent programs out there just waiting for someone to spend an hour or so doing a little finger work. If just a few people offer to type one program a month, the library can grow in leaps and bounds. - ANY TYPISTS ?

PLEASE use the SUGGESTION BOX. This is your easiest most efficient feedback to the executive and committee people. What do you want from YOUR SOFTWARE LIBRARY ? Or call me at 521-3631, there is an answerering machine on this line.

SEE YOU AT THE MONTHLY.....STEPHEN

momin PRINT

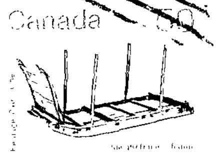
OFFSET PRINTING • LAYOUT • PHOTOCOPYING

• Quality printing at low prices
• Special prices on photocopies

OPEN
MON 8:00 A.M.
TO 6:00 P.M.
SAT

837-1844
ORLEANS

2451 St-Joseph Bl Orleans
Lower Level in Le Centre
K1C 1E7



OTTAWA, ONT

FROM

P.O. BOX 2144, STATION D, ~~OTTAWA~~
*** ONTARIO, CANADA K1P 5W3 ***

Tom Hall
3-107c Clinical Sciences Bldg.
University of Alberta
Edmonton, Alta.
T6G 2G3