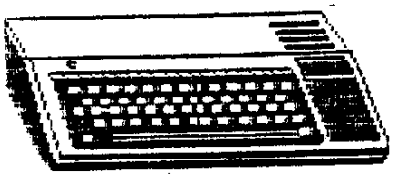
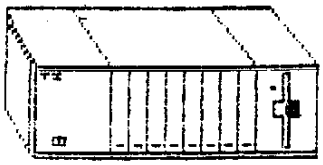
NEW JUG NEWS



NEW JERSEY USERS GROUP



Vol.5 No.4 *Monthly Publication of the New Jersey Users Group* APRIL 1986

****

MEETING

****

APRIL 14 MONDAY 7:00

7:00 - 8:00 BASIC SIG WILL MEET

8:00 - GENERAL MEETING--REPORT ON T.I.C.O.F.F
POTTER & SHULDMAN on analog and digital conversions

****

The New Jersey Users Group meets on the second Monday of each month in the Metuchen Library. Dues are $15 per year.

****

OFFICERS

President.....	Steve Citron..	686-5619
Vice-Presidents.....	John Bonito...	653-2637
	Bob Costello..	663-4512
	Mel Gary.....	828-5407
	Bob Guellnitz.	382-5963
Secretary.....	Carol Sudol...	494-3781
Treasurer.....	Marv Shuldman.	821-8158
Newsletter Editor....	Mel Gary.....	828-5407
Software Library.....	Dave Green....	463-9133
	Leon Green....	828-2435
Advanced Prog. Sig...	Jay Holovacs..	356-3150
Basic SIG.....	Bob Haefeli...	572-2828

****

SUBSCRIPTION FREE WITH PAID MEMBERSHIP, TO USERS GROUPS AND SELECTED VENDORS

****

MAY 1986

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
				1	2	3
4	5	6	7	8	9	10
11	12 GENERAL MEETING	13	14	15	16	17
18	19	20 STEERING COMMITTEE	21	22	23 NEWSLETTER DEADLINE	24
25	26	27	28	29 ADVANCED PROG.	30	31

MODEMS FOR SALE

[illegible] - $20 New!. TI CABLE NEEDED.

[illegible] - $30 Used. TI cable included.

Contact Mel Gary (201)828-5407

President:

Steve Citron
981 Townley Ave.
Union, NJ 07083

Send Dues To:

Marv Shuldman
28 Tyndall Rd.
Kendall Pk., NJ 08824

Write For Application:

Bill Dubrow
21 Seaward Ave.
Metuchen, NJ 08840

NOTE WELL!

Anyone with software to be returned or donated to NEW JUG's Software Library should contact either Leon Green (828-2435) or Dave Green (463-9133). The sucess of the library depends on you!

BOSTON FAYUH

Here is a quick report on the New England Fayuh, as seen by one of the organizers (me):

The new computer-on-a-card was there and was working in the morning! It was in a wire wrap version that crashed by the afternoon MYARC presentation in the auditorium. A lot of people saw it, and it certainly was not vaporware. The first 20 PC boards have been on order for about two weeks. If they had been delivered on time, one of them would have been at New England, not the wire wrap version. They should be ready real soon now (or even sooner). For the wire wrap version to survive the trip at all, was perhaps fortunate. They aren't made to travel! Mass production by sometime in June, if all goes well. The card will include four ports - one for the IBM type keyboard, one for mouse, one for joysticks, and one for the monitor. The card will not support your color TV as the 4A does. If you want color, you must use a composite or RGB monitor. RGB will be necessary if you are to take full advantage of the high-res graphics or 80-column text capability and still have color. The present TI monitor does not have the resolution required for 80 columns!

As far as XBII is concerned, version 2.1 was not quite ready yet. The MIN/MAX is debugged and the integer math is working, but the user generated DEFs and SUBs are still giving some trouble. Lou Phillips made it quite clear that purchase of version 2.0 now entitles you to 2.1 without additional charge (except mailing?). If one person in an area gets the update on disk, it can be passed on to others, since the disk alone is worthless without the eprom on the 128/512k card. There is no good reason to wait to buy the present version, unless you just don't want to be bothered with an incomplete version for now. I bought it this week to become familiar with the new functions, even though I knew it would not run most of my own software yet. I expect it will be ready within the month.

Lou also clarified some comments made at TICOFF. The finished computer will have a new operating system, with perhaps some GEM similarities. (Paul Charlton says GEM-like displays aren't and operating system, they're an I-O device.) XBII will be version 3.0 with new graphic modes, not the currently expected 2.1. It may not even be called "extended BASIC" any more, but given a new BASIC name. He didn't say so, but I expect that not all that will be done for the initial production run, if it's in June, but both BASIC and the O.S. will be on disk anyway and can be upgraded as they are improved.

Barry Traver gave a superb inspirational talk about the TI family community and how it pulls together to support us all. Jim Horn talked about the role of telecommunications in the TI world and in general, and then he called JZ out of the audience for a few words - ostensibly about Compuserve. Well, JZ showed us all he is a very fine public speaker and an astute analyst of sociological trends. Remember that, future faire organizers! We all knew JZ could talk on a keyboard, but he is even better up in front of an audience! Paul Charlton, as usual, enthralled the technically sophisticated. His audiences are small, because his level is beyond a lot of us, but those that stay really listen. Chris Bobbit spoke of some exciting new things under investigation by ASGARD, particularly Richard Rosen. Their work could lead to an alternative type of new computer - bare bones in execution, but very fast and very capable. Prototypes will be developed, but there is no commitment to market it, unless it looks like there really is a demand for it.

All-in-all, the New England Fayuh seemed to run very well. It certainly turned a good profit for the clubs participating, and the dealers seemed to do quite well, too. The headcount was about 600 people, with about half of the cars in the parking lot from out of state. It was done without any paid advertising, but every other source we could think of was exploited and the word reached the people it was intended for. Peter Hoddie's phone was ringing every few minutes for the last three days before the Fayuh from people who had heard about it and wanted directions.

(d/l from CompuServe by sel gary)

PROGRAM FILES

by R.A. Green
Ottawa 99/4 User's Group

There are seven different ways to store programs in the TI 99/4A. In this article we will have a look at each of these seven forms and at how they are used.

Everyone is familiar with the form used by TI BASIC to store programs on cassette or disk. It's identified as "PROGRAM" in the disk catalog. It is created or stored by the BASIC "SAVE" command and loaded by the BASIC "OLD" command. This is the only way that TI BASIC uses to store your programs.

EXTENDED BASIC can, and usually does, use the same form as TI BASIC to store programs. There are, however, two other forms that XB uses. Both these can only be used to store programs on disk.

If you have the 32K Memory Expansion, you can write an XB program which is too large to store in the usual format. XB will store these large programs in an "INTERNAL VARIABLE 254" file. The usual "SAVE" and "OLD" commands are used to store and load these programs.

The third form used by XB is the "merge format" stored in a "DISPLAY VARIABLE 163" file. This form is created when the "MERGE" option is specified on the "SAVE" command. The beauty of merge format is that when it is loaded it does not necessarily overwrite the program in memory. The "MERGE" command does just that - it merges the new program (or program segment) with the program in memory according to line numbers.

Now, we get to the good stuff, Assembler Language programs. There are three forms for an assembler program: tagged object, compressed tagged object, and memory image.

Tagged object is stored in a "DISPLAY FIXED 80" file on disk only. All program data is in hexadecimal so that it can be edited by the E/A editor. Tagged object can be loaded via "CALL LOAD" in XB, option 3 on the E/A menu, option 1 on the MM menu or "CALL LOAD" in TI BASIC when either the E/A or MM module is used. The program can be "absolute" or "relocatable". An absolute program must always be loaded at the same place in memory. A relocatable program can be loaded any place in memory. A tagged object program may have references to other programs or subroutines. The loader will resolve these external references, except for the XB loader.

Compressed tagged object is very nearly the same as tagged object except that the program data is stored as bytes rather than as hexadecimal digits. Compressed tagged object loads faster than regular tagged object as you would expect. The XB loader cannot load compressed object.

Tagged object, in either form, is produced by the Assembler when it assembles a source program.

The "memory image" form of assembler program is the most compact and the fastest loading. It can be stored on cassette or disk. It is identified as "PROGRAM" in the disk catalog (just like a BASIC program). Memory image programs can be loaded by option 5 on the E/A menu or option 3 on the TI Writer menu (and I assume, by Multiplan although I have never tried since I don't have Multiplan). It should be noted that there is one slight but important difference between how the E/A calls a memory image program and how TI Writer doe it. TI Writer blanks the screen just before calling the program and the E/A does not. This means the program must turn the screen back on or nothing will show. Memory image programs are created by a Utility program (one is provided on the E/A disk).

A PROGRAM file, containing an Assembler memory image or a BASIC program, can be read or written to any input/output device with a single I/O operation. This is one of the reasons they load so quickly.

There is a restriction on the size of an Assembler memory image program of 2400 bytes (9216 decimal). However, the E/A and TI Writer modules will load multiple memory image files to make a program of any size. They use the convention that the file name of the second and following files is obtained by incrementing the last digit or letter of the previous file name. For example, the TI Writer editor consists of two memory image files: EDITA1 and EDITA2.

As a matter of interest, the ADVENTURE, TUNNELS OF DOOM, PERSONAL RECORD KEEPING, STATISTICS, and PERSONAL REPORT GENERATOR modules use a memory image or "PROGRAM" for their data bases. The fact that memory images can be saved or loaded with a single I/O operation makes them attractive for such uses.

A lot of the Assembler language games that are circulating around are in memory image format so let's look closer at them. Assembler memory image files have a three word header followed by the data to be placed in memory. The three header words are:

(1) This word is a "flag". If it is not zero (i.e. FFFF) then this file is not the last in a multi-file program. For example, the flag word for EDITA1 is FFFF indicating that there is another file called EDITA2; the flag word for the EDITA2 file is 0000 indicating it is the last file and there is no EDITA3.

(2) This word is the length of the memory image in bytes, including the six byte header.

(3) This word is the CPU memory address where the memory image is to be loaded.

Execution of a memory image program always begins at the first byte of the first segment loaded.

Finally, the seventh form for programs. This form is created and loaded by "EASY BUG" of the Mini Memory module. It can be written only to cassette and is a memory image, but is slightly different from the E/A memory image file. The EASY BUG memory image program can consist of only one segment. The header on the EASY BUG format is two words, as follows:

(1) This word is the CPU memory address at which the memory image is to be loaded.

(2) This word is the length of the memory data, not including the four header bytes.

If this whole thing is too complicated - maybe a table showing all of the options will help.

FILE TYPE	CONTENTS	MODULE	DSK	CS
PROGRAM	BASIC Program	Console	YES	YES
PROGRAM	BASIC Program	XB	YES	Yes
INTERNAL V 254	BASIC Program	XB	YES	NO
DISPLAY V 163	MERGE Program	XB	YES	NO
DISPLAY F 80	Tagged Object	XB	YES	NO
DISPLAY F 80	Tagged Object	E/A	Yes	NO
DISPLAY F 80	Tagged Object	MM	YES	NO
DISPLAY F 80	Compressed Object	E/A	YES	NO
DISPLAY F 80	Compressed Object	MM	YES	NO
PROGRAM	E/A Memory Image	E/A	YES	YES
PROGRAM	E/A Memory Image	TIW	YES	YES
PROGRAM	MM Memory Image	MM	NO	YES

(reprinted from THE DATA BUS - the Delaware Valley Users Group's Newsletter)

PILOT-99

by Bill Harms

Programmed Inquiry Learning or Teaching

Here is a brief description by Bill Harms of another programming language available for our TI99/4A.

Although I've just spent a few days learning about PILOT, I can really write a useful, enjoyable program. This language is EASY. It doesn't have many of the capabilities of TI BASIC, but it does have others not found in even TI Extended Basic.

Thomas P. Weithofer sent me the program PILOT 99, and documentation. He developed this TI99/4A version with help from Texas Instruments, Cin-Day Users Group, and Xavier University professionals. It's copyrighted 1985 by Thomas Weithofer and portions of the manual are by permission of Texas Instruments. It is a public domain package that costs one only about $10.00 plus 2 SSSD disks. What a great value! Thomas' address is 1000 Harbury Drive, Cincinnati, Ohio 45224. Many TI User Groups will have a copy in their library.

PILOT was largely created by John A. Starkweather, Ph.D. at Univ. of Cal. in San Francisco starting in 1962. In 1973 national standards were developed for the basic commands (only 8) and syntax, and now one can get a version of PILOT for most personal computers. It was developed on a small computer to be able to function completely on a small computer. Dr. Starkweather wrote a short book, which I've found to be the perfect guide. It's called, "A User's Guide to PILOT" and published by Prentice-Hall, Inc. at Englewood Cliffs, New Jersey 07632. I ordered it at the local B. Dalton Bookseller.

I would evaluate the TI version as one of the best teaching aids available in the world of software, since it's easy to write programs and offers most all of the features that make a lesson useful and enjoyable. The only feature I would like to see added is that of Speech.

PILOT 99 seems to be written in TI-Forth and thus a program can run pretty fast. It shows the power and versility of TI-Forth. While one is thus limited to a small program running at one time, one can run programs quickly with each drawing needed data from files the other programs have created.

To use the version of PILOT 99 that I got, you will need TI's Editor/Assembler cartridge, expanded memory, a disk system, and a word processor that can create display/variable-80 (text) files. You would write the program in the word processor just like the big computers/software use., which is nice in some ways since with one like TI-Writer you've got a full screen editor and other useful commands available. Then you would fire up the Editor/Assembler and use the Load and Run Option, entering: DSkn.PILOT. When it is loaded enter the file name of the program you created with the word processor. The PILOT 99 software will run the program until it finds an error in which case you get an error message at that point. Thomas Weithofer says there is also a version one can use out of TI Extended Basic.

PILOT 99 adds many commands beyond the basic PILOT set. You have all the normal TI Extended Basic Sprite Commands, which provide great enjoyment to a user and liven the presentation of any subject matter. Thomas has also added the Joystick commands, TI's character graphics commands with color, real live Bit Map Graphics ie, Draw Circle, and Mass Storage device commands for files usage.

The manual is excellent, all 70 pages of it (on disk). Each command is described and an example given in a program context. However, it says that data files are Internal Fixed 80 Relative Update, but the file I got when writing data out to disk was Display Fixed 80. To help me use the manual I created a kind of Table of Contents and Index.

Bit Map graphics are easy to create and are displayed in the top 2/3rds of the screen with the bottom 1/3 reserved for full sized text. In the top 2/3rds graphics area you can also display text, but it will be smaller(64 characters per line). The command for Draw Rectangle is: DR: row1, cla1, row2, cla2, ie. DR:50,50,100,100 will draw a rectangle with the top left at position 50,50 and the bottom right at 100,100. Then

one could use the command "T:Thats a rectangle, folks!" to produce the message at the bottom of the screen.

Better yet, to describe the language, you could ask the computer operator ie. student some questions about the rectangle. Here's a really short program to illustrate.

By the way, PILOT doesn't use line numbers. It's like LOGO, LISP, and some other advanced languages in this respect. One uses labels and subProgram like techniques to structure the program and direct the flow of action.

```
R: Remark only - prog. to demo a Q A.
IG:
DR: 50,50,100,75
TG: 1,5,shape is 50 by 25 units
T: how high is that rectangle?
A: $A
M: 50,50 UNITS
TY: That's perfectly correct
TN: Nope, thats not just right
T(#A=25): You were thinking of the WIDTH
T: press any key to proceed
**********************************
```

R: is for a REMark
IG: is to Initialize Graphics.
TG: puts the text at row,column used
T: is to Type something to the screen.
(TP: is to Type to Printer)
A: is to Accept an Answer
M: is to Match to the following possible strings
each seperated by a comma
TY: is to Type only if the previous Match was True
TN: is to Type only if the previous Match was Not-true
T(#A=25): is to Type only if the expression is True
(here users answer of 25 would be true)

Instead of the TY: and TN: we could have used a command- JM:*LABEL for Jump-on-Match to a label. After the *label would come some testing routine that ended with an E: command to return the program flow to the line following the JM:*label.

We could have used the Match or Jump command- MJ: string-to-match,more. If no match is found to the strings in the statement, the program jumps to the next M: or MJ: statement.

User subroutines are invoked with a simple- "U:*YOUALL" (U:*title). They are also ended with the command- E:.

Problems can be identified with the PR: command, then you can jump to them easily.

You can put the Y or the N or the conditional expression ie,(#A=25) after any of the basic commands.

To save that answer to a disk file we would just add a command- Write Answer- WA: right after the A: in the program above. Earlier in the program you would have the command to open the file- OF: DSK2.FILENAME or some other file and then later would close the file with- CF:.

For math you use the C: (Compute command) with the characters (- instead of the = sign. For example: C: #F(-88 or C: #E(-#G. The first sets F equal to 88 while the second sets E equal to the value of G. All the other TI numeric operators ie. + are available as are the numeric functions such as TAN for Tangent.

PILOT is for easy interaction between the computer and the user. A simple example of it is:

```
T: Enter your name
A: $A
T: Enter an adjective
A: $B
T: Enter a type of animal
A: $C
T: Enter a part of an animal
A: $D
T: Enter a color
A: $E
CH: (this means Clear-Home the cursor)
R: * * * *
T: $A had a $B $C,
T: whos $D, was $E as snow
T: Everywhere that $A went, the $C
T: was sure to follow.
```

There are many other commands in PILOT 99, but most are just like TI Basic or the Sprites in TI Extended Basic. Most are easy to remember and there are only 34 with the 1 or 2 digit code. I've barely scratched the surface in this memo of the many ways the commands can be combined to produce a very enjoyable interactive session of learning or data collection. Dr. Starkweather describes many in his book.

--- E X P L O R E ---

(d/l from CompuServe by mel gary)

The author of PILOT99, Thomas P. Weithofer, died on April 5 at the age of 22. Memorial contributions may be made to the Christ Hospital PULmonary Disease Fund, 2139 Auburn Ave., Cincinnati, OH 45219, or to the Ministry to the Elderly and Sick, Nerinx, KY 40049

ED.: I have a copy of the program if anyone is interested.

C99:BEGINNERS

by Ron Albright

I have been exploring c99 for the TI of late. Written by Clint Pulley (38 Townsend Avenue, Burlington, Ontario, Canada L7T 1Y6) and available as Fairware, the language is a full-featured version of "small c". I have found few limitations with the language (lack of floating-point and math routines are the major ones), and have been able to do some nice routines with the language. Briefly, C is a very popular programming language through which, it has been estimated, 70% of commercial software for other machines is written. So what makes it different? It is a "compiled" language. That means, once you have written your program in c99, you run a companion program called a compiler. The compiler takes your C source code and generates assembly source code. The resultant code can then be run through the TI Assembler to generate object code, which executes just as fast as if you went through the strenuous (to me, anyway) task of writing assembly source code to start with. C is much easier to learn that Assembly language and is efficiently compiled with the c99 compiler. I have seen some programs written with c99 alone (there are a few on Compuserve; a simple text editor and a word-counter for TI Writer files by Warren Agee, a program similar to the TI Writer formatter, and a graphics demo by yours truly) and they are indistinguishable from pure assembly language, because the end-product is just that. If there is any interest, I will address the language more in depth in some more starter-level tutorials. I am no expert,by any stretch of the imagination, but I am learning and plan to spend a great deal of time with the language. Warren Agee, of Livonia, Michigan, has uploaded several tutorials to Compuserve which are mid-level in their scope and excellent in their content and expertise and these will help you as you get further into the language. It is a marvelous programming tool and, hopefully, this simple file will help you get started. Learning a new language is never easy, but it is time we all advanced beyond BASIC and started working in another environment. c99 provides a reasonable alternative. I could never think in reverse, so I gave up on Forth; I am too dense to learn assembly language. Pilot is too slow and requires too many disk accesses. Besides C is used in so many other machines and for so many other applications, it has to be good. Let's begin by seeing what we have to work with.

First, equipment-wise, you need the following: console, monitor, 32k memory expansion, at least one disk drive and controller, the Editor/Assembler package (cartridge or disk version) and, of course, the c99 system disk. A printer is nice (see below) but is certainly not imperative for programming purposes. Ideally, you would have two drives as this makes the work much easier, as does having at least double-sided drives (but ain't that always the case!). If you have double-sided drives, you can save yourself a lot of disk-swapping by, first, of course, making a backup of the c99 system disk and, secondly, copying from the Editor/Assembler disk, the files ASSM1, ASSM2 (the files for assembling source code) and EDIT1 (for the E/A Editor) on to the c99 system disk. But, if you have a single-drive or single-sided system, don't despair...things will work just fine with what you have.

Once you have gathered your tools, you should get a disk directory printout of the c99 system disk. Pulley even provides a disk catalog program on the system disk (called "SD" and running out of E/A 5 on my disk) but it doesn't print to printer). You will notice that there are a long of files in all shapes and "colors" (D/V 80, D/F 80, and PROGRAM files) and we will first go over what is important and what is not. Some of the files you will be using a lot, others seldom if at all, at least to start. Here are some of the files you should have and what they are for. I will list them in order of importance and probably frequency of use.

C99C,C99D,C99E

These are the compiler files. They are the heart and soul of the c99 system. There are PROGRAM image files and are run from Editor/ Assembler option 5. Unlike some PROGRAM image files, these CANNOT be run from option 3 of the TI Writer module. In my brief experiment they could not be loaded from XB with the FUN LOADER from Australia. The first thing I did with these files is rename them to be UTIL1, UTIL2, UTIL3. Then, when you chose the LOAD and RUN option from E/A (option 5), you only have to hit "Enter" and the files will be loaded by that name as a default without typing them in.

CSUP

This file is very important. It is a D/F 80 (which always means it runs from E/A option 3) which must be loaded immediately after you load you completed, assembled program. We will discuss this more later, but suffice it to say that your c99 program will never run if you don't load this file after it and with it.

C99MAN1,C99MAN2,C99MAN3

These are the D/V 80 files that contain the documentation Clint Pulley provides with the c99 system. They are not going to go very far in teaching you how to program in c99. Like the manual TI provided with the TI Forth system, they are simple a brief tutorial on how the different files work, and what they do, what the error messages mean, ect. They are quite adequate for their intended purposes. Pulley tells you up front "This manual assumes a knowledge of standard C or the

availability of a suitable reference." That translates into "If you have never programmed in C, go buy a book!" I will recommend a couple at the end of this piece. Fair enough, Clint! If you have a printer, print these files out for future reference. If not, find a friend who does. You will need a hard-copy of these files.

C99ERRORS

This is a short D/V 80 file that contains a listing of the 30 or so error messages that the compiler will embed in your compiled code when it encounters one. It will only embed the error number. You will have to look in this file to find out what the number means. Print this out also.

C99SPECS

A terribly important D/V 80 file. This short file tells you what c99 supports and, more importantly, what it does not support, when compared to standard C. Why is this important? I have yet to find a book that addresses only "small c", the version of C (more limited than "big C") that c99 is modeled after. All the texts I am aware of cover the full C language. Small c and c99 do not have all the functions of C. When you look at program listings out of these texts, you will quickly become frustrated if you try to type them in verbatim as they are already. Many program statements in C will give you errors in c99. You have to study this file when typing in program listings out of books to avoid these errors. For example, C supports "floating-point" arithmatic; small c and c99 do not. There are other examples covered in this file; print it out. You will need it.

GRF1DOCS

This is the documentation for the graphics routines supported by the current version (1.32) of c99. Print it out.

ERRFIND1

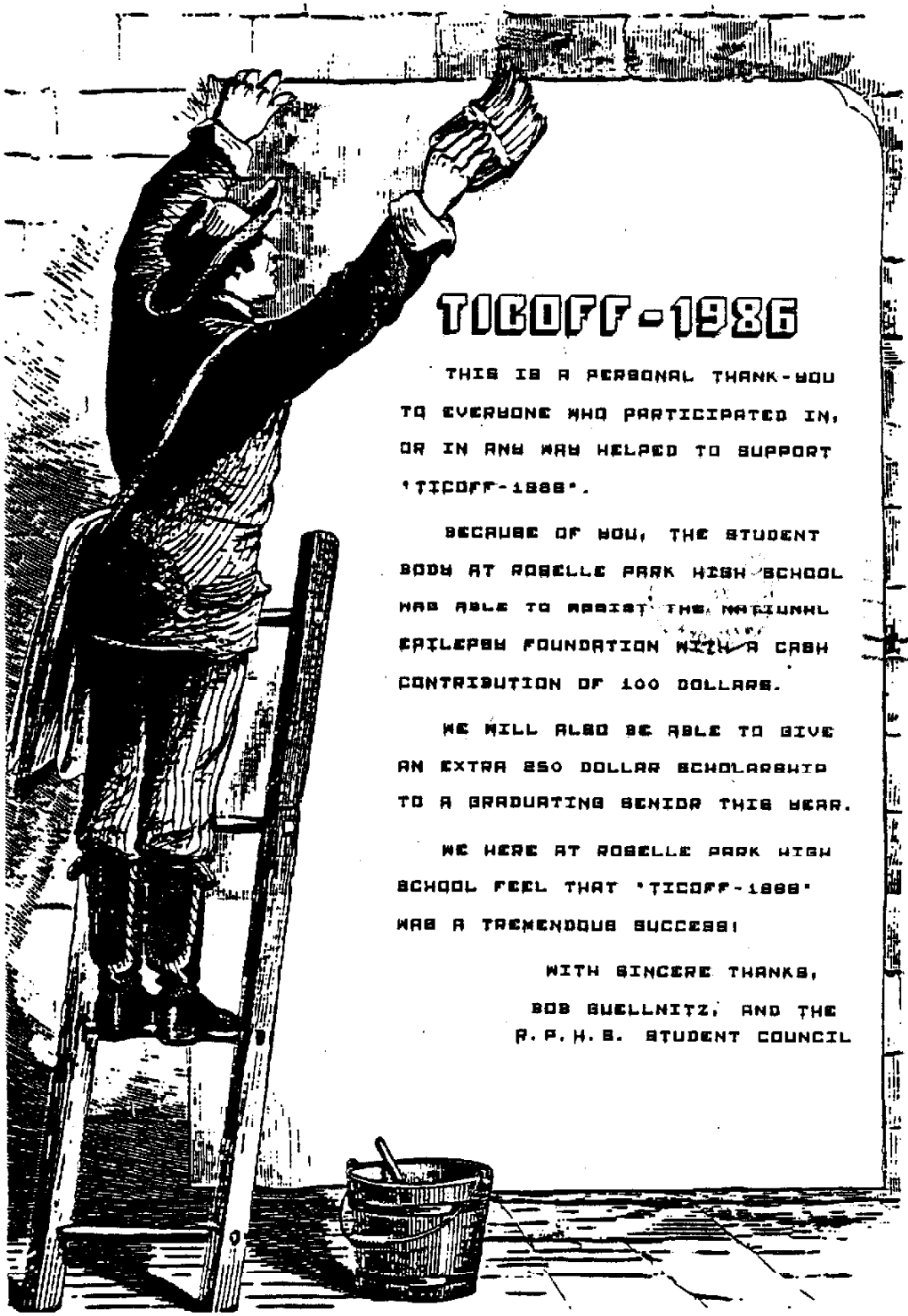
This is a helpful file provided by Clint. It is a PROGRAM file to be run out of E/A 5. Run this if you have run the c99 compiler on a source code file and received the dreaded "!!ERRORS!!" message. What it will do is prompt you for the compiled file's name (not the original source file!), read it in very quickly showing the file on the screen as it reads it. You can stop to read the file by holding down any key; releasing the key resumes the read. Then, after it has read the file, it will flash the lines again on the screen that contain the error message so you can (1) see where the error occured and (2) what the error message was. It is also nifty for reading ANY D/V 80 text file. It's purpose, though, was to help in debugging.

There are several other files that are, for the most part, files to be included in your c99 source codes as you use certain functions. We will go into this in some depth later, but you will use an "#include dsk1.filename" in your source files to copy these files into your source codes. For example, if you used some graphics commands in your source file to draw some sprites or such, you would need to use "#include dsk1.grf1refs" in your source code as a line before you started using the graphics commands. Else, the compiler won't understand what they mean and give you a multitude of errors. If you use commands to access disk files, you would have to use "#include dsk1.stdio" (for "standard input and output") before you started opening and reading from disk files. Notice the use of lower case in these #include statements. The compiler can use lower case, unlike the E/A Assembler which only accepts upper-case. Just keep the list of the other files as they will be used as you start to type in programs.

How does one enter programs with c99? You can do it two ways. You can use TI Writer, but always use "PF" to disk rather than "SF" and throw in the "C DSKx.filename" syntax to clean all the control characters out. Or, preferably, you can use the Editor of Editor/Assembler. We won't do a program this time, as you have enough to do for now.

What about recommended books? I strongly recommend "C PRIMER PLUS" by Waite, Prata and Martin (Sam's Publishing, 1984). It is 500 pages and costs about $22. It is the "Going Forth" (Brodie) for C. It is easy to read, starts at a beginner's level and is chock full of example programs. Some usable with out dialect of small c, some not (at least without some conversions). I went though two other books on C before I found this one. It is the best I have seen. If you know C, the bible (but much too advanced for me) is "THE C PROGRAMMING LANGUAGE" by Kernighan and Ritchie (Prentice-Hall, 1978). I found a back issue of Byte magazine also useful. The August, 1983 issue is devoted to C and contains some very nice articles and tutorials. You can still get a copy of this from Byte.

There you have it. The first chapter in the "Beginner's Guide to c99". If you are interested in more, let me know. I am just beginning myself and we can stumble through this together. My first swim in the waters was just great. Join in and learn c99.
you don't load this file after it and with it.



TICOFF-1986

THIS IS A PERSONAL THANK-YOU TO EVERYONE WHO PARTICIPATED IN, OR IN ANY WAY HELPED TO SUPPORT 'TICOFF-1986'.

BECAUSE OF YOU, THE STUDENT BODY AT ROSELLE PARK HIGH SCHOOL WAS ABLE TO ASSIST THE NATIONAL EPILEPSY FOUNDATION WITH A CASH CONTRIBUTION OF 100 DOLLARS.

WE WILL ALSO BE ABLE TO GIVE AN EXTRA 250 DOLLAR SCHOLARSHIP TO A GRADUATING SENIOR THIS YEAR.

WE HERE AT ROSELLE PARK HIGH SCHOOL FEEL THAT 'TICOFF-1986' WAS A TREMENDOUS SUCCESS!

WITH SINCERE THANKS,

BOB BUELLNITZ, AND THE
R. P. H. S. STUDENT COUNCIL

APRIL 1986

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
		1	2	3	4	5
6	7	8	9	10	11	12
13	14 GENERAL MEETING	15	16	17	18	19
20	21	22 STEERING COMMITTEE	23	24 ADVANCED PROG.	25 NEWSLETTER DEADLINE	26
27	28	29	30			

President:

Steve Citron
981 Townley Ave.
Union, NJ 07083

Send Dues To:

Marv Shuldman
28 Tyndall Rd.
Kendall Pk., NJ 08824

Write For Application:

Bill Dubrow
21 Seaward Ave.
Metuchen, NJ 08840

NOTICE!!

The Software Library Committee, consisting of Dave Green and Leon Green, is working arduously to reconstruct the club's software library. They are now in the process of collecting all of NEW JUG's software that has been loaned out to others, either officially or unofficially. They are asking that all such material be returned as soon as possible so that they may proceed. They are also asking that you contact either of them if you have programs that you are willing to have placed in the Software Library. And they are currently in the process of formulating policies related to the functioning of the Library so as insure its efficient operation. These recommendations will be presented at the next meeting of the Steering Committee. Now that we finally have TWO persons willing to run the Library, let's give them all of the help that we can! After all, what is a Users Group without a Software Library to benefit its members?! Dave Green may be reached at 463-9133 and Leon Green at 828-2435.

MULTIPLAN

by Tom Kennedy

ELECTRONIC SPREADSHEETS...CELLULAR REFERENCING...ABSOLUTE REFERENCING...CELLULAR ANALYSIS... WORKSHEETS... FORMULAS

These are buzzwords of a form of Data Processing that on the surface appears to be incomprehensible to all but accountants and the bridge crew of the Star-Ship Enterprise. As Word Processing is to writing a letter, Data Processing is to using a multiplication table.

Many people have a hard time using spreadsheets, because working with data in this format is similar to learning a new language. But once you learn to use the commands, and the procedure of working with data in a two-dimensional row/column format instead of a one-dimensional equation, you'll find many ways in which Multiplan will allow you to "crunch numbers" faster and easier than using a calculator and notebook. More than that, Multiplan is flexible enough to be used anytime you want to display, or use, numbers or words in a row/column format. In fact, you could even adapt Multiplan to use it as a Word Processor!

What is a spreadsheet? In business, you often hear reference to "our books". The "books" that the businessmen, and we, keep are a pen paper record showing the Debits and Credits of various bills paid and assets gained, plotted against a scale of time. Each intersection of row and column contains an entry for a value. The last column and/or last row contain a summation of all previous columns or rows. In an electronic spreadsheet, you recreate the printed form with the addition that each "cell" (a row/column intersection) can also contain a mathematical equation, or "formula", that automatically acts upon pre-defined cells and displays the result accordingly. If any value in any cell is changed, the formula instantly updates displayed results. This self-maintenance ability is what pays off in using an electronic spreadsheet, such as Multiplan.

To operate the Multiplan software on the TI, you must have 32K memory expansion, and at least one disk drive. An RS232 card and a printer are also handy, but not mandatory because unlike word processing, where the end result is a printed piece of paper, the end result with a spreadsheet is useful data, which may be used many ways. Most worksheets are well over 80 characters in width, (up to 2016!) and this requires a cut-and-paste job, so a wide-carriage printer is preferable.

To load Multiplan, you insert the cartridge and program disk, select Multiplan from the menu, and press <ENTER> to load. Before pressing <ENTER> you can select one of eight screen color combinations by pressing the space bar.

The first thing you will see is a grid across the top and left side of the screen. These numbers are the row and column locations in the top left, or "HOME" position. There are 255 possible rows and 63 possible columns, with the screen framing a small section. Each "CELL" is referred to by its row/column location, such as: R1C1, R10C22, etc. In R1C1, The Home position, there is a solid rectangle, as large as the width of the cell. This is your cursor, or "CELL POINTER", which is where any entry will appear. Below the cell grid is the COMMAND LINE, which shows the primary commands you will use. There are a number of sub-commands related to each of these, but you must type the first letter of the primary command first, or place the cursor over the command and hit <ENTER>.Below the command line is the MESSAGE LINE, which prompts you for further information when needed. In the bottom left corner is the current cell pointer location, and to the right of that is the contents of the current cell. Lastly, in the bottom right corner is the available memory space remaining.

There is a file in the TI Forum Data Library that assists in learning the commands. It is called MPKEYS.HLP and it is recomended that you get a copy before continuing.

Upon selecting a command, a command menu appears with a number of response fields shown. In each response field is a proposed response, which is the default selection unless you change it. To use a command, type it's key letter and fill in the response fields. To move through the fields, use the tab key until the cursor is

highlighting the correct area. Type in your response, and either tab to the next field or hit <ENTER> to activate the command.

When the necessary response is a row/column cell reference, there are two ways to respond: Absolute and Relative. Absolute referencing is a numerical definition of the cell location, such as R5C10 (the intersection of ROW 5 and COLUMN 10). A group of cells, a RANGE, is called by giving the boundary intersections separated by a colon(:), such as: R2C1:R4C10 defines a 3-by-10 cell grid consisting of columns 1 through 10 on rows 2 through 4.

Relative referencing involves identifying the desired cell by displacement from another cell, usually the one the cell pointer is currently on. As an example, if you are on row 5, column 10, (R5C10) and you wish to refer to a cell two rows up and three columns over, (R3C13) you could type in R-2C+3 or use the cursor keys to move the cursor over R3C13. The relative address will automatically update as you move. When the cursor is in place, hit <ENTER> (or tab to the next field) and the reference is defined.

So far, we have covered what you see on the screen and in response to the various commands; what the commands and key functions are; and how to fill in response fields where needed. Before going on to building a worksheet, you'll need to know how to save what you're working on, and how to load it back in. Besides the fact that you'll want to take a break occasionally, it's nice to be able to experiment with the commands, "messing up" the worksheet, and then loading the "clean" version back in to continue.

The LOAD and SAVE commands are under the command TRANSFER. Hit "T" and the menu will be displayed. Since the first option is LOAD, hitting <ENTER> now will prompt for a filename. To select SAVE, (or another option) hit the first letter and <ENTER>, or tab through to the desired item and hit <ENTER>. When loading or saving, you'll be prompted for a filename the first time, which will become the default response.

Multiplan also incorporates an extensive helpfile contained on disk. When the command line is displayed, you select HELP with either the Help action key (<AID> or "?") or by typing "H". The worksheet will be replaced by the beginning of the helpfile. If a command has been selected, hitting the help key will display the specific section of the helpfile that pertains to the command you are using. The help menu allows you to RESUME (return to command menu), START at the top of the helpfile, or move to NEXT or PREVIOUS page of information. The first step to creating a worksheet is to decide how many rows and columns you'll need, and how the data will be displayed. It is best to sketch this out on paper to get a feel for how it will look. Also, you'll need to decide what formulas will have to be created that use the data contained in the worksheet. Lastly, you will probably want to change the format of many of the cells, usually by rows or columns. Most often, the formatting required is for display purposes. Cell width, alignment of the data within the cells, etc.

Now that you know how everything will look, begin by formatting the cells. Upon start-up, the cells are set with a number of defaults. You may want to change the widths of some columns, to between 3 and 32 columns, to show all of the entry for the cells. If the data in a cell is too large to fit the width of the cell, it will be truncated to fit, unless it is a numerical entry, where it will be replaced by a sting of "#"'s.

FORMAT CELLS is used to set cell alignment and display format. A cell can be aligned to either center text for columnar headers, etc., or to align data displayed in tables. For instance, a table of dollar values could be shown with a "$" in front and decimal points aligned. The display formats are used to show how the data appears in a cell. CONTinuous allows the text in a cell to run over the right boundary to the next cell. If all cells are made continuous, you have a word processor-type format. EXP displays numbers in scientific notation. Fixed Point rounds off decimals to a defined number. GENeral is as you see when starting up, values displayed as entered. INTeger rounds off all numbers to integers. "$" (Dollar) adds a dollar sign to numbers and rounds to two decimals. "*" Replaces the number with an equivalent number of asterisks, to use like a bar graph. "%" displays the number in percent form. Lastly, the "-" just leaves the setting at the previous option.

Now that the cell formats are defined, it's time to start entering data. Begin by labeling your rows and columns, as necessary. To enter data, either text or values, move the cursor to the desired cell and hit either "A" or "V", depending on the type. The command line will disappear and you'll be prompted for either text or value. Type in your entry and hit enter either <ENTER> to return to the command line, or use the appropriate FCTN-ARROW key to move to the next cell. With the FCTN key, when you land on the next cell, you are prompted only for text/value entry. In this case, you do not hit A or V to declare type, but when you begin entering data, Multiplan decides what style the data is, and responds accordingly. The only disadvantage is that there's a slight delay between the first character of your entry and the remainder, so if you type in, for instance, the word "TOTALS" too quickly, all you'll see in the cell is "TTOLS". After a bit of use, a "stutter" habit is developed in how you enter data, so this becomes less apparent. When entering data, if an error is made, do not use the FCTN-S key to backspace for correction (as programmers are used to), the backspace key is CTRL-H (as telecommunication folks are used to).

If, after creating part of a worksheet, you need to

add or delete rows or columns, three commands apply. DELETE completely removes any number of rows or columns. BLANK just removes the data in the cells, the row/columns remain and retain their formats. INSERT creates a new row or column set to the default settings.

Formulas are used to perform a mathematical computation upon the data in a cell or group of cells. One example is in a sales order form, where you have a column of data that is totaled at the bottom, multiplied by a tax percentage, and the tax added to the result. The cell in which the sub-total is to appear would contain a formula describing a sum of the data in the columns, expressed as either a chain addition problem, (R3C5+R4C5+...+R10C5) or using the SUM() function and a range of cells. (SUM(R3C5:R10C5)). The formulas can become quite complex, depending on the work performed. Appendix C contains a list of the mathematical functions that can be used in building formulas. Formulas can also consist of names of cells as the opperand, as in "SUBTOTAL x .079", to calculate the entry for a cell named TAX. Names are assigned with the name command. Names can be any continuous string of alpha- numeric characters, but must begin with a letter. Simply place the cursor over the cell to name and press N. Type in the desired name to the response field, and TAB to the next field. The current cell will be shown as the proposed response. If a range of cells is desired, hit the FCTN key, at the cell response, to move the cursor from the current location to the end point, then hit <ENTER>. In this manner, a whole row or column can be named. Names can also be used in the GOTO command to aid in moving quickly to a location. "GOTO TOTALS" for example.

Windows allow you to view more than one area of your worksheet at one time. You can split a row or column of titles to form a window over the data, so as the cursor is moved throughout the worksheet, the headers remain in place to see what data is shown. Also, separate worksheets can be developed in one and divided into windows so all can be seen ar once. After selecting the window command, four options are shown. SPLIT is what opens the windows, either horizontally, vertically, or at preset titles. LINKing two or more windows scrolls them together as you move through the worksheet. BORDER is used to put a border of any character surrounding the windows, to make them easier to read. A window is cancelled with the CLOSE option.

Once you have finally created the worksheet, and all the data has been entered, what do you do with it? In a sense, the end product is the worksheet, because you may refer to it constantly as new data is applied, and a printed copy might become outdated quickly. After all, that's part of the reason you are working on an Electronic Spreadsheet in the first place, the instant and easy update of information.

In some cases though, a printout is desired, either in the form of a disk file that can be incorporated into a document on a word processor, or a hard-copy printout for reference. The printer command has four options used in printing the worksheet. FILE prints the worksheet to disk in display variable 80 format, which can be loaded into a word processor. Before printing a hard copy, you must first set margins and print options. The MARGINS option sets the limits of rows and columns in the printout, along with indentations and paginations. OPTIONS defines the portion of the worksheet to be printed, using a range of cells. The set-up field contains te device name of your printer. The last two fields let you print the formulas "hidden" in cells, and whether or not to print the row/column numbers. After margins and options are defined, select the PRINTER option to begin the print-out. If the width of the worksheet exceeds the width of your printer carriage, the left half will be printed entirely, then the right half below that, so the two can be cut--pasted together.

In some cases, you may be working on a number of worksheets that are related to each other, such as in a business with SALES/PAYROLL/INVENTORY spreadsheets. These separate files can be linked together so data can be drawn from, as an example, the INVENTORY file to be used in the SALES worksheet and information from SALES could be used in PAYROLL.

The eXTERNAL command, (press "X" at command line) is used to COPY data from an inactive sheet into the active one. You are prompted for the filename of the source sheet, the name (or R/C reference) of the source cell, the destination cell of the data, and the LINK option. If LINK is selected, then the two sheets will become linked so that when the destination sheet is loaded, the source sheet will automatically be used to supply data where needed. The LIST option displays the names of all sheets supporting the active sheet. The USE option allows you to switch which inactve sheets will support the active sheet, so long as they are in the same format. As an example, the SALES sheet would call upon different INVENTORY sheets for each month, all created in the same format, with different data.

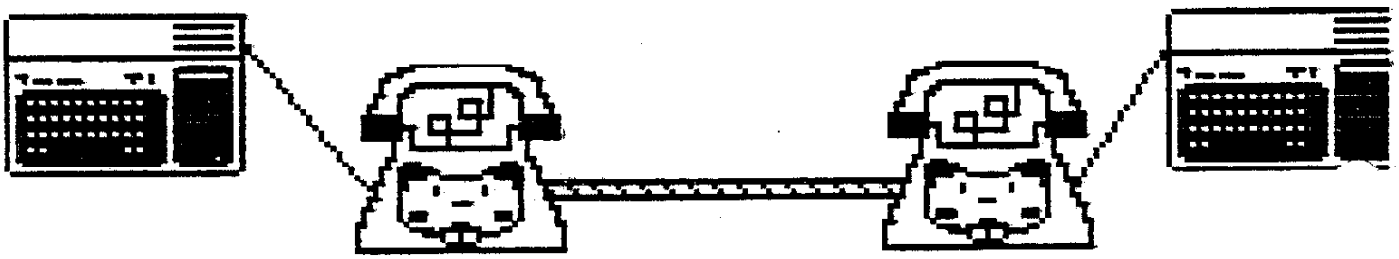
Multiplan is one of the most powerful tools to be used on any computer. It's versatility allows it to be used in many different applications. Word Processing, record keeping, budget/accounting, etc. Any application that requires storing data in a tabular format. The instant update of information and the advanced mathematics capability can be used in a variety of ways.

Versatility is the main attraction of the many spreadsheet programs used on various machines, and in fact, Multiplan can even use files stored in VISICALC(tm) format. VISICALC, one of the "first" major spreadsheets, is similar to Multiplan in many ways: the screen display; cursor positioning; error correction; and entering data and formulas. The referencing of cells is more detailed with Multiplan, including the ability to name cells for

ease of use. It has been shown that Mulitplan can be easier to pick up and use for the person not familiar to speadsheets, although once the concepts are mastered, the usage is similar in all. With a familiar knowledge of a program like Multiplan, you could do away with a word processor, a database manager, or even a pocket calculator, although each has it's specific advantages.

We have tried to cover the basics of getting started in working with spreadsheets, but this has still only scratched the surface of the wealth of information within the manual supplied with Multiplan. A walk-thru in the first half provides a very good introduction, and the second half documents each command and function in detail. There are also a number of good books available on Multiplan, and the software is the same on nearly every machine.

(d/l from CompuServe by mel gary)



LOCAL BBS

BILL REISS 679-0549 24 HRS

BILL WRIGHT 257-2607 24 HRS

BEAVER BRD. 238-8170 9PM-9A

TECHIE 300-1200

C TUTURIAL-2

By Warren Agee 70277,2063

Second in a series of tutorials on the C language

In my first tutorial, I covered storage classes, something necessary to know before you even start programming in C. Functions are another basic concept which must be grapsed before writing C programs. Simply put, a function is a subroutine designed to perform a specified task. In some cases, values are passed to and from functions, while other functions require no communication. Numerous functions are part of the standard C library, like gets() and puts(), which allows input and output of strings, respecitively. Others, like fopen(), are kept in function libraries and stored on disk. And, of course, you may write your own functions. Indeed, the process of writing a C program involves writing user-defined functions, then putting all these functions together into a runnable program.

So, where do we begin? First of all, naming conventions. Although a function may have a name of any length, the c99 compiler only recognizes the first six characters, and they may be only alphabetic. Unlike most other compilers, the underscore (_) is not allowed. Secondly, what distinguishes a function name from a variable is the presence of parentheses. Depending on the purpose of the function, the parentheses may be empty, like getchar(). If the function requires values to be passed to it, these are placed inside the parentheses, as in puts("\nHello there!") . Now that the cosmetics are out of the way, let's get down to creating a function.

As I mentioned in the last tutorial, to call a function, merely type in its name, followed by a semicolon. To alert the compiler that you are creating a function, omit the semicolon.

```
clr()
{
  int c;
  putchar(12);
}
```

Here we define function called clr(). Note the missing semicolon. Also note that since the parentheses are empty, we are not going to communicate any values to the function. Next we have an opening brace, which signals the beginning of the function body. Note that the brace aligns with the first letter in the function name above; this is a standard C convention to make programs easier to read. Then we indent a few spaces, another convention. We then define the integer variable "c." Because this statement occurs inside the function body, it is local to that function (See Tutorial #1 for more info). The next statement is a standard console i/o function which prints a character to the screen whose ASCII value is in parentheses. In this case, putchar(12) simply clears the screen. We then find a closing brace which ends the function. Notice that the two braces line up.

Suppose that we want to pass one or more values to a function. Look at this:

```
add(n1,n2)
int n1,n2;
{
  int sum;
  sum=n1+n2;
  return(sum);
}
```

The first line tells the compiler to expect 2 values in the parenthesis when this function is called. We give these two values the names n1 and n2. When one calls this function, two numbers may appear in parentheses [like add(1,2)] or two variables [like add(a,b)]. The next line is a variable declaration, which was described in the first tutorial, but the purpose here is a little different. The function add() receives two values; now the compiler has to know what KIND (class) of values they are. Since we are passing numbers, we declare them as integers. also notice that this must come *before* the opening brace. We then declare another variable, sum, to hold the sum of the two integers. We perform the addition just as one would do in BASIC. The next line is very important.

When this function is called, we give it two numbers, and we want back the sum, right? Since the variable "sum" is local to add(), once we return to the calling program, the value of sum is lost. "Sum" only exists in add() and nowhere else. What we have to do is artificially send the value of "sum" back to the calling program, and we do this with the return statement, as shown above. Now, when we call add(), we will get back the value of sum, like this:

```
main()
{
  int c;
  c=add(5,2);
}
```

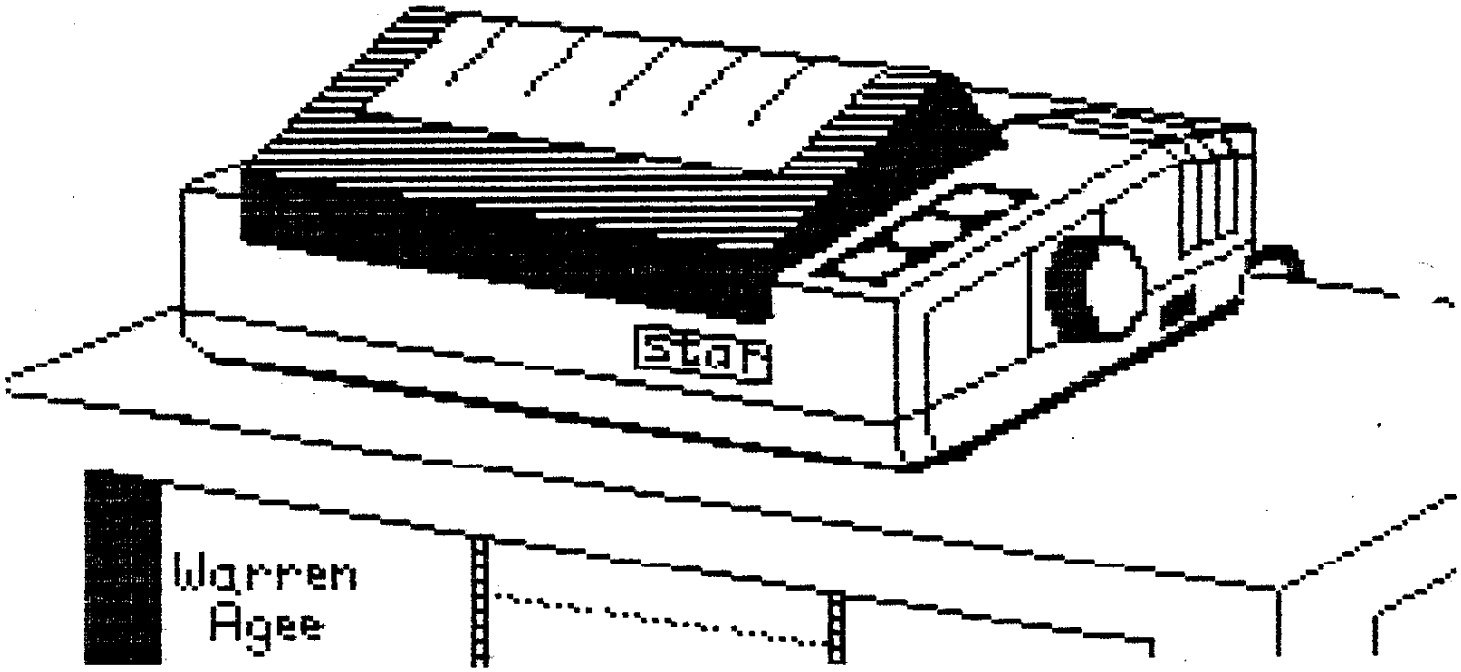
The expression "add(5,2)" is replaced by the answer, and we assign that value to c. If we just wrote "add(5,2)" and did not assign it to anything, the sum would just be discarded.

But why do all this? We could just declare "sum" as an external variable in main(). That way "sum" would retain it's value throughout the entire program. In very large programs, you can run into difficulties if you use

only external variables. Stick to local (automatic) variables whenever possible.

Well, there you have it! There is a lot more to cover as far as functions go. The return statement only returns ONE value, no more. If you need more than one value back from the function, you have to use pointers. Pointers can be quite sticky and confusing to beginners, so I will be spending quite some time on them in the next few tutorials. So stay tuned, and experiment! It's the only way to learn! (Well, reading my tutorials may help a bit!)

(d/l from CompuServe by mel gary)



FOR SALE

EPSON MX-80 PRINTER-PARALLEL

GOOD CONDITION

[illegible]00 DOLLARS

CONTACT MEL GARY AT 828-5407

SOLD

ANNOUNCING THE TI-99/4A TRAVelER!

An exciting new magazine-on-disk, which will include over 700 sectors of programs and articles in each issue, featuring authors like Mack McCormick, Jonathan Zittrain, Tom Weithofer, Tom Kennedy, Todd Kaplan, and Barry Traver.

The first issue (September 1985) -- which has already appeared -- includes an article on TI ARTIST and GRAPHX by Ron Albright, assembly language articles by Mack McCormick on accessing the RS232, an article "TI&ME: Saving Tips on CompuServe" by CIS Wizop Jonathan Zittrain, two utilities by Tom Freeman (one to print DV80 files _sideways_ on your dot matrix printer, and the other to print two-column newsletter-style sheets with full justification), a versatile assembly language utility accessible from Extended BASIC called "RAW" (for "Read And Write") for single-sector disk access by Barry Traver, another utility by Barry to show DV80 files on your screen in 40-column text mode, a colorful game called HoleyMoley by John Behnke, plus much, much more.

The "diskazine" is actually priced less than some "freeware": you get a six-issue subscription for only $30 (that's over 4000 sectors, so that you are only paying less than 3/4 of a penny per sector!). This special price is only guaranteed through the end of 1985, after which time subscription rates may increase, so _now_ is the time to send in your check!

The TI-99/4A TRAVelER is edited by Barry Traver, whose programs have appeared in various publications ("Giant and Dwarfs" in 99'er Home Computer Magazine, "Merge/Read" in Craig Miller's Smart Programmer, and "Numb/Conv" in Super 99 Monthly, where he is now a regular contributing staff writer). He was recently maed a full sysop on the TI FORUM on CompuServe, where he earlier received the honor of being asked to serve as Chairman of the Expert Member Board.

The "diskazine" is being "published" on commercially-made SS/SD "flippies," so that the same format will work on everyone's disk system. Because of the medium employed (disk rather than cassette), it will be assumed that most subscribers will also have at least a 32K RAM card, Extended bASIC, and probably Editor/Assembler as well, and the contents of TRAVelER will be chosen accordingly. (In other words, you can expect something a bit more sophisticated than the TI BASIC programs which are generally all that is available in magazines and books in bookstores!)

TRAVelER SUBSCRIPTION FORM

Name_____

Address_____

City_____ State_____ Zip Code_____

Please send your check to the following address: Barry Traver, Editor, GENIAL TRAVelER, 835 Green Valley Drive, Philadelphia, PA 19128. Thanks!