TEXAS INSTRUMENTS

TI Forum

by Ron Albright & Jonathan Zittrain

In the July column we asked for letters from those interested in hearing more about 99 and since we did, we'll cover more in this column. We will go into C99 programming in more depth and will compose a short program which will demonstrate console input and output. The program, again, will not "do" anything, but it will demonstrate how to display information on the screen and accept user input from the keyboard. It will also demonstrate one of the most powerful features of C99, that is, a direct assembly language interface which will allow the inclusion of assembly language code into the C99 source code for direct assembly by the Editor/Assembler package. We will take a source code listing and examine it line-by-line, explaining each function as we go along.

If you will, take a look at Listing 1 for an example of a C99 program. Let's make a couple of observations. First, you will notice that the C99 program is typed in using lower case. It is standard C (and C99) syntax to type everything in a C program in lower case, except for constants in #define statements (which we do not use in this example. The one instance where we use upper-case, is in the direct assembly language interface statement "REF PRINTF". This is because everything between "#asm" and "#endasm" is in assembler syntax and not C99 syntax. It is ignored by the compiler and acted upon only by the assembler. The printf function (a powerful C routine) is in an external file on your system disk (it is a D/FBO already-assembled file) and is not acted upon by the compiler but must be loaded (like CSUP) before running this program. There are two ways to write libraries in C99. The first being to write it as source code in C syntax, which would be used in your program as an "#include DSKx.filename" statement. This type of library function has to be compiled and assembled fresh each time. These files are in D/V80 format and an example would be the "grflrf" library which holds the graphics commands in C99. We will use this type in a future program. The second way is to write it in C99, compile it, and assemble it to D/F80. This type of file, exemplified by the "printf" library function, is utilized with a "#asm",REF DSKx.file name, and "#endasm" sequence, as in our program. This second type of library is "ready-to-use" by the program and requires very little space in your compiled code. It is ignored by the compiler and, for the most part, by the assembler and will speed the process for program generation. But remember, when you use this external library, it MUST be loaded in after CSUP and from the option 3 E/A prompt, before running your program.

Now, let's look at the program. Lines 1 through 3 are our syntax to tell the compiler to ignore the lines between "#asm" and "#endasm", and the signal to the assembler that we will be using an external file "PRINTF" for our program. The next line is the required "main()"; function. As we noted in July, a "main()" function is required in every C program and is, traditionally, the first function call. The next line is the open brace "{" which marks the beginning of the function main. The next line is "putchar(12)". The putchar() function is an example of a function resident to C that is predefined in the compiler. It is no different from a user-defined function (like "main()") except that it is already known to the compiler. It is also an example of a function that needs to have a variable in its parenthesis, in our case, "12". What does this mean? Putchar() simply means

TI-RLE Graphics From Epson FX-80

to "put a character to the screen" (our "input/output" device, or "I/O"). We are printing ASC 12 (CTRL L) to the screen. What does it do? It simply clears the screen. It would be akin to "a CALL CLEAR" in Basic syntax. We could have printed any character to the screen with this function, by passing the desired ASCII number to putchar. If we had wanted to print an asterisk on the screen,

continued on page 82

Texas Instruments Introduces New OmniLaser Family of Page Printers

TI OmniLaser Printer Family

Texas Instruments Incorporated recently announced the OmniLaser series 2000 family of page printers, setting a new measure for performance and reliability in laser printers. With faster printing speed, a much higher duty cycle and a machine life that is up to 15 times greater than first generation laser printers, TI's OminLaser printers represent a second generation of page printers designed for shared resource environments.

"By addressing the needs of the systems market, the OmniLaser printers can reduce the cost of ownership of page printers, since a printer can be shared by many users," said Tom Stringfellow, vice president and manager of the Peripheral Products Division in TI's Data Systems Group. "Because of significant advances in printing speed, durability and machine life, the OmniLaser printers are truly practical in the systems environment."

The OmniLaser Series 2000 family consists of three printers: the OmniLaser 2015, OmniLaser 2108, and OmniLaser 2115. The OmniLaser 2015 is intended for high-quality, high-volume word processing with limited graphics, while the OmniLaser 2108 and 2115 support the Adobe Systems Incorporated PostScript page description language, an emerging standard that allows users to print pages that integrate text, graphics and scanned images. All combine second-generation print engines with proprietary TI controllers to create high-performance page printers.

"TI's semiconductor technology capability will make possible significant price/performance advances in the OmniLaser Series 2000 family," Stringfellow said. "As TI perfects new technologies, we plan to incorporate them into the OmniLaser family to continue to bring the greatest possible value to users."

The OmniLaser 2015

With a print speed of 15 pages per minute (ppm), the OmniLaser 2015's print engine is nearly twice as fast as most first-generation laser printers, which typically prints eight ppm. The 2015 has a maximum duty cycle of 25,000 pages per month, compared to an average of 3,000 for first-generation printers; the 2015's durability ideally suits it for rigorous use with even a large system. And with a machine life of 1.5 million prints—up to 15 times greater than that of most current laser printers and the cost per page, much lower than its first generation counterparts.

The 2015 has the capacity to handle 500 sheets of paper through two 250-sheet input bins, allowing users the convenience of loading letter and legal size papers without fre-

continued on page 82

Texas Instruments Adopts PostScript For Laser Printer Family

In support of its introduction recently of the OmniLaser family of printers, Texas Instruments also announced an agreement with Adobe Systems Incorporated to adopt is PostScript page description language. PostScript will reside in two of TI's new OmniLaser Printers, the models 2108 and 2115.

PostScript allows the flexibility to fully integrate text and graphics, thereby allowing complete control over how the printed page will look. It enables the printing of rotated type, textures, patterns, halftones, images and incorporates a variety of graphic arts-quality typefaces that can be printed in any size and anywhere on the page.

"PostScript is a dynamic tool that will enable TI's OmniLaser printers to meet the needs of our target applications, including desk-top publishing, computer-aided engineering, forms generation and elaborate business presentations. These jobs require the full integration of text, graphics and scanned images," said Tom Stringfellow, vice president and manager of Peripheral Products Division in TI's Data Systems Group.

PostScript's state of the art capabilities have been adopted by many major manufacturers, including Digital Equipment Corporation, Apple Computer and Wang Laboratories Incorporated. PostScript is widely recognized in the industry as a leading page description language for printers.

The OmniLaser family of page printers includes the models 2015, 2108 and 2115. The OmniLaser 2015 can print 15 pages-per-minute with a basic controller for limited text and graphics. The models 2108 and 2115 use the PostScript controller. The model 2108 can print eight pages-per-minute, and has a duty cycle of 10,000 prints-per-month. Its life expectancy is approximately 600,000 prints. The model 2115 can print 15 pages-per-minute for an approximate 25,000 prints-per-month. Life expectancy for the models 2015 and 2115 is about 1.5 million prints, which is up to 15 times the expected output of first-generation printer engines.

For additional information contact Texas Instruments P.O. Box 809063, H-86 Dallas, TX 75380-9063; (80 527-3500.

TI Forum continued from page 81

we could have used "putchar(42)" (ASC 12 is CTRL L—a clear screen code). You will notice also that the putchar function is ended with a semicolon. Every function call in C is ended with a semicolon. Remember that and check your program lines for a semicolon after each line as it is, in my experience, the biggest single source of compiling errors when you are starting out.

The next line is a function call, I think, unique to C99. It is the "locate()" function. All this does is place the cursor on the screen at the row and column locations you pass to the variable, locate(). Think of it like the "DISPLAY AT (row, column)" statement in XB. Unlike the DISPLAY AT statement were you designate in one statement the row/column location AND the string, in the case of locate() you only designate the row/column. The next statement will designate what you want to print. So we pass row 9, column 10 to the locate() function and end it with a semicolon. Next, we designate what we want "displayed" in our two-part "DISPLAY AT" statement, using another function, "puts()." Puts simple means "put string." Just like putchar meant "print this char", puts simply means "print this string". We also pass to this function a variable in the form of a statement enclosed in parenthesis. So, we have told the compiler, simply, in two statements, nothing more than "DISPLAY AT(8,10): "= = = = = Menu 1.0 = = = = =". The next 8 statements are 4 pairs of incrementing locate and puts statements to complete our initial screen. The next line is our first keyboard I/O function. And now here is your first test: If "puts" meant "put string" and "putchar" meant "put character." What does "getchar" mean?

Answer: "get a character"

Hope you got that right. All getchar does is fetch a character from a keyboard press and passes it to the program. You will notice that, unlike "puts" and "putchar" where the program passes the variable to the function by including it in the parenthesis of the function call, getchar() has no variable passed in its parenthesis. That is because we will get the variable for getchar from the keyboard press not the program. Getchar is nothing more than an "INPUT X" statement in BASIC. Just like in BASIC, the program waits till a key is pressed to go on. Right now, we don't care what key is pressed; only that a key IS pressed. We will look closer at getchar() later in this tutorial. The final function call in the main() function is to call a user-defined function in the program. The next block of code is called "menu()" and, by including the "menu();" statement in main() we thereby pass program control to the next function. Think of it as similar to a "GOSUB" in BASIC. Since we don't have line numbers in C, we call routines by name. (Actually, this is closer to "CALL subroutine" in XB, but

continued on page 214

Listing 1

```
#asm
  REF PRINTF
#endasm
main()
{
        putchar(12);
        locate(8,10);
        puts("=====Menu 1.0======");
        locate(9,10);
        puts("==A menu-Utility==");
        locate(10,10);
        puts("==================");
        locate(20,13);
        puts("Written in c99");
        locate(21,10);
        puts("(c) 1986 Ron Albright");
        locate(24,13);
        puts("Hit any key...");
        getchar();
        menu();
}

menu()
{
        int c;
        putchar(12);
        locate(1,10);
        puts("==Menu-Utility==");
        locate(5,1);
        puts("[1] Choice 1");
        locate(6,1);
        puts("[2] Choice 2");
        locate(7,1);
        puts("[3] Choice 3");
        locate(8,1);
        puts("[4] Choice 4");
        locate(24,1);
        puts("Your Selection?  ");
        c=getchar();
        if(c<49 | c>52)
            menu();

        putchar(12);
        locate(1,1);
        c=c-48;
        printf("Your choice was %d from the menu ",c);
        locate(24,1);
        puts("Press any key to continue...");
        getchar();
        putchar(12);
        main();
}
```

New OmniLaser continued from page 81

quently restocking paper trays. Resolution is a high 300 x 300 dots per inch, for crisp graphics and word processing that approaches typeset quality. The 2015 has two slots for plug-in font modules. Standard typeface is Courier 10, 12 and 16.7 (compressed) pitch; additional fonts can be downloaded into the standard 512K system RAM.

The 2015 is targeted for environments such as legal and insurance offices, or other business applications that primarily require word processing, with additional requirements for data processing, forms generation and business presentation graphics.

The OmniLaser 2115 and 2108

The OmniLaser 2115 uses the same 15-ppm print engine as the 2015; its powerful PostScript controller is built around the 32-bit Motorola 68000 microprocessor, the same class of microprocessor that drives TI's new Business System 1000 series of supermicrocomputers. With three megabytes of RAM, the OmniLaser 2115 has more computing power than AT-class products, providing enhanced flexibility in layout and design of documents. The 2115 has two font cartridge slots, and optional fonts can be addressed via the font cartridges.

The speed and durability of the OmniLaser 2115 print engine, combined with the power and flexibility of PostScript, makes it an ideal laser printer for shared resource use in such applications as in-house publishing, CAD/CAE workstations and forms generation.

While the 2115 brings high-functionality to the shared resource market, the 2108 serves as a workstation printer for the same graphics-intensive applications. The 2108 has an eight ppm engine with a 10,000 page per month duty cycle, a 600,000 print machir life and a 68000-based controller with two megabytes of RAM.

Pricing and Availability

The OmniLaser 2015 will be available in July from TI resellers at a suggested list price of $5995. The OmniLaser 2115 and 2108, will be priced at $7995 and $5995, respectively.

The OmniLaser Series 2000 printers have industry-standard emulations and interfaces. OmniLaser 2015 standard printer emulations include the Diablo 630, Hewlett-Packard LaserJet, Texas Instruments OMNI 800 Model 855/Qume Sprint 11, and optionally via emulation cartridges, the TI Model 810, IBM Proprinter and HP LaserJet 500 Plus.

RS-232 serial and Centronics-type parallel interfaces are standard. The OmniLaser 2115 and 2108 have Diablo 630, HP LaserJet 500 Plus, TI 855 and Hewlett-Packard 7475 plotter emulations, and interfaces for RS-232, Centronics-type parallel, RS 422 and AppleTalk.

Service will be provided by TI's nationwide network of trained technicians. Service programs include on-site maintenance, key operator training and a customer support line, among others.

Texas Instruments Incorporated pioneered the technology behind its patented semiconductor thermal printing process and the world's first thermal printing, silent data terminal. The company's Data Systems Group also introduced the OMNI 800 Model 810, which has earned an industry-wide reputation for reliability in dot-matrix printers.

For additional information contact Texas Instruments, P.O. Box 809063, H-860, Dallas, TX 75080-9063; (800) 527-3500.

Mention that you read about it in *Computer Shopper*. ●

.I Forum
continued from page 82

let's not confuse ourselves at this point.) We are "GOSUBing" to the function "menu()". We close our main() function with a closed brace and we are finished with our first function in this short program.

We next start a new function, called "menu()". We do not end the naming of a new function with a semicolon. We then use another open brace to signify to the compiler the start of this function's code. The next line is an important one. It is the declaration of a variable, "C". C, unlike BASIC, requires that you declare ALL variables within a function at the START of the function. (Again, for the sake of simplicity, I will not discuss the difference between local and global variables). WE are declaring an "integer" ("int") variable, called "C." We are telling the program, "we will refer to something called "C" in this function and it will be an integer." Next, we clear the screen again with putchar(12), and set up another screen display with a series of 6 pairs of "locate" and "puts" functions. Again, a series of "DISPLAY AT's".

The next statement sets out previously declared variable "C" equal to another "getchar". Unlike our first getchar(), this time, we care what key is pressed as it determines which menu was selected. With this statement, we equate "C" to the ASC value of the keyboard item pressed. We are, in effect, saying TWO things: `INPUT X  C=ASC(X)`

The next statement is simply to test if the keyboard press for getchar() was something between 1 (ASC 49) and 4 (ASC 52). If it was less than ASC 49 or greater than ASC 52, we call menu() again. In essence, we are saying, in BASIC,

```
100 INPUT X
110 IF (X<1) OR IF (X>4) THEN 100 ELSE 120
120 (the next statement)
```

The "!" (not a colon but FCTN A) is a "bitwise inclusive OR". If either or both the operands are TRUE, then the "THEN" operand is executed. "IF the ASC value of C is less than 49 OR the ASC value of C is greater than 52, THEN ask again; ELSE do the next statement." If either of the statements are true, do "menu()" again. If both are false (i.e. C is between 49 and 52), keep going.

The next two statements clear our screen again with putchar(12), and place our cursor at row 1, column 1 with "locate(1,1)". Then, we take our integer variable C and set it equal to C-48. What does that do? If C had the ASC value of 49 ("1" was pressed), C-48 would give us C = 1. If C equaled 52 (4 was pressed), C-48 would be 4. This statement is nothing more than saying, in TI BASIC, `C=CHAR$C`

Then we use a new function, "printf". Like puts(), printf() allows us to print a string. But it is different in that it also allows us to print a decimal integer variable also as part of the string. Look at the printf statement. The "%d" in the statement tells the compiler that we are going to print a decimal integer variable (thus, the reason for the "d"; we would have used "%C" to print a character variable). The entire statement is enclosed in quotes, then a comma, then the integer variable we want placed in the screen display where the "%d" is. In this case, the integer variable "C". Since we didn't know before hand, which key would be pressed for the menu selection, we had to use a variable. And to print a variable, we have to use printf() instead of puts(). If we were clairvoyant and knew that menu selection 1 would be chose, we could have used puts ("Your choice was 1 from the menu"); but we didn't so we use the variable C and printf(). Next, we move the cursor to row 24, column 1, print "Press any key to continue.." and then wait for a keypress with getchar(). Again, this time we don't care which key is pressed for getchar(). Then, we call our first function "main()" and set things up all over again. We close the function "menu()" with a closed brace to signify the end of the function "menu()".

We then save this to disk. Run the compiler for an output file and assemble the output. When we load our program, we have a slight twist. First we load our assembler output with E/A option 3, then we load CSUP as always. But, instead of entering nothing when we get our "File Name:" prompt for the third time, we have to load a third file. When you get the filename prompt again after loading CSUP, load in the D/F80 file "PRINTF" from your C99 program disk. Since we REFerenced it in our program, we HAVE to load it before the program will run. When PRINTF is loaded, hit enter the fourth time you get the "FILE NAME:" prompt from E/A 3, then enter "START" for the "Program Name;" prompt and our menu example should run.

There you have it. Your first C99 "program" (such as it is). If you want more on C 99, write us at *Computer Shopper*. As always, we will include some programming tips with your reply if you enclose an SASE. And we will reply to every letter with an SASE.

Quick Product Note: I have just received a copy of Joy Paint 99 (Great Lakes Software, 804 E. Grand River Ave., Howell MI 48843; $49.95). While I plan a full review soon I have to tell you now that I think it is the very best graphics program for the TI I have ever seen. Absolutely. Buy it. More on that later. Corcomp (2211-G Winston Road, Anaheim, CA 92806; (714) 956-4450) continues its support of the TI with two new product announcements. They have a "99Home Sentry" system which is the software to run the popular X-10 Powerhouse to computerize your home. Lights, cooling/heating, and appliances can be programmed to run off the X-10. The software and hardware has a SRP of $79.95 and you'll need a separate module (running $13.95) for each electrical appliance you want to control. This system is available for many other computers and now we are in that mainstream as well. The second product from Corcomp is a memory expansion device. You can get a stand-along model (not requiring Expansion Box) in a 256K version ($249.95) or 512K ($269.96) or a card for the expansion box (256K for $169.95; 512K for $229.95). They also hint at some new software to utilize these devices to their maximum potential. Write or call Corcomp for details. While it is difficult to print in this column schematics for hardware modifications, we hope to make some available by mail soon. If you have made any changes that you would like to share with others, or if you are interested in doing soldering to improve your system, send your schematics or needs to us at the *Computer Shopper*. We know there are schematics available for adding 32K memory directly to the console, building an auto-answer/auto-dial device to a cheap Volksmodem, building an 8K "super cartridge," and others and we hope to be able to mail those out soon. You have to write to let us know your needs. Speaking of schematics, one of the best sources of that kind of information comes in the form of a newsletter from Bruce Caron out of Canada. It is full of technical information of the highest quality. The *R/D Computing Newsletter* is monthly and excellent. It is not new having been in publication for over a year and the back issues are worth getting. You can subscribe for $14 year and that includes 12 issues and first class postage. Write to Ryte Data, 210 Mountain Street, Haliburton, Ontario, Canada KOM 1S0. Thanks to the Mid-South Users Group (P.O. Box 38522, Germantown, Tennessee 38183-0522) for the great newsletter. What an active group. We'd like to see more users group newsletters as they are a major source of developments for the 99/4A home computer. We're also interested in giving a little ink to any Fairware programs users write, so send us the details and we'll try to publish them.

So much new software has been written for the TI-99/4A recently, all of which is public domain! Here's an overview of some of the latest developments for our not-so-small computer.

TI-RLE — "Run Length Encoded" graphics have rapidly become the industry standard for high resolution graphics. Two programs have recently been written that will allow the TI-99/4A to convert an RLE file on disk to a picture that can be displayed on the screen or printed on the printer. Paul Gray has written 99RLE, a public domain Forth program that will do such a translation of RLE files, and Travis Watford has written an assembly language program for the same purpose.

Where can RLE files be found? Well, bulletin boards and major telecommunications networks are a good starting place to look. CompuServe, a major computer network (we'll be delving more into telecommunications in later articles), has quite a few RLE pictures available, from mugshots of the FBI's Ten Most Wanted to pictures of famous celebrities. Although many of these pictures are copyrighted, many are not, which means that they will soon be finding their way to bulletin boards and user group libraries as soon as Paul and Travis' translation programs do!

This is really a major advancement for the TI-99/4A, not to mention an impressive enhancement. Figure 1 is an RLE picture of W.C. Fields, downloaded from CompuServe's TI Forum, which demonstrates the resolution of RLE graphcis. Paul Charlton, author of the "fairware" FAST-TERM terminal emulator, says that plans for adding RLE protocol to his program are in the works. Adding RLE display abilities to a terminal emulator means that RLE pictures could be displayed and printed as they are received from a network or BBS, eliminating the need to receive what looks like junk and then translate it with one of the translator programs. "It's number six on my list," says Paul, "after writing a new operating system and taking care of a few other tasks." A promise was extracted from Paul to release a new version of FAST-TERM as soon as possible.

Other programs include a program being developed by Coe Case to actually copy IBM-format disks (MS-DOS or PC-DOS) to TI format with a TI computer. Coe has successfully taken a document created with Wordstar on a PC and transferred it to a TI-99/4A with his program. More work needs to be done, as Coe needed to create a directory and bit map on sector 0 of the new TI-format disk, as well as a file header on sector 1. There is certainly a lot of potential here, though, allowing those 99/4A users with PCs of some sort at work or elsewhere to set up some kind of meaningful dialog between the computers. We will keep you updated on progress with this program!

Yet another interesting enhancement for the TI-99/4A has been the discovery by Howie Rosenberg of "DCOPY," a public domain disk copier that not only will work with the Myarc 128 or 512K RAMdisk card, but will also ARCHIVE a group of files in a directory. The idea behind archiving files is to take any number of separate files and place them all under one filename, seemingly as one DISPLAY/FIXED 128 file. Using DCOPY, the one file can then be ungrouped again into its original parts.

This is especially convenient for transferring files to a BBVS or other telecommunications network, since only one file need be downloaded instead of several, eliminating the possibility of missing a necessary file in a series. The archiving process adds two sectors to the combined length of all the files, so there is no actual compression of the data. The ability to compress files (making them unusable until uncompressed, but smaller until then) would mean that less time would be necessary for transferring files to telecommunications networks, meaning a savings in money for pay services and allowing more users to access a BBS in a given time, since less time would be needed for each user to send or receive a file.

Although we do not have compression ability yet, the archiving feature of DCOPY should be very useful. DCOPY, like RLE, should be making its way to local bulletin boards and TI users groups very soon.

It is incredible to see such continuing interest in and development for the TI-99/4A. These new programs are a reminder that our computer is far from dead or abandoned; we are doing just fine. ●