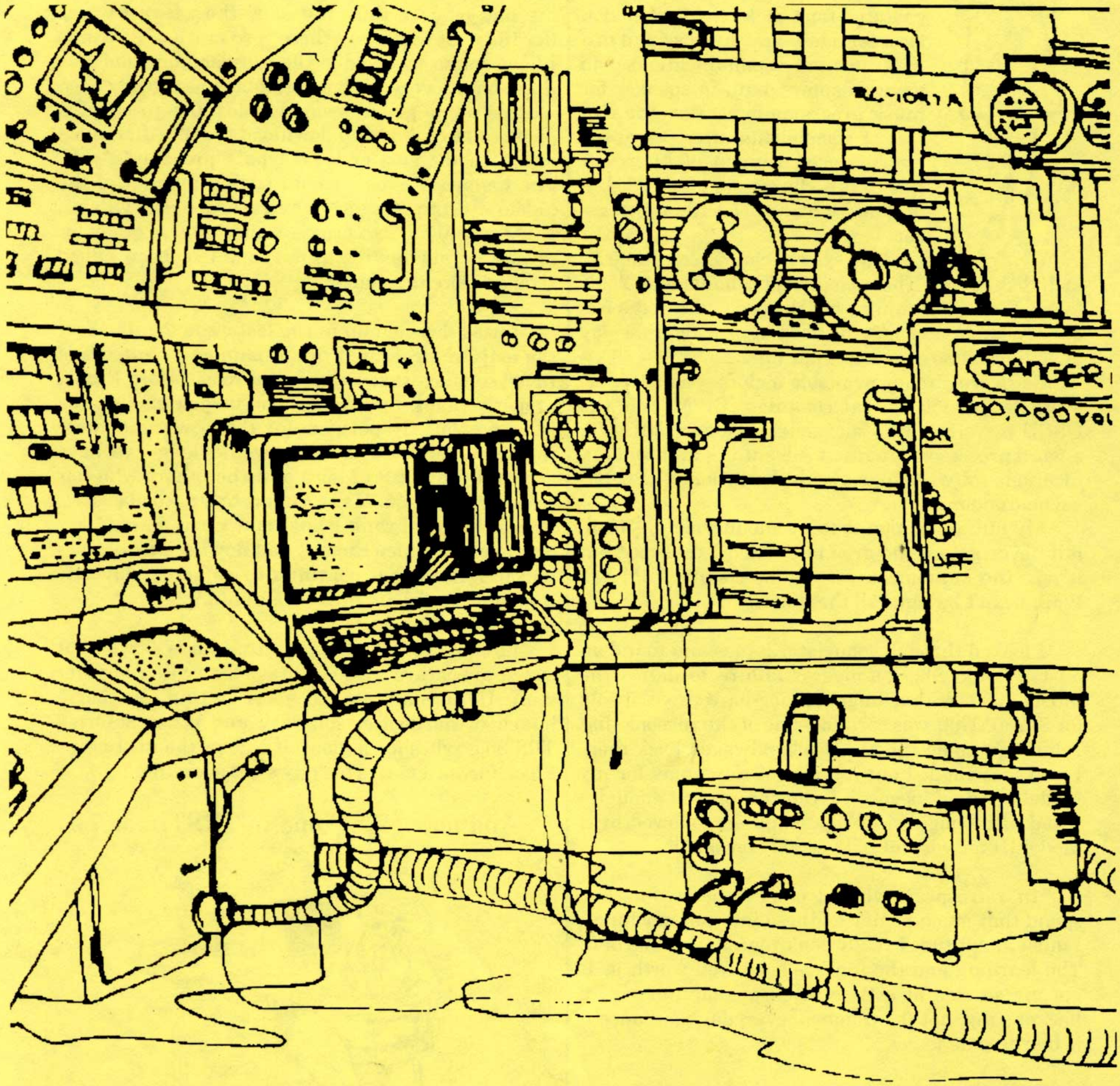
TRSTimes

Volume 7. No. 4 - Jun/Aug 1994 - $4.00



THE TRSTimes EDITORIAL OFFICE

LITTLE ORPHAN EIGHTY

The other day, while reading through an old issue of 80 Micro, I came across the following description of a brand-new computer, not yet released.

"Standard equipment includes: a detachable 83-key keyboard; a cassette tape jack; five expansion slots for additional memory and display, printer, communications and game adapters; built-in speaker for music programming; automatic self-test of components after power-on; an enhanced version of Microsoft Basic; 16K characters of user memory; and a high speed 16-bit processor.

The system can accept two 5.25 inch disk drives. The memory is expandable to 256K bytes. Either a monitor or a television with radio frequency modulator may be used to provide a display of 25 lines of 80 characters per line.

Software already available includes: a disk operating system; a Pascal compiler; CP/M and the USCD p-system; VisiCalc: general ledger; A/R; A/P; a word processor; Microsoft Adventure; and communications software to use with the optional RS-232C asynchronous adapter."

My initial reaction to this announcement - "Nah, it'll never get off the ground!" - then I realized that it was the announcement for the coming IBM-PC. Well, I can't be right all the time!

I looked through some later issues and found an ad for Radio Shack's biggest failure to date - the Model 2000. With a 'huge' 10 meg hard drive, it sold for $4250. That was certainly one of the reasons that I didn't jump on the MS-Dos bandwagon back then. I had just shelled out a series of tidy sums for my Model I and all its goodies. But that wasn't enough - I had also bought a Color Computer, followed by a Model III for which I paid $3000 - on sale!!!

In retrospect, while I can't believe that I've spend that much money on these infernal machines, I must admit that I really don't regret one dime of it. The learning and the fun has been well worth it. It got me started at a time when a computer was a hacker's toy, not the common, everyday appliance it is has become today.

Back then, if you wanted software, you'd better learn to write it yourself. We overcame many obstacles - badly designed hardware, lousy operating systems, expensive software that didn't work until you rewrote a good portion of it - and those were the good times!

I really shudder to think how I would feel about computers if I had not had these early experiences. Starting out from scratch today must be overwhelming, and yet everything is easier - the software works (for the most part), and there is so much of it. Would I have gotten involved in the programming end now that almost everything imaginable has already been written (and given away for for the price of a phonecall to a BBS and download time)? After much reflection the answer is "maybe", "probably", "yes". The happiest times I spend sitting in front of the Model 4 or my 486 are when I use Basic, QBasic, EDAS, or MASM to create something or other for myself or my family. But then, I've been called strange more than once!

I used this column in the last issue to talk about the evils of structured programming. I ranted and raved so much that I ran out of room before I could properly thank the people who helped making the issue possible. I apologize for that - without these fine individuals, TRSTimes would long ago have gone the way of 80 Micro and the others who has ceased publication. Many thanks to Dr. Allen Jacobs, Daniel Myers, Frank Slinkman, Chris Fara, Gary Shanafelt. Mathieu Simons and Roy Beck for articles and programs that continue to make our favorite computer useful, productive and fun.

Finally, Roy is telling me that he is slowly, but surely going through his storage bins with computer stuff. He needs room, so he is getting together a bunch of hardware, software and other assorted TRS-80 goodies for a blowout sale in the next issue. I look forward to it. This guy's got everything.

And now...Welcome to TRSTimes 7.4.

TRSTimes magazine

Volume 7. No. 4 - Jul/Aug 1994

PUBLISHER-EDITOR
Lance Wolstrup
CONTRIBUTING EDITORS
Roy T. Beck
Dr. Allen Jacobs
TECHNICAL ASSISTANCE
San Gabriel Tandy Users Group
Valley TRS-80 Users Group
Valley Hackers' TRS-80 Users Group

TRSTimes is published bi-monthly by TRSTimes Publications. 5721 Topanga Canyon Blvd., Suite 4. Woodland Hills, CA 91367. U.S.A. (818) 716-7154.

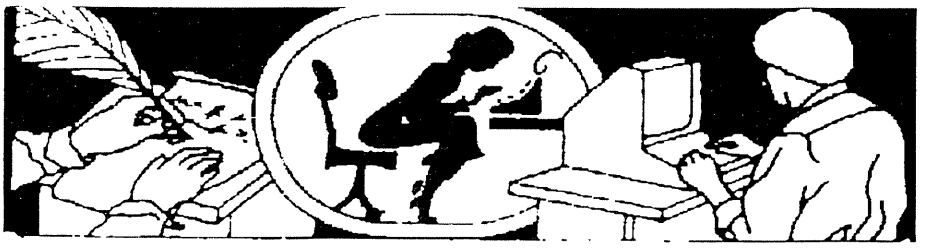
Publication months are January, March, May, July, September and November.

1994 subscription rates (6 issues):
UNITED STATES & CANADA:
$20.00 (U.S. currency)

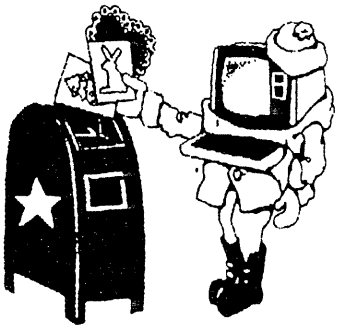
EUROPE, CENTRAL & SOUTH AMERICA:
$24.00 for surface mail or $31.00 for air mail. (U.S. currency only)

ASIA, AUSTRALIA & NEW ZEALAND:
$26.00 for surface mail or $34.00 for air mail. (U.S. currency only)

Article submissions from our readers are welcomed and encouraged. Anything pertaining to the TRS-80 will be evaluated for possible publication. Please send hardcopy and, if at all possible a disk with the material saved in ASCII format. Any disk format is acceptable, but please note on label which format is used.



THE MAIL ROOM



THE *GOTO* CONTROVERSY

Thanks for the newest TRSTimes magazine. Lance, as an educator, I must take you to task on your essay (*Little Orphan Eighty 7.3*). I do not program and only know enough programming to maybe control my printer; but I use computers all the time. I am the computer coordinator for my building at school and certainly am the most knowledgeable computer person in our moderately large school district. I use the computer to do free-lance writing and have paid for all of my computer equipment by writing. I also am about to embark on some new computer equipment of my own, and that is finally stepping up to some used Macintosh equipment and a Stylewriter. Now, Lance, I am not sure who you refer to as the "... famous radio personality as 'the Dumbing of America'.", but you know and I know that America is not dumbing down.

Lance, I will put the top seniors across the U.S. up against the top seniors in your high school class and, brother, yours and mine wouldn't be in the dog race. Without computer technology, our family simply would not have been able to keep pace with my children's academic needs and demands.

Your problem has nothing to do with the "dumbing down" of America; yours is a problem with one specific teacher/programmer who shares some different programming values as you. But, Brother Lance, that is what our children have to deal with, and, yes they have to adapt to every teacher. I suggested to my daughter, Holli, to add an extra paragraph on her senior research paper on literacy. It sounded good to me, no better. When she got it back, the teacher said that my paragraph was the weakest part of her research paper and should have been deleted. The teacher is excellent and received the Teacher Of the Year Award for our school district. She is awesome. I would never criticize her for our academic differences. And you know what, Lance, after rereading Holli's paper, Mrs. Taylor was probably correct.

Hang in there guy, but generalizations about many, derived from one person's experience with one professor, sounds like something Rush Limbaugh does. The man is a certified user who couldn't hack college and uses Republicans and conservatives to make bunches of money.

Dale Hill
Washita, OK

Dale, first let me say that it pleases me that you are proud of the education your children are getting from your school district. This is the way things ought to be - this is the way things used to be, but, in my opinion, it is certainly not the way things are now. My statement of 'dumbing down' was not a generalization, it is, indeed, based on statistics available to anyone interested - the Scholastic Aptitude Test Scores (SAT). In the twenty-year period of 1971 to 1991 the California Verbal test average dropped from 464 to 415. That is 49 points, or better than 10%. The national average dropped from 452 to 422. That is 30 points. The Mathematics test scores went from 493 to 482 in California, and from 484 to 474 nationally. The reasons for this are manyfold, but since I promised myself when TRSTimes was concieved not to use it as a political vehicle, I will leave well enough alone and let the numbers speak for themselves.

Ed.

Your outrage at the persecution of "GOTO" in programming was right on the money! The fanatical ideology of "structured" programming is a digital equivalent of "political correctness".

Go to!
Chris Fara
Tucson, AZ

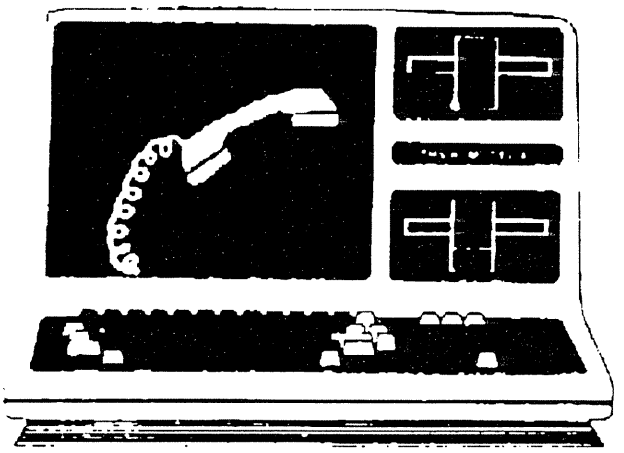
ENJOYING THE TRS-80

I am really enjoying my TRS-80 Model 4 and TRSTimes. Many thanks to Daniel Myers for his fine articles. I have now been further in the Infocom games that I've ever been. Also thanks to Mr. Slinkman for his C tutorial. I can program a little bit in Basic and will try to learn C. All in all, thanks for an excellent publication with excellent writers.

Walter J. Marakov
Middletown, OH

ADVENTURES ON THE INTERNET

by Gary W. Shanafelt
Department of History, McMurry University, Abilene, TX 79697



Suddenly, everyone is interested in the Internet. It's been around for quite a while, but in the last year or so with talk of the Information Highway, it seems to have come into its own. More and more people are logging on, at least people with PCs or MACs. What about TRS-80s? No problem -- you don't need a new computer to join the latest wave in the information revolution.

WHAT IT IS

If you've never heard of the Internet, it's roughly a network connecting mainframe computers around the country. It started out connecting mainly government computers, then it was extended to universities. No one knows exactly how many systems are jerry-chained together or what information is available on all the systems -- except that it is a lot. There are even specialized programs to help you find it, with exotic names like ``gopher'' and ``wais.'' And it is fast. Transferring e-mail or files is almost instantaneous. It's also cheap. Once you're on, you're not accessed massive long distance phone charges, even if you're connected to computers thousands of miles away.

The Internet's main hassle is getting on. Since it is a network of mainframes, you have to have access to a mainframe that's part of the network. Some of the major information services like America OnLine are offering connections, but of course that will cost you more money than if you can gain access through a government or school computer. From the mainframe, you simply turn on your terminal and call up whatever Internet access software the mainframe uses to make its connections. Or if you're somewhere else, you use your PC to log onto the mainframe the same way you would log onto a BBS, with a modem and a terminal package. Once you're logged on, you can then connect to most any other computer system included in the network.

The above is covered in numerous books, mostly published in the last year or two, so I won't repeat it all here. The point of this article is to discuss how you do it with a TRS-80, or at least how I've done it with mine.

WHAT YOU NEED

First, you need access to a mainframe computer connected to the Internet, or an information service that gives you access. In my case, I can log onto the mainframe computer of the college where I work, a DEC VAX running the VMS operating system. This has led to different challenges than you might have with an IBM or something running UNIX, though I can't comment on either of those systems since VMS is the only one I've had to worry about so far. Of course, you need a telephone line and a modem. Finally, you need a Model 4.

When I connect with my TRS-80, I'm essentially turning it into a dumb terminal, sending commands to our campus VAX which executes them for me. If I then connect to some other system on the Internet, my TRS-80 becomes a terminal for it. It might be another VMS system, or IBM, or UNIX. That's a lot of variety.

How do you maintain any kind of command com-

patibility? I've found that most systems you might connect to default to VT100 terminal emulation. That is, they assume you can emulate the command set of this common type of terminal, which is roughly the same as the ANSI commands of the IBM PCs.

Can you do this on a Model 1 or 3? Not having a Model 1 or 3, I can't say for sure. There is a freeware Model 3 terminal program called Ansiterm, which requires a hi-res board, and which should support some of the VT100 commands. But since a VT100 has an 80-column display and the Model 1/3 only 64, the amount of compatibility may not be very high. You'll have much better prospects with a Model 4. The Model 4 not only gives you 80 columns of text; it also offers a number of terminal packages that include VT100 emulation. The degree of emulation differs from program to program. Inverse video is handled differently on different programs, for example.

The terminal program I use is Mel Patrick's FastTerm II. It has both normal connection mode and VT100 emulation mode.

In normal mode, which is all you get with old standards like Bill Andrus' XT4 terminal program, the special VT100 commands sent by the host computer create a patchwork of numbers and bracket characters all over your screen, since the TRS-80 doesn't know what to do with them.

In VT100 mode, those commands reposition the cursor, move lines, and, in short, do what they're supposed to do on a VT100 terminal. The latest version of FastTerm II is version 4.65. You can download it from Mel Patrick's BBS at (604) 574-2072, in Canada. It used to be shareware but is now free. To get full use of it, you also need the manual, 43 pages of tightly-packed laser printing. Ask Mel how much it will cost for him to send you a copy.

Other VT100 emulation options are available if you don't like FastTerm or don't have it. If you have a hi-res graphics card, you can run Richard Van-Houten's Ansiterm 4, available from COMPUTER NEWS 80 for $30. This is a full-featured terminal program with the feature, unavailable anywhere else, of using the hi-res board to display IBM (rather than TRS-80) special characters. It is an enhancement of the Model 3 Ansiterm mentioned above. Richard has also written a brief program simply called VT100/CMD which is available on many BBSes.

None of the above programs will give you ``full'' VT100 emulation. Features like screen scrolling or non-destructive cursor movement don't seem to have been implemented. But for most purposes they will serve your needs.

A number of older commercial programs from the 1980s also support VT100 emulation, if you can find a copy. Omniterm was the TRS-80 standard for years, and the later Omniterm Plus included VT100 emulation. Another commercial program, Teleterm, supported both VT100 and VT102 emulation. Unfortunately, neither of these programs seem to be available anymore. I phoned the latest numbers I could find from old issues of 80 MICRO, but the numbers had invariably been disconnected.

MAKING THE CONNECTION

One if the nicer features of FastTerm in addition to automatic dialing (which any good TRS-80 terminal program has) is its scripting language. This allows you to automate the whole process of logging on to another system, for once the automatic dialler connects you to it, FastTerm then executes the commands in your script, one by one. A script with a terminal program works much like a JCL file from the DOS level. Anyhow, the script I use to log onto our VAX looks like this:

```
[WAIT FOR "Local>"
[SEND "C"
[WAIT FOR "Enter username>"
[SEND "GSHAN"
[WAIT FOR "Password:"
[SEND "--------" (my password)
[VT100
[CRLF ON
[CLS
[LWAIT FOR "NO CARRIER"
[CRLF OFF
[DEFAULT
[DTR
[END
```

Most of this is self-explanatory. After the initial login dialog, the script switches FastTerm to VT100 emulation mode and turns on CRLF (otherwise you get double spacing). When I'm done and log off, the modem generates a ``no carrier'' message which causes the script to switch back to normal terminal and CRLF mode.

The nice thing about FastTerm's scripts is that they do all this automatically, for it's otherwise a fair amount of keystrokes to change emulation and CRLF modes.

I suppose if all you do is call the Internet, you can make VT100 the default and just save it in your FastTerm configuration file. But I call other numbers which do NOT require VT100 emulation and so the script saves me a lot of reconfiguration hassles. I have a different script, for example, for logging onto CompuServe.

On our VAX, once you're successfully logged on, you get a ``$" prompt. To connect to another computer system on the Internet, you issue the TELNET command.

To get on the Library of Congress, for example, involves typing: TELNET LOCIS.LOC.GOV, ``LOCIS.LOC.GOV" being the L.C. computer name. Usually when you're asked for a login name, GUEST or ANONYMOUS will get you on. (Incidentally, if you can log on to the Library of Congress during normal business hours, it will be a miracle; the best time to connect with many areas of the Internet is at night or weekends, when the traffic is down).

Transferring files is handled with the FTP (``file transfer protocol") command. It took me a while to figure out how to do this with my Model 4. The command itself is easy enough; if you want to send or receive a file from, say, the University of Oakland computer, where a lot of files are stored, you type FTP OAK.OAKLAND.EDU. This is the same syntax as TELNET.

You then go through the same login procedure. The FTP default is for transferring ASCII files: to download a binary file (anything that is not raw text), you issue the command BINARY. A lot of files are binary: anything that has been archived and has a .ZIP extension is binary.

You may have to switch subdirectories to get where you want to go on the host mainframe: its DIR and CD commands will usually do this. Finally, you type GET and the name of the file. What this does is download or transfer the file from the FTP host to the mainframe you called up when you first started FastTerm, in this case our school VAX.

FTP handles the transfer so fast that you'll initially think there was an error that aborted the program. There wasn't: this is what dedicated communication lines are all about.

FTP is actually pretty simple, and is described in all the Internet manuals. What I found to be not so simple was getting the file from our VAX to my Model 4. The problem was that the only transfer program the VAX had for moving files between it and PCs was Kermit.

FastTerm supports XMODEM, 1K XMODEM, and YMODEM -- but not Kermit. Nor do AnsiTerm or Omniterm. I figured there must be a VMS XMODEM program available somewhere, so I logged on to the CompuServe VAXFORUM and, sure enough, there were several.

None of them, though, is very easy to use, thanks to the peculiarities of the file structure of the VMS operating system. It turns out that binary (.EXE) files are stored on the VAX in fixed-length blocks of 512 bytes each. XMODEM transfers data in blocks of 128 bytes.

For some reason, all the VAX XMODEM programs I have so far found expect the 512-byte block files to be converted to 128-byte block files before they will work properly. Otherwise, they only transfer one forth of the file and terminate!

VAX programmers wrote separate utilities to make the conversion (EXETOXMOD.EXE converts from 512 to 128 bytes; XMODTOEXE.EXE converts from 128 to 512 bytes). Why they couldn't write one program that would do it all is beyond me (but the more contact I have with mainframes and their programs, the more I understand why the PC was invented For The Rest of Us).

In any case, the procedure I have to use to get a binary file on the VAX downloaded to my Model 4 is the following: first you convert the file to 128-byte blocks, then you transfer it with XMODEM on the VAX through FastTerm's XMODEM option to your disk. If you want to upload a binary file, you reverse the procedure: upload with XMODEM, then run the file through the second conversion program which processes it from 128-byte blocks to 512-byte blocks. You're now ready to use FTP to send it on to someone else.

There's one more complication. For reasons inexplicable to all except the gurus of VMS, if you try to transfer a text file with the binary options of VMS XMODEM, you'll get every line ending in a carriage return filled out with extraneous space characters. So, the VMS XMODEM asks you to specify whether you're handling a text or binary file.

You can tell pretty easily by issuing the DIR/FULL command on the VAX: this gives you a file directory with a full slate of information about each file, including the block size. Consider anything with 512-byte fixed-length blocks to be binary, and everything else text. I find all this ridiculous. And people call LS-DOS an obsolete operating system! Unfortunately, though, unless someone writes a decent VMS XMODEM program, we're stuck with it.

A lot of my time on the Internet has been spent with e-mail. At this point, Internet mailing will accept only 7-bit (ASCII) files. That is, you can send a letter with no special characters or codes, to someone, but you can't send a binary file. (Nor can you do it with FTP, for that connects with the directory of the host computer, not an individual user's personal address).

If you want to send a binary file direct to someone else's Internet address, you use two programs called UUENCODE.EXE and UUDECODE.EXE. The first converts a binary file into ASCII characters so you can send it as normal 7-bit e-mail to any address on the Internet. (Files so converted end with the extension .UUE). The second program decodes a UUENCODEd file back into its original binary code. For this to work, both the sender and the receiver have to have copies of the coding utilities.

While there are versions for MS-DOS, there is no version for the TRS-80. Luckily, though, there is a version for the VAX; I just have to remember to do any coding of files there and not after I've downloaded them to my Model 4.

WHY NOT KERMIT?

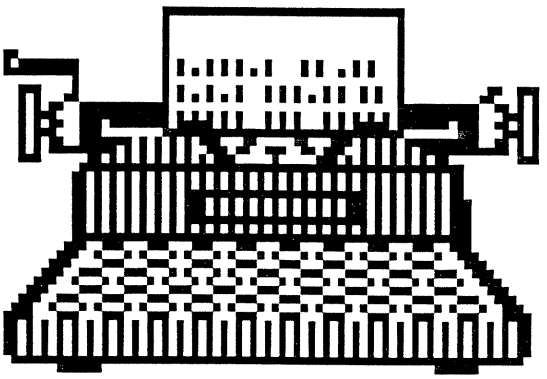
At this point, you're probably thinking, wouldn't it be a lot easier just to use Kermit, which our VAX had to begin with? Actually, Kermit on the VAX isn't that much easier to use than XMODEM, for you still have to specify whether you're processing a binary or ASCII file -- though at least you can dispense with all the nonsense about 512- vs. 128-byte file blocks.

The main problem with Kermit is that there's no up-to-date TRS-80 version. The last one I know of was written by Gregg Wonderly back in 1986, and it has two major shortcomings. First, it doesn't support VT100 emulation. The best it can handle is Heath 19 emulation, which is roughly akin to the DEC VT52 terminal. The VT52 was the predecessor to the VT100, which has pretty much replaced it as the minimal standard for terminal emulation. Few mainframe programs allow you to use it instead of VT100 mode (though on a standard VAX you can issue the command SET TERM/52 to get it).

The second problem is that Model 4 Kermit doesn't seem to allow binary file transfers. The documentation I have for version 4.x states that "binary does not work properly due to the fact that Model 4(p) Kermit does not do 8-bit quoting." This may have been fixed in a later version, but without the docs being updated. In any case, try as I may, I have never been able to transfer binary files between the Model 4 and our VAX via the copy of Kermit that I have. So, use FastTerm (or rewrite Kermit for the 1990s!)

I hope this gives you some idea how you can use the Model 4 to access the Internet. You can e-mail me there at GSHAN@MCM.ACU.EDU.

PROGRAMMING TIDBITS



CONFESSIONS OF A RELUCTANT PROGRAMMER

In the beginning was my father's typewriter, an antique black affair with round keycaps. In a cavity in front of the platen there was a fan-like array of skinny rods with cast-metal types at their tips. A brisk keystroke would activate a rod via a wondrous system of levers and the type would hit the paper through a fabric ribbon to produce a somewhat fuzzy character. The contraption was a source of endless fascination for a schoolboy I was then. I wonder if my father ever knew how often in his absence I abused his machine. Okay, Dad, I confess: at least once a week.

Typewriters became an obsession throughout my school and college days, but I could not afford my own until I had my first full-time job. With a brand new Olivetti on my desk, I felt I had arrived.

My happiness was shattered during one fateful stroll in front of a Radio Shack store. In the window there was something that looked like a cross between television and typewriter. Typewriter? Naturally I had to look it up. As a result Tandy made a small profit and I became a bewildered owner of Model I. Oh yes, in college I had some computer training, mostly Fortran, but it never turned me on. It was way too cumbersome, what with mainframe time-sharing, endless waiting for "queued" printouts, not to mention the Fortran code whose vocabulary and grammar looked downright criminal to a Fine Arts student I was then.

Model I, however, was a different animal altogether. For one thing, it was personal, like my typewriter. I could turn it on and off any time of the day and night, without calling ahead for a time slot at the "terminal", without "logging", without passwords. Then there was BASIC. Still not quite Shakespeare, but in comparison with Fortran almost legible, and I was willing to accept it as a reasonable approximation of a "language". Best of all, it produced instant results in front of my beady eyes the moment I'd press the "Enter" key. Over the next few weeks Tandy made some more profit, until I had assembled the works: an awesome 64-K of memory, "expansion interface", two disk drives and a printer.

At that time I was gainfully employed in a company where one of my most hated chores was the preparation of monthly productivity reports and budget forecasts for my department. A helpful Radio Shack salesman persuaded me that Visicalc was just what I needed. But it soon became apparent that forcing Visicalc to fit my company's forms was even more of a chore than the old pencil-and-eraser method. Since the accounting calcs were pretty simple, I ventured into writing my first "real" program. Looking back at that program I blush. The code was a disaster. Still, thanks to BASIC's rich suite of string formatting functions the printed reports looked quite impressive. And the mere fact that they came from a computer (from a computer!) gave them an instant credibility with my boss.

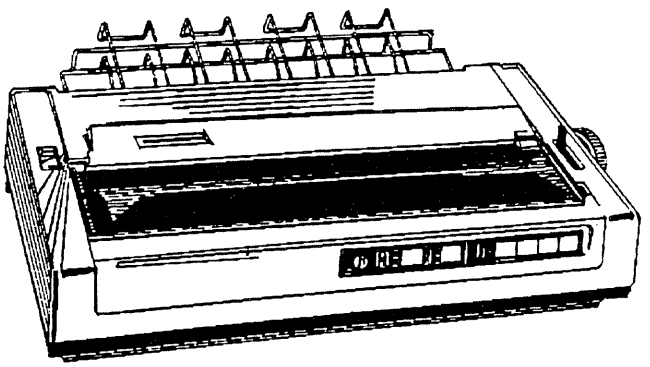
A couple of years later I found myself in need of a Computer Aided Drafting software. By then I had already a Model III and Radio Shack just came up with the high-res board. In my naivety I expected to walk into the store and buy a CAD program for it. No way, experts assured me, this kind of program couldn't run on a TRS-80 computer. You need a more "powerful" machine, they opined, such as the "new" IBM PC. Having sunk a small fortune into TRS, I couldn't afford IBM. Yet I badly needed some sort of simple CAD, so I had no choice but to try something on my own. After many sleepless nights it did work, more or less. At least it was good enough for my immediate needs. Unfortunately, as the French say, appetite grows once you start eating, and I wouldn't stop fiddling with it until one day I realized that I had in my hand the "impossible"

thing, a fairly comprehensive CAD program for a TRS-80. Thus was born xT.CAD for Mod-III, soon thereafter replaced by an improved, much faster version for Mod-4.

The irony of it was that the support of xT.CAD took so much of my energy that I hardly had the time to use it for its original purpose which was to help with my drafting. I became a slave of my own creation. Like the proverbial "sorcerer's apprentice", having unleashed forces I didn't understand, I was sinking deeper and deeper into pixels, bits, bytes, nybbles, modules, overlays, drivers, filters, Boolean and other shifty manipulations, finally drowning in the assembly language. Assembly what? Language? "Hiccups" would be a better term for it, if you'd ask the Fine Arts student I used to be... but now I wore the label of a "programmer" and Microdex was my middle name!

Labels have a strange habit: they stick and define their own reality. One thing leads to another, whether you ask for it or not. A fellow calls for help with his JCL files. No wonder. His expectations are far beyond what JCL can do. After trying without success to excuse myself from this unsolicited business, I end up writing the "Direct from Chris" menu system. Similar story with "Clan". Originally it surfaced in form of a request to "patch" Mod-III version so it would run in Mod-4 mode. For almost a year I did my best to avoid the issue and was happy to hear periodic rumors that someone or another was already working on it. In the end no one did and since I am no good at "patching", I got stuck with writing Clan-4 "from scratch".

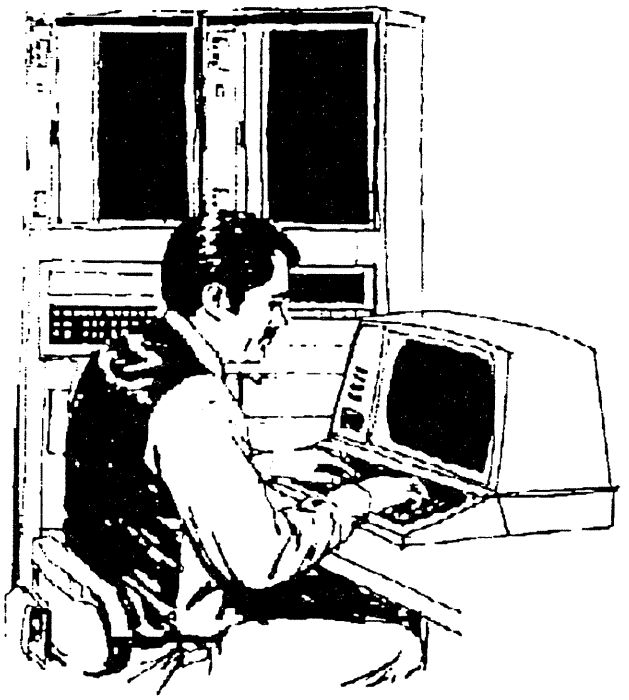
Of course I am not sorry. All those digital adventures have their rewards. For instance it's wonderful to see that "Direct from Chris" seems to have inspired an "aftermarket" of add-ons and enhancements. But sometimes, deep in my heart, I miss that old black typewriter with the round keycaps.



MAKFILE FOR NEWDOS/80

Model I/III

by Mathieu Simons



The November/December issue of TRSTimes magazine presented a program written by Gary Shanafelt and Lance Wolstrup to allow creation of /DSK files compatible with Jeff Vavasour's Model I Emulator. It was written on LDOS and, as such, used LDOS specific CALLs. Unfortunately, this made it incompatible with NEWDOS/80.

To rectify this oversight, Mathieu Simons of Belgium, got out his editor/assembler and did just what a self-respecting hacker does — he modified the program to work exclusively in NEWDOS/80.

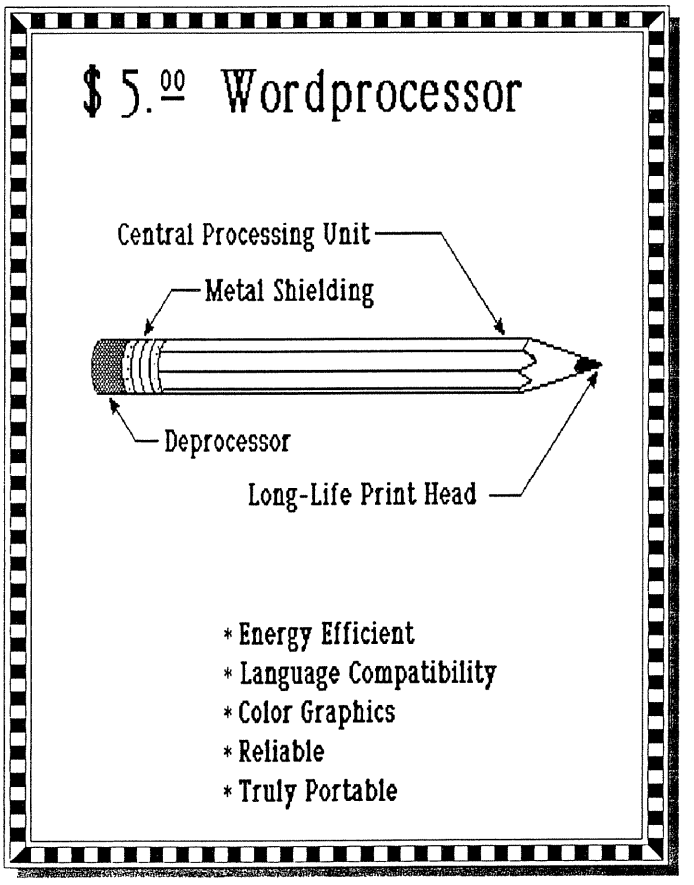
So, get out the NEWDOS/80 version of EDTASM and take advantage of Mathieu's hard work. Type in the program, assemble it, and you'll have the ability to use Model I/III NEWDOS/80 as your transfer medium on the way to the PC Model I Emulator.

```
        ORG   7000H
START CALL    1C9H          ;cls
        LD    A,(293)       ;test for model
        CP    73            ;is it Mod III
        JR    Z,MOD3        ;yes - jump
        XOR   A             ;set up Mod I
        LD    (COPYRG),A    ;nop
        LD    A,31H         ;set to "1"
        JR    MOD13         ;and jump
MOD3  LD    A,33H         ;set to "3'
MOD13 LD    (MODEL),A     ;and stuff into text
        LD    HL,HELLO$     ;point to header msg
        CALL  4467H         ;and display
ASKSRC LD   A,0FFH        ;set Dec trkcount to 0
        LD    (DDTRK),A     ;stuff it in buffer
        LD    HL,0606H      ;print@(6,6)
        CALL  LOCATE        ;position cursor
        LD    HL,SRCMSG     ;point to src prompt
        CALL  4467H         ;and display it
        LD    A,1           ;set default
        LD    (SRC),A       ;and store it
        LD    HL,BUFFER     ;point to buffer
        LD    BC,0100H      ;max input chr=1
        CALL  40H           ;@keyin
        JP    C,EXIT        ;exit if break
        LD    HL,BUFFER     ;point to input
        LD    A,(HL)        ;and get it
        CP    13            ;is it enter?
        JR    Z,ASKDST      ;default, so jump
        CP    30H           ;is it 0?
        JR    C,ASKSRC      ;jump if less
        CP    38H           ;jump if
        JR    NC,ASKSRC     ;larger than 7
        SUB   30H           ;strip ascii
        LD    (SRC),A       ;and store in buffer
ASKDST LD   HL,0800H      ;print@(8,0)
        CALL  LOCATE        ;position cursor
        LD    HL,DSTMSG     ;point to dest prompt
        CALL  4467H         ;and display it
        LD    A,2           ;set default
        LD    (DST),A       ;and store it
        LD    HL,BUFFER     ;point to buffer
        LD    BC,0100H      ;max chat input is 1
        CALL  40H           ;@keyin
        JR    C,ASKSRC      ;prev prompt if break
        LD    HL,BUFFER     ;point to input
        LD    A,(HL)        ;and get it
        CP    13            ;is it enter
        JR    Z,ASKTRK      ;jump if default
        CP    30H           ;is it 0
        JR    C,ASKDST      ;jump if less
        CP    38H           ;is it larger than 7
        JR    NC,ASKDST     ;jump is so
        SUB   30H           ;strip ascii
        LD    B,A           ;dst drv num to B
        LD    A,(SRC)       ;get src drv num
        CP    B             ;compare them
        JR    NZ,SETDST     ;jump if different
;
```

```
DRVERR LD   HL,0D00H    ;print@(13,0)
       CALL LOCATE      ;position cursor
;
       LD   HL,SRCDST   ;point to error msg
       CALL 4467H       ;and display it
;
       CALL 49H         ;@key
;
       CP   13          ;is it enter
       JR   NZ,DRVERR   ;not enter
       JR   ASKDST      ;dest drv num
;
SETDST LD   A,B         ;retrieve dst drv num
       LD   (DST),A     ;and store it
ASKTRK LD   HL,0A08H    ;print@(10,8)
       CALL LOCATE      ;position cursor
;
       LD   HL,TRKMSG   ;point to trk prompt
       CALL 4467H       ;and display it
;
       LD   A,35        ;set up default
       LD   (TRK),A     ;and store it
;
       LD   HL,BUFFER   ;point to input buffer
       LD   BC,0200H    ;max input chars=2
       CALL 40H         ;@keyin
;
       JR   C,ASKDST    ;jump if break
       LD   A,B         ;a=num of key stokes
       OR   A           ;any there
       JR   Z,ASKNAM    ;jump if default
;
       LD   HL,BUFFER   ;point to input
       CALL DECHEX      ;convert to hex
;
       LD   A,C         ;xfer hex num to A
       CP   10          ;is it 10
       JR   C,ASKTRK    ;jump if less
       CP   81          ;is it larger than 80
       JR   NC,ASKTRK   ;jump if yes
;
       LD   (TRK),A     ;store num of tracks
;
ASKNAM LD   HL,0C0FH    ;print@(12,15)
       CALL LOCATE      ;position cursor
       LD   HL,NAMMSG   ;point to name msg
       CALL 4467H       ;and display it
;
       LD   HL,BUFFER   ;point to input buffer
       LD   BC,1800H    ;max chars=23+cr
       CALL 40H         ;@keyin
;
       JR   C,ASKTRK    ;prev prompt if break
;
       LD   A,B         ;num of chars input
       OR   A           ;any input there
       JR   Z,ASKNAM    ;filename - jump
;
PUTDN  INC  HL          ;find end of filename
       LD   A,(HL)      ;get char
       CP   ':'         ;is drv num attached
       JR   Z,PUTDN1    ;yes - so skip colon
       DJNZ PUTDN
;
       LD   A,':'       ;append colon
       LD   (HL),A      ;to filename
PUTDN1 INC  HL          ;next position
       LD   A,(DST)     ;get dst drv num
       ADD  A,30H       ;make it ascii
       LD   (HL),A      ;append it
       INC  HL          ;next position
       LD   A,13        ;append
       LD   (HL),A      terminator
;
ASKRDY LD   HL,0E00H    ;print@(14,0)
       CALL LOCATE      ;position cursor
;
       LD   HL,RDYMSG   ;point to ready msg
       CALL 4467H       ;and display it
;
       CALL 49H         ;@key
;
       CP   1           ;is it break
       JR   Z,ASKNAM    ;prev prompt if break
;
       CP   13          ;is it enter
       JR   NZ,ASKRDY   ;no - ask again
;
       LD   HL,BUFFER   ;point to buffer
       LD   DE,FCB      ;point to FCB
       CALL 441CH       ;get filespec
;
       LD   HL,BUFFER   ;point to buffer
       LD   DE,FCB      ;point to FCB
       LD   B,0         ;LRL = 256
       CALL 4420H       ;open file
;
       LD   A,(TRK)     ;get trackcount
       LD   B,A         ;xfer to B
TLOOP  PUSH BC          ;save it
       LD   A,0FFH
DDTRK  EQU  $-1         ;track # to display
       INC  A           ;increment it
       LD   (DDTRK),A   ;and stuff in buffer
       LD   L,A         ;trk num to HL
       LD   H,0
       LD   DE,DECTRK   ;convert to decimal
       CALL HEXDEC      ;convert to dec ascii
       LD   HL,(DECTRK) ;get track #
       LD   (DECTRK2),HL ;save again for wrt
       LD   HL,0E00H    ;print@(14,0)
       CALL LOCATE      ;position cursor
       LD   HL,RDMSG    ;point to rdmsg
       CALL 4467H       ;dsplay 'read track #'
       LD   B,10        ;10 sectors
SLOOP  PUSH BC          ;save sector count
```

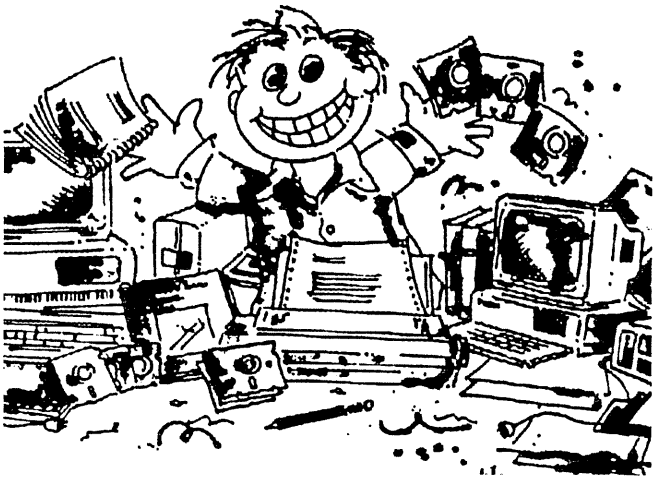
```
        LD      DE,DFCB     ;load disk DCB
        LD      HL,BUFFER   ;point to i/o buffer
        CALL    4436H       ;4777H read sector
        JR      Z,SLOOP1    ;jump if good read
        CP      6           ; dir read ???
        JR      NZ,ERROR2   ;no - so error
SLOOP1 PUSH DE              ;xfer sector
        POP     IX          ;to IX
        LD      DE,256      ;point to nrxt
        ADD     HL,DE       ;segment of buffer
        LD      (IX+3),L    ;store buffer
        LD      (IX+4),H    ;address
        LD      DE,DFCB     ;point to dfcb
        POP     BC          ;restore sector count
        DJNZ    SLOOP       ;and continue
        LD      HL,0E00H    ;print@(14,0)
        CALL    LOCATE      ;position cursor
        LD      HL,WRTMSG   ;point to write msg
        CALL    4467H       ;display 'writing ...'
        LD      B,10        ;10 sectors
LOOPW PUSHBC                ;save sector count
        LD      HL,BUFFER   ;point to i/o buffer
        LD      DE,FCB      ;point to FCB
        CALL    443CH       ;write with verify,
                            ;4439H is without
        JR      NZ,ERROR2   ;jump if error
        PUSH    DE          ;save FCB pointer
        POP     IX          ;copy it to IX
        LD      DE,256      ;figure new
        ADD     HL,DE       ;address
        LD      (IX+3),L    ;and store it
        LD      (IX+4),H    ;in DCB
        LD      DE,FCB      ;point to DCB
        POP     BC          ;restore sectr counter
        DJNZ    LOOPW       ;and continue
        POP     BC          ;restore track counter
        DJNZ    TLOOP       ;and continue
        LD      DE,FCB      ;point to FCB
        CALL    4428H       ;close file
;
ASKRPT LD       HL,0E00H    ;print@(14,0)
        CALL    LOCATE      ;position cursor
        LD      HL,OKMSG    ;point to oh\kmsg
        CALL    4467H       ;and display it
;
        CALL    49H         ;@key
        CP      1           ;is it break?
        JR      Z,EXIT      ;yes - jump to exit
        CP      13          ;is it enter?
        JP      Z,ASKSRC    ;back to 1st prompt
        JR      ASKRP       ;prompt again

EXIT    RET                 ;back to dos
;
DECHEX LD       A,(HL)      ;get tens digit
        SUB     30H         ;strip ascii
        ADD     A,A         ;2x
        LD      C,A         ;save 2x in C
        ADD     A,A         ;4x
        ADD     A,A         ;8x
        ADD     A,C         ;now 10x
        LD      C,A         ;copy to C
        INC     HL          ;point to ones digit
        LD      A,(HL)      ;get tens digit
        SUB     30H         ;strip ascii
        ADD     A,C         ;we have hex number
        LD      C,A         ;to C to be compatible
        RET                 ;return to caller
ERROR2 LD       DE,FCB      ;point to FCB
        CALL    4428H       ;close file
;
ERROR POP       DE
        POP     BC
        OR      192         ;set bits 6 & 7
        LD      C,A
        LD      HL,0E00H    ;print@(14,0)
        CALL    LOCATE      ;positi0on cursor
        CALL    4409H       ;@error
        JR      EXIT        ;and exit to dos
;
HEXDEC LD       A,L         ;get number
        LD      HL,TABLE    ;point to table
        LD      B,2         ;only need 2 digits
HEX0    PUSH    BC          ;save loop counter
        PUSH    AF          ;save number
        LD      B,0         ;sub loop counter
        LD      A,(HL)      ;get sub number
        LD      C,A         ;store it in C
        POP     AF          ;restore number
        PUSH    AF          ;and save it again
HEX1    SBC     A,C         ;subtract table value
        JR      C,NEXT      ;until below 0
        INC     B           ;sub good - inc countr
        JR      HEX1        ;and do it again
NEXT    PUSH    BC          ;save sub countr
        LD      A,B         ;copy it to A
        OR      A           ;is it zero
        JR      Z,NEXT1     ;yes - so jump
        XOR     A           ;A=0
HEX2    ADD     A,C         ;get the digits value
        DJNZ    HEX2
NEXT1 POP       BC          ;restore sub counter
        LD      C,A         ;copy digits val to C
        LD      A,B         ;copy number to A
        ADD     A,30H       ;make it ascii
        LD      (DE),A      ;store num in buffer
        POP     AF          ;restore original num
        SUB     C           ;subtract digit value
        INC     DE          ;next buffer location
        INC     HL          ;next table value
        POP     BC          ;restore loop counter
        DJNZ    HEX0
        RET
;
```

```
LOCATE PUSH BC
       PUSH DE
       LD   DE,15360      ;start of screen
       LD   B,0
       LD   C,L           ;store horiz value
       LD   A,H           ;get vert value
       OR   A             ;is it vert 0?
       JR   Z,NOMULT      ;yes - don't multiply
       LD   L,H           ;vert value in L
       LD   H,0           ;vert val now in HL
       LD   B,6           ;multiply HL by 64
MULT64 ADD HL,HL          ;
       DJNZ MULT64
NOMULT ADD HL,BC          ;add in horiz value
       ADD  HL,DE         ;add the mem offset
       LD   (4020H),HL    ;cursor set
       POP  DE
       POP  BC
       RET
;
HELLO$ DB  'MAKFILE'
MODEL DB   32
       DB   10
       DB   'Transfer Model I disk into a normal '
       DB   'data file',10
       DB   'By Lance Wolstrup and '
       DB   'Gary W.Shanafelt ',10
       DB   'Copyright ',21
COPYRG DB  239
       DB   21
       DB   '1993 V.2.0. All Rights reserved',10
       DB   'Adapted for Newdos/80 I/III '
       DB   'by Mathieu Simons ',10,13
;
SRCMSG DB  15,31
       DB   'Enter source drive number '
       DB   '(default = 1):',14,3
;
DSTMSG DB  15,31
       DB   'Enter destination drive number '
       DB   '(default = 2):',14,3
;
SRCDST DB  15
       DB   'The source and destination drives '
       DB   'cannot be the same ',10
       DB   '- press ENTER to continue ',14,3
;
TRKMSG DB  15,31
       DB   'Enter number of tracks '
       DB   '(default = 35):',14,3
;
NAMMSG DB 15,31
       DB   'Enter name of destination file: ',14,3
;
RDYMSG DB  15,31
       DB   'ENTER when source and '
       DB   'destination disks '
       DB   'are ready ',14,3
;
OKMSG  DB  15,31
       DB   'Disk now transferred to normal '
       DB   'data file - '
       DB   'press ENTER ',14,3
;
WRTMSG DB 15,31,'Writing and verifying track # '
DECTRK2 DM        ' '
       DB   13,3
RDMSG DB   15,31,'Reading track # '
DECTRK DM ' '
       DB   13,3
DST    DB   0
TRK    DB   0
TABLE DB    10,1
FCB    DS   32
DFCB   DB   82H            ;this is the Disk-FCB
       DB   04,20H
       DW   BUFFER
       DB   00
SRC    DB   0
       DB   0,0,0
       DW   0000
       DW   0FFFH
       DS   12H
BUFFER DS   2560
;
       END  START
```



KELLY'S TIPS

by Kelly Bates



FLAT RIBBON AND CONNECTORS

As I sit in front of my 4P with 128K of memory and 2 external drives, it occurs to me that my experience might be of value to some of the hardware hackers out there. So, since I have never seen an article on this subject, consider the following a short tutorial.

Flat ribbon comes in several varieties. Usually it is grey with one of the side conductors being red to indicate wire number 1. Another variety that I like is the multicolor type, where the number 1 wire is Brown, followed by Red, Orange, Yellow, Green, Blue, Violet, Grey, White, Black, and then the sequence repeats. The color coding is the result of standardization - an easy way to remember the color sequence is that it is the same for resistor bands - and a phrase we were taught way back then was "Bad Boys Rape Our Young Girls But Violet Gives Willingly." The first B stood for Black (or Zero), the second B stood for Brown (or One), R stood for Red (or 2), and so on. On the flat ribbon there is no Zero wire, so number 1 is brown.

The flat ribbon, however built, is multistranded. The amount of strands vary, which is why some cables feel stiffer than others. Radio Shack ribbon cable always felt more flexible, so they probably have fewer strands. I think the standard is 20 gauge - but get 3M, as that is the best.

Now examine both sides of the cable - one side is flat, or at least flatter than the other. This oddity will let you fit it easier to the connectors. Try both sides - it will work either way. Ribbon cables comes in many widths, so you have to ask for it by the number of conductors, ie; "Give me 4 feet of 20 conductor flat ribbon, please".

So much for wire - now let's discuss the connectors.

The most common one we use is the 34 pin Edge card connector. That is the one we connect to on the rear of our drives (not the power plug). This connector can be mounted to the flat ribbon 4 different ways. Say what?? Let me explain - the face of the connector has a slot, it connects to an Edge card connection on the rear of the drive. Look at the opposite side of the connector. This is the side that mounts to the flat ribbon.

Here come 4 ways - real fast. Place the connector against the wire on one end - it will fit, right? Now do it to the other side of the cable. That's 2. The cable has another end - do those 2 steps there and I believe that is 4 ways to put a connector on the same cable - only 2 of which are correct.

On the connector at one end is a raised triangle. It means that this is the end to connect #1 wire to. It is a simple of maintaining orientation. When you put another connector on the other end of the cable and reference the triangle, then you do not even have to look at the first one, except to note which side of the cable we are mounting to.

Now, if you remove the back of the connector so that we can mount it to the flat ribbon, you will see a group of Y-shaped connections. Our cable wires must mesh one conductor wire to one Y. If improperly positioned, we will have many 'shorts' because we have part of two conductors in one Y. You will note they are staggered. 17 on one side and 17 on the other. The distance between the pairs of pins is one-tenth of an inch, so a flat ribbon one inch wide has 20 wires, and a 34 pin edge card connector will be about 2 inches long. The contacts in the connector on each side are one tenth of an inch apart. Standardization!

I frequently see where people try to reuse connectors. Bad practice! If just one of the Ys is spread, then the Y will not contact that conductor, because it is necessary to cut through the insulation

and pinch the wire for proper contact - and on reuse of wire, what if the Y cut one or two strands (the tiny wires inside each conductor) the first time the connector was used, the resistance of that conductor changes. Suit yourself!

To install a connector on the flat ribbon, get a hammer and 2 pieces of wood and work on a flat surface. Remove the back of the connector and By Hand start the assembly. Check to be sure you are meshing the wires into the connector properly. To this point, no harm is done - just realign. Next, put a piece of wood on your flat surface, lay the connector and flat ribbon on the wood, lay the other piece of wood on the face of the connector (yes, you must hold it), and then hit it with the hammer. Easy! Work your way across the connector, meshing a little at a time - and evenly. Check your cable and flat ribbon for the following: Look down the cable on the mesh side, and when you can no longer see the shiny Ys inside, then you have used the hammer enough. A better way is a Press, or perhaps Vice. The wood prevents hammer marks on the connector.

I use the this method because it is cheap, but it does take time, though. Basically all connectors that we use on the TRS-80 are the same, so the lesson here applies to them all.

HACKER'S DELIGHT

Have converted an old PC AT case for TRS-80 use. It had a standard AT motherboard with 640K installed. I removed all the guts except the power supply (150 watt with 4 drive connectors - just what I needed!)

The front of the case will access 4 slimlines or 2 of the full-height antigues - or as originally configured, a 20 meg hard drive and two 360K slimline floppies. I had lent the unit (operational with monitor) to a friend, but when it came back, the only thing that worked (besides the power supply) was the monitor. The cable from the CPU to the monitor was also missing, so I had to find one to see what would work. The top of the CPU case has actually been disconnected.

Anyway, no big loss! I installed my two 720K floppies on the right and two 360K slimlines on left, fixed the top rear of the cover so it would open and close properly - it's better now than when originally built. The case now has two release buttons for the cover - one on either side of the front. Release and open the cover (hinged) to expose the inside for access.

The case is 6" high, 19 1/2" wide (includes the 'feet') and 17 1/2" deep (includes the front bezel). On close examination, I discovered that I could put 5 1/4" floppies from back to front on the left side internally, or a combination of 3 1/2" and 5 1/4". Now the case doubles as a transport for floppies - or books, if so inclined.

The physical size of the case is perfect to sit a desktop Model 4 on top, which means that it does not take any space horizontally. Also, the drive power cables are long enough to extend outside the case to test other drives - a really fantastic external drive unit. Might mention that the color of the case matched the Model 4 decor.

So, what else could I install? One thing soon - an amplifier, cass cable and speaker for sound. That would let me plug the entire unit into a friend's computer. Got plenty of power, so I might as well use it! I have a standard cass cable, but I could build another easily, as I would only need two wires for the sound.

The daisy cable I built has an edge card connector and a 34-socket connector, which means that I could plug the unit into a 4P and actually boot my externals from it (with no mod to the 4P - only disassembly). Can you imagine, a 4P booting from a 3 1/2" 80 track floppy? All I have to do for the test is disconnect the 34-socket at the CPU and plug my externals in!

Now, what the above means is that you can format your 3 1/2" floppy as an 80 track boot disk in external mode, disassemble your 4P, hook up the externals to the 4P CPU, and test your 3 1/2" floppy to see if it will boot. Then install your 3 1/2" floppy in the 4P and close the case. Be sure to set up all your floppy DOS boot disks first.

The desktop Model 4 is a different ball of wax, though. On it, remove the top of the computer, remove the edge card connector on the top of the drive controller, and plug in the externals using the edge card connector.

I am setting up my boot disks at a leisurely pace. So far, I have TRSDOS 1.3 for Model III ready, as well as TRSDOS/LS-DOS 6 for Model 4. I am working on LDOS, NEWDOS/80 v2, and CP/M by Montezuma - and later, maybe Multidos and DosPlus.

ATTENTION TRSDOS 1.3 USERS!

ANNOUNCING "SYSTEM 1.5.", THE MOST COMPREHENSIVE 1.3. UPGRADE EVER OFFERED!

MORE SPEED!! MORE POWER!! NEW LOW PRICE!!

While maintaining 100% compatibility to TRSDOS 1.3., this upgrade advances DOS into the 90's!
SYSTEM 1.5. supports 16k-32k bank data storage and 4MGHZ clock speed (4/4P/4D).
DOUBLE SIDED DRIVES ARE NOW 100% UTILIZED! (all models).

config=y/n	creates config boot up	filedate=y.n	date boot up prompt on/off
time=y/n	time boot up prompt on/off	cursor='xx'	define boot up cursor character
blink=y/n	set cursor boot up default	caps=y/n	set key caps boot up default
line='xx'	set *pr lines boot up	wp=d.y/n	write protect any or all drives
alive=y/n	graphic monitor on/off	trace=y/n	turn sp monitor on/off
tron=y/n	add an improved tron	memory=y/n	basic free memory display monitor
type=b/h/y/n	high/bank type ahead on/off	fast	4 mghz speed (model 4)
slow	2 mghz speed (model 3)	basic2	enter rom basic (non-disk)
cpy (parm,parm)	copy/list/cat ldos type disks	sysres=h/b/'xx'	move/sys overlay(s) to hi/bank mem
sysres=y/n	disable/enable sysres	macro	define any key to macro
spool=h/b.size	spool is high or bank memory	spool=d.size='xx'	link mem spooling to disk file
spool=n	temporarily disable spooler	spool=y	reactivate disabled spooler
spool=reset	reset (nil) spool buffer	spool=open	opens, reactivates disk spooling
spool=close	closes spool disk file	filter *pr.adlf=y/n	add line feed before printing0dh
filter *pr.iglf	ignores 'extra' line feeds	filter *pr.hard=y/n	send 0ch to printer (fastest tof)
filter *pr.filter	adds 256 byte printer filter	filter *pr.orig	translate printer byte to chng
filter *pr.find	translate printer byte to chng	filter *pr.reset	reset printer filter table
filter *pr.lines	define number of lines per page	filter *pr.width	define printer line width
filter *pr.tmarg	adds top margin to printouts	filter *pr.bmarg	adds bottom margin to printout
filter *pr.page	number pages, set page number	filter *pr.route	sets printer routing on/off
filter *pr.tof	moves paper to top of form	filter *pr.newpg	set dcb line count to 1
filter *ki.echo	echo keys to the printer	filter *pr.macro	turn macro keys on/off
attrib :d password	change master password	device	displays current config

All parms above are installed using the new LIBRARY command SYSTEM (parm,parm). Other new LIB options include DBSIDE (enables double sided drive by treating the "other side" as a new independent drive, drives 0-7 supported) and SWAP (swap drive code table #s). Dump (CONFIG) all current high and/or bank memory data/routines and other current config to a disk data file. If your type ahead is active, you can (optional) store text in the type buffer, which is saved. During a boot, the config file is loaded back into high/bank memory and interrupts are recognized. After executing any active auto command, any stored type ahead data will be output. FANTASTIC! Convert your QWERTY keyboard to a DVORAK! Route printer output to the screen or your RS-232. Macro any key,even F1, F2 or F3. Load *01-*15 overlay(s) into high/bank memory for a memory only DOS! Enter data faster with the 256 byte type ahead option. Run 4MGHZ error free as clock, disk I/O routines are properly corrected! Spool printing to high/bank memory. Link spooling to disk (spooling updates DCB upon entering storage). Install up to 4 different debugging monitors. Print MS-DOS text files, ignoring those unwanted line feeds. Copy, Lprint, List or CATalog DOSPLUS, LS-DOS, LDOS or TRSDOS 6.x.x. files and disks. Add top/bottom margins and/or page numbers to your hard copy. Rename/Redate disks. Use special printer codes eg: LPRINT CHR$(3); toggles printer output to the ROUTE device. Special keyboard codes add even more versatility. This upgrade improves date file stamping MM/DD/YY instead of just MM/YY. Adds optional verify on/off formatting, enables users to examine *01-*15, DIR, and BOOT sectors using DEBUG, and corrects all known TRSDOS 1.3. DOS errors. Upgrade includes LIBDVR, a /CMD driver that enables LIBRARY commands, such as DIR, COPY, DEBUG, FREE, PURGE, or even small /CMD programs to be used within a running Basic program, without variable or data loss.

SYSTEM 1.5. is now distributed exclusively by TRSTimes magazine.
ORDER YOUR COPY TODAY!

NEW LOW PRICE $15.95

TRSTimes - SYSTEM 1.5.
5721 Topanga Canyon Blvd., Suite 4
Woodland Hills, CA. 91367

BEAT THE GAME

by Daniel Myers

THE ENCHANTER

Welcome to the wonderful world of Enchanter! Before we start, a few words about the game: first, it helps to make a map as you go along. You will not be visiting all the locations, and if you make a misstep somewhere, having a map will save you a lot of time and trouble. Second, you will need to have food and water with you most of the time. Make sure you eat and drink when the program warns you that you are hungry and thirsty. Also, keep in mind that, if your water supply runs out, you can always get more, but once the bread is gone, you won't be able to obtain more food. So, don't go too far astray, or you might starve to death! Finally, remember to save the game every once in awhile, especially before doing anything dangerous!

Ok, the game begins with your being summoned by Belboz to a council of the Circle of Enchanters. You are told that you must put an end to Krill, a nasty and powerful Wizard, after which, with your trusty spell book, you are sent into the game proper, where you find yourself at a fork in the road. From there, go NE, then North, and you will be in a shack. Get the jug and the lantern, then open the oven door and get the bread. You have food, now you need some water.

Go South from the shack, then NE, SE, NE to the Shady Brook. Here you fill your jug with water. Head SW SE to another fork in the road, and from there SW SW to a deserted village. Well, almost deserted. There's one place that seems to be inhabited. You head South, and run into an old crone who hands you a spell scroll, and pushes you back out the door again. This scroll has the REZROV spell. Use the GNUSTO spell to write it into your spell book, then learn the spell. Now, go NE NE, and you're back at the fork. From there, head along East until you come to the outer gate of Krill's castle. REZROV the gate, then continue East to the inside gate.

Learn the FROTZ, NITFOL, and REZROV spells. FROTZ the lantern, then go North twice to the tower, and then Up the tower steps to the Jewel Room. There is an egg here, with all manner of little switches and doodads on it. You could actually open the egg by figuring out the proper sequence, but that isn't necessary. REZROV the egg, and it will open, to reveal a shredded scroll. There is no way to avoid this, even if you opened it by hand. Don't worry about that, however. Just get the scroll, and drop the egg. Learn REZROV once more (very handy, that spell!), then go Down to the Tower, and from there due East through the four Mirror Rooms to the North Gate. REZROV the gate.

Now move out North through the gate to the Woods. Here you find another spell scroll. This is the KREBF spell, which will repair the shredded scroll. You will only need this spell once, so you don't really have to GNUSTO it. In any case, cast the KREBF spell on the shredded scroll, which will be restored and usable. The spell on this scroll is ZIFMIA. GNUSTO that one, then walk East to the Swamp. NITFOL the frogs, who will tell you how to get the CLEESH spell. GNUSTO that one, too.

Now, return to the North Gate, and from there go back West through the Mirror Rooms to the Tower, and then from there due South until you come to the SouthWest Tower. Go East to the South Hall, then South to dungeon. Open the cell door and enter the cell. Examine the walls. Aha! A loose block! Move the block, and you will be able to move East into a secret room. There is another spell scroll here, the EXEX spell. Get the scroll and GNUSTO the spell. You will also find a silver spoon here, but that item is just "window dressing"; it has no useful purpose in the game, you should leave it behind. Now go back West, then South and Up to the South Hall again.

Drop everything you have, and go East into the Gallery. In the dark, you will see that one portrait is lit up. Move that, and you will find a black candle and a black scroll. The scroll holds the OZMOO

spell. Get these items and return West. Pick up your supplies (you won't need the lantern if you take the candle), then GNUSTO the OZMOO spell. About now, you're probably feeling a bit tired. Go West to the Tower, and Up the stairs. Well, look at that! A comfy featherbed. Get into bed and drift off to sleep.

While you sleep, you have a dream. The dream is an indication of the location of another scroll. When morning comes, get up, then examine the bedpost. Aha! A hidden switch! Press it, and a compartment will open up, revealing the VAXUM spell. Get that scroll and GNUSTO the spell. It will soon be time for you to get yourself killed (among other things), so learn the OZMOO, NITFOL, and EXEX spells.

Go down the stairs, the head East until you come to the South Gate. Go South from there to the meadow, then SE to the Shore. Here you will see a giant turtle with a rainbow-colored shell. Cast NITFOL on the turtle, then tell him to follow you. Return to the South Gate and go East from there to the base of the SouthEast Tower. Go up the stairs, and you will be in the Engine Room, which is full of all sorts of dangerous and incomprehensible machinery. Cast EXEX spell on the turtle, then tell him to go SE and get the scroll.

The speed spell will make him fast enough to dodge safely through that room into the Control Room, where the Kulcad spell scroll is. On his way back, he'll set off a trap, but his heavy shell will protect him. You couldn't have managed it, because you have no protection from the sharp spears. The turtle will give you the scroll, then return to the beach. The Kulcad spell is too powerful for you to GNUSTO, so you'll have to just hold on to the scroll until you need it. And now, it's time for you to die.

Go down the stairs, then West to the Hall and North to the Closet. Pick up the Jeweled Box and continue North to the Courtyard. Don't bother trying REZROV on the Box; even that spell isn't powerful enough to open it. Just go East to the front of the temple, drop everything you have, then go East once more. You will be captured and put in a cell to await a sacrificial ceremony, at which you will be the guest of honor.

Now, OZMOO yourself, and Wait. The creatures will soon come for you, and you will be offered up on an altar and a knife plunged into your heart. Because of the OZMOO spell, you won't really be dead. However, you now have the means of opening the Jeweled Box. Once you are on your feet again, step down from the altar, amd go East back to the courtyard. Cut the rope, then open the box and get the MELBOR scroll. Pick up the rest of your possesions, and GNUSTO the MELBOR spell.

Now, learn MELBOR, VAXUM and ZIFMIA, then head West, West to the Inside Gate, and from there to the Mirror Rooms. Here you must wait until you see the Adventurer on the other side of the mirror. At that point, ZIFMIA Adventurer, and he will appear before you, a little bit upset. Since you have a move to spare here, MELBOR yourself, then VAXUM the Adventurer, who will now be very friendly towards you. He will also be looking at your inventory with covetous eyes.

As soon as he's been VAXUM'd, head directly East until you come to the Guarded Room. Don't worry, your new friend will follow you along. Once at the door (and you should carefully read the description of it; it's really amusing), tell the Adventurer to open it. He will do so, and the illusions of monsters will disappear, revealing only a plain wooden door. Go North through the door into the Map Room.

Here is one of those variable things in the game. There are several objects in this room, two of which, the map and the pencil, are crucial to your success. Sometimes, the Adventurer will pick up one or both of these items. You must get them back from him before he leaves, or you may never catch up to him again, in which case the game is lost. If the Adventurer picks up something you need, tell him to give it to you, and he will. In any case, you should drop the dagger now because you don't need it anymore. You also won't need the purple scroll with the FILFRE spell. Make sure you have the map and the pencil, then go back to the North Gate, and from there South to the Library.

Examine the ashes on the floor, then the tracks in the ashes. These will lead you to a mousehole in the wall. Reach inside, and you will find the scroll with the GONDAR spell. GNUSTO that one. While you're poking about here, you might hear gutteral voices coming towards you. Don't worry: the MELBOR spell will keep you protected from any of the hairy creatures that might enter the room.

There is also a dusty old book here that you might want to read, as it will help you to understand what you're doing next. From the Library, return to the South Hall, then go down into the dungeon, and down once more to the first Translucent Room. You will probably be tired now, so just go to sleep right where you are; nothing will hurt you. You only had to sleep in the bed to have the dream to find the VAXUM spell.

When you wake up, eat and drink if necessary, then drop your spell book and the jug. Look at the map, and you will see some lettered points connected by lines. This is a magic map of the area where you are now. If you connect two adjacent points with the pencil, an opening will actually appear between those two rooms. Likewise, if you erase a line between two points, then you close off the opening between the two rooms. However, you can only use the eraser twice and the point twice before each becomes useless. Now that you know what you have, let's put it to some good use.

You are standing at the moment at point B on the map. From there, move South, East, NE, SE, and you will be at point F. The point all by itself, P, is where the Unseen Terror currently resides, and you are about to free it. Draw a line from P to F. You will see the opening appear in the wall before your eyes, and then a very scared Belboz will appear briefly with a warning. Now, move SW twice (the first time you won't get anywhere) and you will be at point P, where the GUNCHO spell is.

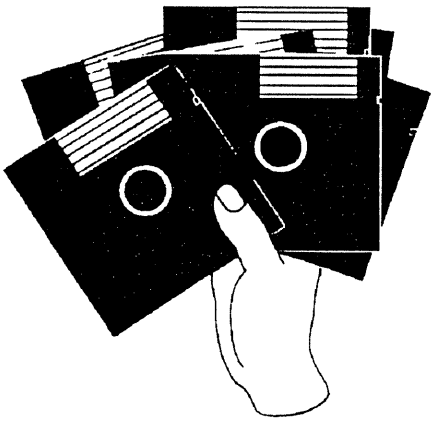
Now, erase the line from B to R, which will keep the Terror from escaping. Also erase the line from M to V, which traps him in the rooms again. Pick up the GUNCHO scroll, and make your way to point J. Draw a line from J to B, then walk West to B and get your spell book. Learn the CLEESH, GONDAR, and MELBOR spells. The GUNCHO spell is too strong to be written in your book, so you'll have to carry the scroll with you.

Now, go Up twice to the South Hall. MELBOR yourself, then go West to the South Gate, and from there due North to the Junction. At that point, head East twice to the Winding Stairs. This is another powerful illusion; no matter how much you walk up or down, you will never get anywhere. KULCAD the stairs, and they will disappear, leaving you over a Bottomless Pit! Fortunately, the bannister turned into a vellum scroll containing the IZYUK spell. IZYUK yoursel, and fly East into (ta-da!) The Warlock's Tower!

Here, at last, you come face-to-face with Krill himself. However, before you can take care of him, you will have to get rid of a couple of his friends. When the dragon attacks, GONDAR the dragon, and when the being attacks, CLEESH him. Now, you're ready for the main event. As Krill begins his chant, GUNCHO him. He is banished forever from this plane of existence, and you have become a member of the Circle Of Enchanters!!

C Language Tutorial, Part II

by J.F.R. "Frank" Slinkman



First, the errata for Part I:

1) On Page 26, right column, toward the bottom, in first line of code inside the brackets in the get_val() function, reads:

```
printf( msg );
```

It should read:

```
printf( "%s", msg );
```

I DID test this code; so I don't know how the error crept in.

2) In my copy of TRSTimes 7.3, the sheet which has Page 27 on one side and Page 28 on the other was turned over, which put the pages in the order 26, 28, 27, 29. This made the article a little hard to follow.

OK. If you haven't fixed the error in prog03a.c yet, please do so now, and compile and run it.

You'll notice, as you input different values for rate, time and distance, that the program deals with integer values, as we specified when we declared the variables.

Thus, if you input a rate of 60 and a distance of 330, the program will report time as being 5, not the correct 5.5.

There are two ways to correct this. The first, and most obvious way, is to change to floating point math. The second way, which runs faster but at the sacrifice of a great deal of precision, is to invent our own kind of fixed point math.

So let's create prog03b.c, by modifying prog03a.c to work with double precision floating point math. To do this, perform the following editing steps:

1. Change the name to prog03b.c

2. After the line "#include <stdio.h>," add the line:

```
#include <math.h>
```

3. In the main() function, change the line

```
int     rate = 0, time = 0, distance = 0;
```

to

```
double  rate=0.0, time=0,0, distance=0.0;
```

4. In the main() function, each of the three times get_val() is called, insert "(double)" immediately before "get_val." For example, change

```
rate = get_val( "Input rate: " );
```

to

```
rate = (double)get_val( "Input rate: " );
```

5. In the main() function, change all the "%d" codes in the printf() control string to "%f".

The reason for change (1) is obvious.

Change (2) causes the compiler to include the file MATH/H as part of your program which, among other things, tells the compiler to include the double precision math functions contained in MATH/REL when needed. This line should appear in any program which uses double precision math.

Change (3) changes the three variables from type int to type double. Note the use of "0.0" instead of just "0" to initialize these variables to zero. Floating point variables, when being assigned specific numeric values as we have done here,

should always have the value in this kind of notation. In other words, there should be a decimal point and something to the right of the decimal.

This convention exists for several of reasons. It is more efficient because the compiler doesn't have to take the integer zero and convert it to double precision form before loading it into the variable; some compilers might try to load the 2-byte integer value of zero into the 8-byte field where the double variable is stored; and it helps those reading program listings stay mindful of what kind of data is being used.

This convention is somewhat similar to using the pound sign in BASIC to ensure the number is stored in full double precision, as in:

```
50 X# = 1 / 3#
```

which produces a different result than

```
50 X# = 1 / 3
```

Change (4) introduces the concept of "casting." The value returned by get_val() will still be a short signed integer, just as in prog03a.c. The addition of "(double)" causes the compiler to take this value and cast (convert) it to double precision form before storing it. This example is somewhat similar to the CDBL command in BASIC. Of course, you can cast any type of numeric value to any other type, just by prefixing the desired type name (e.g., int, float, unsigned long, etc.), in parentheses immediately before the value.

Change (5) is done to tell the printf() function that the values (rate, time, and distance) being passed to it are doubles. Had we not made this change (i.e., had we kept the "%d" codes), printf() would only have looked at the first two bytes of the 8-byte values, assumed they were signed ints, and displayed erroneous values.

Now, when you compile and run prog03b.c, the results of the calculations will be displayed with accuracy to six decimal places. There are ways to change the number of digits displayed, reserve space for the minus sign, force the use of either a plus or minus sign, etc.

For example, if we had used "% 3.3f" instead of "%f," we would have reserved space for a (possible) minus sign and three positions for each the integer portion of the number and the fractional portion. I refer you to the documentation for fprintf() for a full discussion on print formatting.

Note, however, that with prog03b.c you can still only input integer values from the keyboard when prompted for rate, time and/or distance. Only the calculations are done in double precision.

OK. Before you read any further, create prog03b.c, save it, compile and run it. Note the differences in the way it behaves from the previous version.

Now, to permit the input of real numbers, let's edit prog03b.c to prog03c.c, as follows:

1. change the name to prog03c.c

2. After the line "void clr_scr();" add the line

```
double  get_val();
```

3. In main(), remove the casting of get_val() to double all three places it was done previously.

4. In get_val(), change the line

```
int get_val( msg )
```

to

```
double get_val( msg )
```

5. In get_val(), change the line

```
return NULL;
```

to

```
return 0.0;
```

6. In get_val(), change the line

```
return atoi( inbuf );
```

to

```
return atod( inbuf );
```

Change (2) adds a forward declaration of the get_val() function to tell the compiler it now returns a double precision floating point value.

Change (3) reflects the fact that get_val() now returns a double precision value (it's unnecessary to cast a double to a double).

Change (4) declares the function get_val() will return a double result.

Change (5) causes the return of the value zero in double precision form, rather than integer form.

Change (6) invokes the atod() function, which returns a double precision value, instead of the previous atoi(), which returns an integer.

Now, when you compile and run prog03c.c, you'll find it possible to enter real numbers instead of just integers.

I didn't state it before, because I wanted you to get your feet wet first, but there is a very important and fundamental thing to learn before we go any further.

Look at the line "rate = get_val("Input rate: ");" in the main() function.

The expression "get_val(...)" represents a VALUE! This value can be used ANYWHERE in a program where either a number or a variable can be used.

What that value is, of course, depends on the code in get_val(), but it is still a value, just like 6.325 or -0.533 are values.

Consider the line "rate = 60.0;". This will cause the compiler to get the double precision form of the value 60 and store it at the RAM address assigned to hold the value for the variable "rate."

The line "rate = get_val(...)" does EXACTLY the same thing. It gets the double precision value returned by the function get_val(), and stores it at the RAM address assigned to hold "rate."

Functions in C are roughly equivalent to DEF FN in BASIC, but are much more powerful and versatile.
It's impossible, for example, write a DEF FN line in BASIC to do everything get_val() does.

This is a small example of the additional power, flexibility and efficiency C has over BASIC.

O.K. I guess we're all tired of the rate, time and distance problem, so let's move to another area.

Using your test editor, enter this program, and save it out as PROG04/CCC:

```
/* prog04.c */

#include<stdio.h>

void   get_str(), count_chars();

char   inbuf[81];

main()
{
  get_str();
  count_chars();
}

void get_str()
{
  puts( "Input a string now\n" );

  for ( ; ; )
  {  gets( inbuf );
     if ( strlen( inbuf ) )
        break;
  }
}

void count_chars()
{
  int    i, c, length;
  int    alpha, numeric, space, punct;

  alpha = numeric = space = punct = 0;
  length = strlen( inbuf );

  for ( i = 0; i < length; i++ )
  {  c = inbuf[i];
     if ( isalpha( c ) )
        alpha++;
     else if ( isdigit( c ) )
        numeric++;
     else if ( isspace( c ) )
        space++;
     else if ( ispunct( c ) )
        punct++;
  }

  printf( "\nThere are %d letters\n", alpha );
  printf( "There are %d numbers\n", numeric );
  printf( "There are %d spaces\n", space );
  printf("There are %d punctuation
marks\n",punct);
}
```

This program is structured more like a typical C program than the previous examples, in that the main() function does little or nothing more than call other functions which do the actual work.

You've seen the forward declaration of void functions before, and the use of a global character buffer; so let's go right to the get_str() function.

First, it calls puts() to tell the user to enter a string from the keyboard. If you'll remember, puts()

adds a newline character to the string it displays. Thus, the newline character ("\n") at the end of this string will cause an extra, blank line to be printed.

Now we see the use of the simplest form of the "for" statement, which is the C equivalent of BASIC's FOR-NEXT-STEP construction.

The general format of the "for" statement is:

```
for ( initialize; condition; step )
   program_statement;
```

The logic, in pseudo-code, is as follows:

1. execute "initialize"
2. is "condition" TRUE?
   YES:
      A. execute "program_statement"
      B. execute "step"
      C. goto 2
   NO:
      exit

Note that because "condition" is evaluated before "program_statement" is executed, "program_statement" would never be executed if "condition" evaluates to FALSE the first time through the loop.

In this "for" loop, notice there is no starting point, no ending point, and no step. In other words, "for(;;)" creates an endless loop which can only be exited via a "break" statement, which will cause immediate exit from any "do," "while" or "for" loop.

These two semicolons by themselves are examples of the "null" statement. The "null" statement is just a place-holder to meet a language requirement for a statement or expression when no statement or expression is desired or needed.

Because both the initialization and limiting statements are required for a "for" loop, we have satisfied that requirement by using two null statements, which do nothing.

There can only be one program statement in a "for" loop. In this case we are using a "compound statement" made up of two statements combined into one through the use of brackets. You may combine as many statements into a compound statement as you wish.

The first of these two, "gets(inbuf);" gets up to 80 characters of keyboard data, and stores it in the character array, inbuf.

Next, the standard library strlen() function is called to get the length of the string which was input. This line:

```
if ( strlen( inbuf ) ) break;
```

checks the string length for a non-zero (TRUE) value, and if TRUE, exits the endless for loop via the "break" statement. However, if the user merely hit the ENTER or BREAK key without entering any text, the statement would evaluate FALSE (i.e., a string length of zero or an error return code of zero), and gets() would be called again.

What this code does is refuse to accept a situation where nothing is input. The only way this endless "for" loop can be exited is for the user to type in something before hitting ENTER.

Now we need to look at the count_chars() function. Here, we declare a total of seven variables. Note that many variables can be declared on one line.

The next line initializes "alpha," "numeric," "space," and "punct" all to zero. This kind of multiple initialization is perfectly legal, but has gone out of fashion because some people feel it makes program listings harder to read.

It is, however, far more efficient than initializing each variable in a separate line. Personally, I still use this method, except when I'm submitting code to others who might look down their noses at it.

The next line calls the strlen() function to get the length of the string which has been stored in "inbuf" by the gets() function, and stores this value in the variable "length."

Now we come to our first REAL "for" loop. It has all three of the possible parts, namely the initialization of variable(s) before the first semi-colon; the condition which must be met for program control to stay within the loop before the second semi-colon; and the step expression after the second semi-colon.

This line, "for (i = 0; i < length; i++)" is identical to the BASIC line:

```
50 FOR I = 0 TO LENGTH-1 STEP 1
```

Note the "++" operator. This increments a value by one. There is also a decrement ("--") operator which decrements a value by one.

The "i++" is identical to the BASIC, "i = i + 1."

Either operator ("++" or "--") can be placed either before or after the affected variable. If it's placed before the variable, it increments or decrements the variable before using it. If placed after the variable, it changes the variable after using it.

Also note that the first part of this "for" statement could have been written:

```
for ( i = 0; i < strlen( inbuf ); i++ )
```

After all, the length of the string won't change; so "length" will say equal to "strlen(inbuf)."

Obviously, both ways will work. But using the "length" variable is faster because the string length only has to be obtained once, instead of each and every iteration of the loop. It's a LOT faster to pick up an integer variable than it is to obtain the length of a string.

The first of the statements included in the compound statement in this for loop gets a single character from the "inbuf" array, and puts it in the variable "c."

Again, the use of an integer variable ("c") is done for speed, since it's faster to pick up an int off the stack than it is to get a value from an array.

In other words, all the subsequent tests of the variable "c" could also be done for the array element "inbuf[i]." It would just take longer.

In C, all arrays are "base zero." That is, the first array element is element number zero, and the last element of an 81-member array is element number 80.

This is why, in the for statement, we initialized the value of "i" to zero, and used the expression "i < length" -- because if the string is 15 characters long, the last character will be in array element number 14, not element 15.

The rest of the code in the loop puts the character value in "c" to various tests, to determine if it's a letter, a number, a white-space character or a punctuation mark.

Suppose the character is a letter. In that case, the standard library function isalpha() will return TRUE; so the statement "alpha++" will be executed.

In this way, all the letters will be counted, and the total number of letters will be stored in the variable "alpha." Likewise, the number of numeric characters will be stored in "numeric," etc.

After the loop is exited, a report will be displayed on the screen, giving a breakdown of the counts of the various kinds of characters found in the string which was input.

The screen dialogue produced by this program will look something like:

```
Input a string now

Billy, the fat boy, ate 17 of the 21 pies his
mother, Anne, baked on May 5th.

There are 50 letters
There are 5 numbers
There are 16 spaces
There are 5 punctuation marks
```

Compile and run prog04.c now, and try it out with different keyboard data. Observe how fast it counts the various types of characters and reports them. And think how slow the same process would run in BASIC, and how much more work it would take to write the same program in assembler.

Save PROG04/CCC for next time, when you'll learn how to use pointers and the automatic scaling of arrays to do the job even faster.

MEET MY NEW TOY, A UPS!

by Roy T.Beck

My mother is now 88 years of age, and still insisting she wants to live alone, and not in a nursing home or the like. For over a year now, I have been making regular monthly trips to visit her, look after her finances, take her to the doctor, etc. Since she lives 500 miles away, it is a significant trip, and requires my wife and I to be away from here 5 days at a stretch.

I also operate and maintain a BBS, the TR-Surtrove at 213-664-5056, 8-N-1. While the board has been fairly reliable, there have been outages due to power system glitches, drunken drivers hitting poles, etc. When this happens, I can expect to arrive home to find the BBS machine displaying DOS Ready, or worse, on its screen. This annoys me, and of course I know some of you have been unable to access the BBS when you tried.

I finally decided to improve the situation. Recently, I bought a UPS machine to support two of my computers which are always on line.

But first, let me discourse a bit on what constitutes a UPS. A UPS, strictly speaking, is an Uninterruptible Power Supply, hence the acronym. This means a power supply which is theoretically, never, under any circumstances, interrupted, and therefore any device served by it is also never subject to power supply interruptions. As I said, "theoretically" The real world is never so obliging, but actual UPS machines do come close to the ideal, given some maintenance and care.

Practical UPS machines usually consist of a "black box" connected between the commercial power system and the load to be supported. The black box contains an AC to DC rectifier, a storage battery and a DC to AC Inverter. The rectifier serves two functions simultaneously. It serves to keep the battery fully charged, and simultaneously feeds sufficient DC power to the inverter to operate it and through it, serve the connected load to be supported, such as your computer, burglar alarm system, etc.

Some of the important characteristics of a UPS are worth considering. In the event of failure of the commercial power system, the UPS should maintain an uninterrupted flow of power to your load. But, this is expensive, although it can be and is done in many applications. However, there are ways to cheapen the cost and performance of UPS's if you are willing and able to accept somewhat less than perfect power to your load to be supported. The UPS I bought is in this latter category. It is a standby type UPS. By reducing the size of the rectifier to serve only the need to recharge the battery, and by installing a double throw relay, my unit is designed to normally bypass commercial power around the rectifier/battery/in-verter scheme and feed my computers via the relay. In the event of failure of the commercial power, some electronic monitoring circuitry will detect loss of the utility power and shift the relay in 3 milliseconds. This connects the output of the inverter to my computers, and immediately begins discharging the storage battery, but meanwhile supporting my machines. This scheme is acceptable, because 3 milliseconds represents only a fraction of a 60 hertz cycle. 60 hertz means 60 cycles per second, or 16.67 milliseconds per cycle. 3 milliseconds thus represents 18% of a cycle, and most devices can tolerate this much of an outage without distress.

The amount of load which can be carried by a UPS is determined by two things. The battery must be large enough to carry the current without allowing the terminal voltage to sag below the minimum permissible value to the inverter. The inverter itself must be large enough to carry the required volt-amperes taken by the load. This product of volts and amperes, usually abbreviated VA, is not the same as real power, which is the product of volts, amperes, and power factor. Since the power factor of typical computer power supplies is much less than unity, the real power rating of a UPS when serving typical computer equipment is around 60 to 70% of the VA rating. In the case of my unit, it is rated 600 VA and 400 watts. The controlling quantity, as far as the inverter is concerned, is the VA rating.

The second major rating of a UPS is the time duration during which it can carry the desired load. In my case, the machine can carry the 400 watts of load for 5 minutes, which provides support during brief outages of the power system. As the load to be supported is reduced, the time duration increases, approximately inversely. Obviously 5 minutes won't cover my absence if I am out of town, or simply away from home. However, I feel this will be adequate for most realistic problems here at my house, as power outages exceeding even a second or two are extremely rare. I have lived here some 26 years, and I am sure I could count on the fingers of one hand the

times our power was interrupted more than a minute. The most recent was the January '94 earthquake, when power was out about 3 hours.

In commercial applications, the design operating time duration of a UPS is typically 30 minutes. In this time, the UPS owner expects to start up a diesel-electric generator and transfer his critical load to the on-site generator. thus avoiding outages to his critical loads. The time limit is thereby extended to the time his diesel fuel tank can supply. Since is is quite economical to install a large oil tank, the owner can easily provide fuel for several days if he wishes.

An important feature of any UPS is that the inverter should run in synchronism with the commercial power system. In my standby type unit, the inverter is always running, but only carries load if the transfer relay requires it. By maintaining the inverter in synchronism with the commercial system, there is no perceptible "bump" due to phase shift when the transfer relay operates. The same is true when the commercial power returns; the inverter is resynchronized with the commercial system before the transfer relay shifts the load back from the inverter to the commercial system. In the case of the best UPS systems, the inverter is always carrying the load, whether the commercial power is present or not, so there is no interruption when the commercial power returns. Since the better UPS units have a static transfer switch to allow the load to be shifted directly to the commercial system when maintenance must be done, there is still a requirement that the inverter must run in synchronism with the line.

Another characteristic which must be looked at is the harmonic content of the inverter output voltage. The cheapest inverter produces only a square wave of output voltage, which inherently contains a lot of harmonics. Since some loads object seriously to harmonic content, you need to consider this fact. Motors, especially, are sensitive to harmonics in the voltage feeding them, and respond by overheating when high harmonic content is present. While most computer fan motors will tolerate harmonics, they will have a shorter life due to the resultant overheating. UPS machines can be designed with internal filters to limit their harmonic content, but this does raise their cost. Filters, when installed, are usually designed to limit total harmonic distortion to 5%.

Batteries come in several varieties. The cheapest are lead-acid batteries. For UPS units designed for installation in occupied spaces, the usual choice is the sealed cell, maintenance free type, available from several manufacturers. These can be expected to have about a 5 year life, even if not called upon to carry the inverter load. If the UPS is required to carry load for significant lengths of time, the batteries will be subject to significant discharge/recharge cycles. This process will also shorten the battery life.

A typical time to recharge the battery in a UPS is 10 hours, known as the C-10 rate. This means the battery's capacity is esentially fully recharged in about 10 hours. Actually, batteries recharge in a tapering fashion, and require 2 or 3 days to truly recharge all the way, but in 10 hours you will get approximately a 90% recharge. Pushing more current into the battery won't solve this problem; batteries simply won't fully recharge in less than several days, but the last 10% is not so critical as the first 90%!

Another concern about UPS units is that they are usually current limited in their output capacity. This is because the solid state devices will not tolerate large overcurrents. They will simply disintegrate. Accordingly, there is usually protective circuitry which turns them off if the current exceeds the permitted value. As a user, this fact is significant to you if you wish to support any load which requires large amounts of inrush current at startup, such as electric motors. AC motors can easily draw up to 10 times rated current when starting. If the motor is small, such as a fan motor, the effect on the UPS may be negligible. Don't try to start a large motor on a UPS unless the system design took this into account.

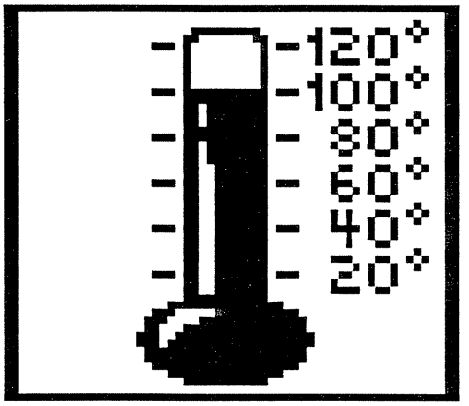
As a matter of interest, my UPS is of the standby type, is rated for 600 VA and 400 watts for 5 minutes, cost $250, weighs 23 pounds, and is 6.5 inches high, 4.75 inches wide, and 15.5 inches deep. I haven't opened it up, but I am sure it contains a sealed 12 volt lead-acid battery which accounts for most of its weight.

So far, I have only taken one out of town trip since installation of the UPS, but everything was normal when I returned home. I believe we will now have more dependable operation of the BBS, and of course, my other machine will benefit also, as I leave it on line to accept the odd FAX sent to me. That machine is also on the same phone number, 213-664-5056. I realize not everyone needs or even wants a UPS in their home installation, but I thought my setup might be of interest to all of you, hence this article. Having installed a number of these things in various refineries and other commercial installations, I finally have one I can call my own.

HOW HOT IS IT?

Model 4 - BASIC

by Lance Wolstrup



With the World Cup being held in the U.S., with several games currently being played in Los Angeles (the Rose Bowl in Pasadena), I have recently discovered just how many friends and relatives I have in Europe. Yes indeed! Much to my wife's chagrin, my house has become the equivalent of the Woodland Hills Motel 6 for die-hard soccer fans.

This state of affairs started some months ago with phone calls from distant cousins and people I had met once in a Norwegian car wash in 1971, I think! After the pitch for a place to stay, somehow the conversations always turned to the weather. "How hot is it in June", they asked. The answer is "Hot, Hot, Hot". Southern California is blessed with warm winters, but cursed with hot summers. We have day after day of temperatures in the high nineties, occasionally dipping above the hundred mark. Europeans don't relate well to Fahrenheit, so the next question was inevitably, "What is that in Centigrades?" I had to admit that I didn't know. After a few of these conversations, I sat down in front of my Model 4 and wrote the first version of TEMPCONV/BAS.

The initial version was just a few lines of code to perform the conversion from Fahrenheit to Celsius and back. Roy Beck looked at the program and suggested that I expand it to include conversions to Kelvin and Rankine. Though I had heard of Kelvin, I really didn't know 'who, why or what'. I knew even less about Rankine. Great way to program, ain't it!!!

However, Roy provided me with the conversion formulae needed for the program, and he was also kind enough to write a short history of the four temperature scales. His essay can be found immediately following the program listing.

TEMPCONV/BAS, while expanded to include Kelvin and Rankine, is still a short program. It is only 31 lines in length, but, I hope you'll agree, it still has some fancy features.

When the program is RUN, the screen displays the names of the four temperature scales: Celsius, Fahrenheit, Kelvin, and Rankine. The cursor is blinking next to Celsius. This indicates that you may now type a temperature to be converted. You may use the <up-arrow> and <down-arrow> keys to move the cursor to the temperature scale you wish to be the base scale. Typing a number there will convert it to the other three scales. Pressing the Esc sequence (<shift-up-arrow>) will erase the screen and exit the program back to Basic.

You may convert positive or negative temperatures, using up to 4 digits - the limits being 9999 to -999. The program itself is reasonably simple. Line 10 - 15 sets up the initial data by storing the screen width in variable SW, reading the names of the temperature scales into the array NM$(X) and then jumping over the subroutines to the beginning of the actual program in line 100.

Lines 20 - 23 appear in almost all programs that I write. It is the print@ routine to print flush left (20), centered (21), flush right (22) or anywhere at all on the screen (23).

Lines 30 - 39 contain my standard input routine, modified to accept a minus only as the first character

Line 100 erases the screen and displays the program name and the copyright.

Line 110 displays the names of the temperature scales. Line 120 initializes variable V with 8. This is the vertical position of the cursor when we enter line 130, which sets the maximum length of the user input to 4, and then enters the input subroutine in

line 30. Returning from the subroutine we check variable FL. If it is set to 1 it means that the user pressed the escape key (<shift up-arrow>) and we then abort the program. If, on the other hand, only <ENTER> was pressed, variable IN$ will equal "" and we go back to line 120 and start the input over again. If neither of these conditions apply, we have a valid number and we can go on to line 140.

Line 140 determines on which line the input was made. For example. the Celsius input takes place on screen line 8 - thus V=8. The Fahrenheit input occurs on screen line 9 - thus V=9, etc. Consequently, by using the ON V-7 GOTO..... we are sending the program to the line that handles the proper temperature conversion.

I chose to use Celsius as the common denominator, so line 150 simply converts the value from IN$ to numeric array variable TM(1). This variable holds the Celsius temperature. Line 160 converts the Fahrenheit input to Celsius and stores it in TM(1). Line 170 converts the Kelvin input to Celsius and stores it in TM(1). Line 180 converts the Rankine input to Fahrenheit, and then from Fahrenheit to Celsius.

Line 200 is the line common to lines 150, 160, 170 and 180. It converts the Celsius temperature to the other scales. TM(2) is Fahrenheit. TM(3) is Kelvin, and TM(4) is Rankine.

Line 210 displays the temperature conversions with one decimal point. This is needed to be as accurate as possible.

Lines 220 - 240 ask if another conversion is desired, and takes the appropriate action as instructed.

```
0 'tempconv/bas
1 '
10 SW=80
14 FOR X=1 TO 4:READ NM$(X):NEXT
15 GOTO 100
16 DATA Celsius,Fahrenheit,Kelvin,Rankine
20 H=0:GOTO 23
21 H=INT((SW-LEN(A$))/2):GOTO 23
22 H=SW-LEN(A$)
23 PRINT@SW*V+H.A$;:RETURN
30 IN$="":FL=0:L=0:H=45:
A$=STRING$(8,32):GOSUB 23:A$="":GOSUB 23
31 I$=INKEY$:IF I$="" THEN 31
32 IF I$=CHR$(11) AND V=8 THEN V=11:
GOTO 30
ELSE IF I$=CHR$(11) THEN V=V-1:GOTO 30
33 IF I$=CHR$(10) AND V=11 THEN V=8:
GOTO 30
ELSE IF I$=CHR$(10) THEN V=V+1:GOTO 30
34 IF I$=CHR$(13) THEN 39
ELSE IF I$=CHR$(27) THEN FL=1:GOTO 39
35 IF I$=CHR$(8) AND L=0 THEN 31
ELSE IF I$=CHR$(8) THEN H=H-1:A$=CHR$(32):
GOSUB 23:A$="":GOSUB 23:L=L-1:
IN$=LEFT$(IN$,L):GOTO 31
36 IF L=0 AND I$=CHR$(45) THEN 37
ELSE IF I$<CHR$(48) OR I$>CHR$(57) OR L=ML
THEN 31
37 L=L+1:A$=I$:GOSUB 23:H=H+1:IN$=IN$+I$:
GOTO 31
39 RETURN
100 CLS:V=0:
A$="TEMPERATURE CONVERSIONS":
GOSUB 21:V=V+1:
A$="Copyright (c) 1994 by Lance Wolstrup - all
rights reserved":GOSUB 21:
V=V+1:A$=STRING$(SW,140):GOSUB 20
110 V=8:H=30:FOR X=1 TO 4:A$=NM$(X):
GOSUB 23:V=V+1:NEXT
120 V=8
130 ML=4:GOSUB 30:
IF FL=1 THEN CLS:END
ELSE IF IN$="" THEN 120
140 ON V-7 GOTO 150,160,170,180
150 TM(1)=VAL(IN$):GOTO 200
160 TM(1)=(VAL(IN$)-32)*5/9:GOTO 200
170 TM(1)=VAL(IN$)-273.11:GOTO 200
180 TM(1)=VAL(IN$)-459.6:TM(1)=(TM(1)-32)*5/9
200 TM(2)=TM(1)*9/5+32:TM(3)=TM(1)+273.11:
TM(4)=TM(2)+459.6
210 V=8:H=45:FOR X=1 TO 4:
PRINT@SW*V+H,USING"######.#";TM(X);:
V=V+1:NEXT
220 V=15:
A$="Would you like another conversion (Y/N) "
+CHR$(14):GOSUB 21:H=H+LEN(A$)
230 I$=INKEY$:IF I$="" THEN 230
240 IF I$="Y" OR I$="y" THEN
A$=STRING$(SW,32):
GOSUB 20:GOTO 120
ELSE IF I$="N" OR I$="n" OR I$=CHR$(27) THEN
CLS:END
ELSE 220
```

THE TEMPERATURE SCALES

by Roy T. Beck

There are four different temperature scales in everyday use in the USA, known as Fahrenheit, Celsius, Kelvin and Rankine. The most common, of

course, is the Fahrenheit scale, (abbreviated F.), used for cooking and weather reporting. If no scale is specified, then Fahrenheit is usually meant. The other commonplace scale is Celsius, (abbreviated C.), used for scientific and technical measurements in the USA, and for most purposes throughout the rest of the world. Until some 20 years ago, the Celsius scale was called the Centigrade scale, same abbreviation, same meaning. The name change was to honor a scientist named Celsius.

The Fahrenheit scale was the earliest scale, developed by German physicist Gabriel Daniel Fahrenheit, 1686-1736. Its zero point was empirically established, the method being the lowest temperature that he could attain in his laboratory, using a mixture of salt and ice. The explanation of the choice of interval and the boiling point of water requires more space than I have here. The result was a water freezing point of 32 degrees and a boiling point of 212 degrees, both at standard atmospheric pressure.

The Centigrade scale was invented by Anders Celsius, 1701-1744, a third generation member of a swedish scientific family. He established the zero point at the freezing point of water and the 100 point at the boiling point of water, both at standard atmospheric pressure. Because the difference between these two points was 100 degrees, the degree itself was 1/100 of the range. The name Centigrade refers to centi which means 1/100. The scale was later renamed Celsius in his honor, keeping the letter C as the abbreviation.

Because the temperature difference from freezing to boiling was 180 degrees in the Fahrenheit scale and 100 in the Centigrade scale, then the ratio 100/180 expresses the size of a Fahrenheit degree compared to a Centigrade degree. Reducing the fraction, one reaches the ratio 5/9, which is embodied in the conversion formulas.

The other two scales are more rarely used, and then only in certain kinds of scientific work. These are the Rankine and Kelvin scales. They both have their zero points at a temperature known as Absolute Zero. This temperature is that temperature at which all molecular motion theoretically ceases, and below which there is no lower temperature possible. It is actually impossible to attain absolute zero in any laboratory setting, but by the expenditure of great ingenuity and mucho dinero, it is possible to approach to within a fraction of a degree of absolute zero.

Absolute zero has been found to be approximately 273.2 Kelvin or Celsius degrees below the zero point of the Celsius scale, and approximately 459.6 Fahrenheit or Rankine degrees below the zero point of the Fahrenheit scale. I don't have a handbook handy to give more exact values, and these values are the ones I remember from my physics class, lo these many years ago.

The Kelvin scale was named for Lord Kelvin, Sir Joseph John Thomson, 1856-1940, professor of experimental physics at Trinity College. He did much work in atomic and nuclear physics. The absolute zero point is of significance in thermal radiation analysis.

The Rankine scale was named for William John MacQuorn Rankine, a Scottish scientist, 1820-1872, He worked in a range of disciplines, including molecular physics. He wrote several treatises on steam engines. The concept of absolute zero is significant in the thermodynamic analysis of steam as used in engines.