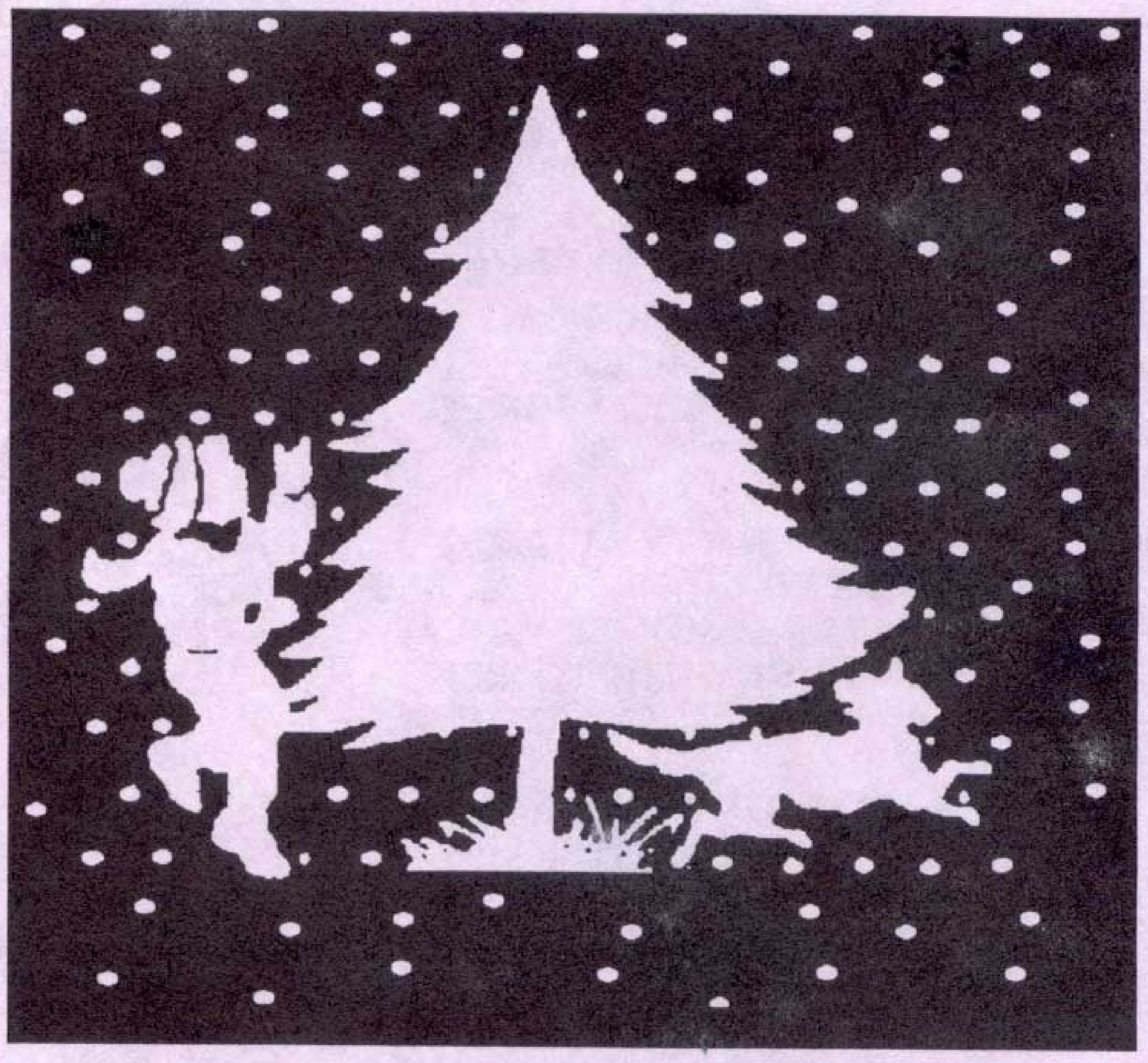
TRSTimes

Volume 7. No. 6 - Nov/Dec 1994 - $4.00



Happy Holidays 1994

DR. PATCH

UTILITY FOR TRS-80 MODEL 4 AND LS-DOS 6.3.1

A *'MUST HAVE'* FOR ALL LS-DOS 6.3.1 OWNERS.

DR. PATCH MODIFIES LS-DOS 6.3.1 TO DO THINGS THAT WERE NEVER BEFORE POSSIBLE.

COMPLETELY SELF-CONTAINED - MENU-DRIVEN FOR MAXIMUM USER CONVENIENCE.

FAST & SAFE - EACH MODIFICATION IS EASILY REVERSED TO NORMAL DOS OPERATION.

DISABLE PASSWORD CHECK IN FORMAT/CMD
FORMAT DOUBLE-SIDED AS DEFAULT
FORMAT 80 TRACKS AS DEFAULT
DISABLE VERIFY AFTER FORMAT
CHANGE 'DIR' TO 'D'
CHANGE 'CAT' TO 'C'
DIR/CAT WITH (I) PARAMETER AS DEFAULT
DIR/CAT WITH (S,I) PARAMETERS AS DEFAULT
CHANGE 'REMOVE' TO 'DEL'
CHANGE 'RENAME' TO 'REN'
CHANGE 'MEMORY' TO 'MEM'
CHANGE 'DEVICE' TO 'DEV'
DISABLE THE BOOT 'DATE' PROMPT
DISABLE THE BOOT 'TIME' PROMPT
DISABLE FILE PASSWORD PROTECTION
ENABLE EXTENDED ERROR MESSAGES

DISABLE PASSWORD CHECK IN BACKUP/CMD
BACKUP WITH (I) PARAMETER AS DEFAULT
BACKUP WITH VERIFY DISABLED
DISABLE BACKUP 'LIMIT' PROTECTION
DISABLE PASSWORD CHECK IN PURGE
PURGE WITH (I) PARAMETER AS DEFAULT
PURGE WITH (S,I) PARAMETERS AS DEFAULT
PURGE WITH (Q=N) PARAMETER AS DEFAULT
IMPLEMENT THE DOS 'KILL' COMMAND
CHANGE DOS PROMPT TO CUSTOM PROMPT
TURN 'AUTO BREAK DISABLE' OFF
TURN 'SYSGEN' MESSAGE OFF
BOOT WITH NON-BLINKING CURSOR
BOOT WITH CUSTOM CURSOR
BOOT WITH CLOCK ON
BOOT WITH FAST KEY-REPEAT

DR. PATCH IS THE ONLY PROGRAM OF ITS TYPE EVER WRITTEN FOR THE TRS-80 MODEL 4 AND LS-DOS 6.3.1.

DISTRIBUTED EXCLUSIVELY BY TRSTIMES MAGAZINE ON A STANDARD LS-DOS 6.3.1 DATA DISKETTE, ALONG WITH WRITTEN DOCUMENTATION.

DR. PATCH $14.95

NO SHIPPING & HANDLING TO U.S & CANADA. ELSEWHERE PLEASE ADD $4.00 (U.S CURRENCY ONLY, PLEASE)

TRSTimes magazine - dept. DP
5721 Topanga Canyon Blvd. #4
Woodland Hills, CA 91367

DON'T LET YOUR LS-DOS 6.3.1 BE WITHOUT IT!

TRSTimes magazine

Volume 7. No. 6 - Nov/Dec 1994 - $4.00

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.

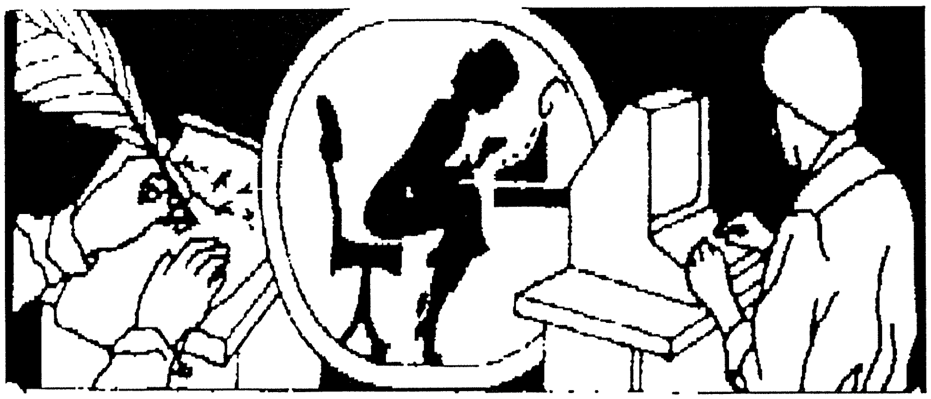
Publication months are January, March, May, July, September and November.

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.



LITTLE ORPHAN EIGHTY

The question we've been asked often the last couple of months is:

"Will TRSTimes continue in 1995?"

The answer is "Yes", we will do one more year of information and fun for our favorite machines. This will then mark our 8th year of publication - or just as long as 80 Micro lasted - who would have thunk it!!

Another question asked often is: "How is TRSTimes doing?" The honest answer is that we would have closed our doors long ago had it been a business. However, TRSTimes is not a business - it never was. Goodness knows that my family and I would have starved to death had we depended on it for a living. No, TRSTimes was always intended as a fun thing to do *after* I had earned my wages for the day - and I will certainly continue as long as it remains fun.

And fun it is when good people, such as Roy Beck, Chris Fara, Danny Myers, Kelly Bates, and Frank Slinkman, continue to favor the pages of TRSTimes with their knowledge. We also hope that you, the readers - old and new alike, will send in your questions, answers, frustrations, articles, programs, etc. so we can share them with the rest of the TRS-80 population.

Now for the bad news: The postal rates are scheduled to go up in January, so we are forced to follow suit — all in U.S. currency, the new rates for the 1995 TRSTimes subscriptions will be:

U.S.	$21.00
Canada	$22.00
Europe, Central/S. America	$26.00 surface
	$34.00 air mail
Asia, Australia, New Zealand	$28.00 surface
	$36.00 airmail

I trust you will stay with us for another year.

As of this writing, Los Angeles is getting ready for yet another of our infamous, high-publicity trials. The defense and prosecution are both seeking a jury of just the right ethnic and racial mix that will give their side an advantage. Prospective jurors must fill out questionnaires of close to a hundred pages so the attorneys can pick the ones they feel can be swayed. Justice has become a game, and we, the taxpayers, are paying an enormous price for exercises in political correctness. The jury-selection process will be lengthy and will cost us hundreds of thousands of dollars, or even more.

To dispense with this baloney, I propose we take the jury-selection away from the lawyers and put it where fairness and impartiality cannot be questioned.

That's right, gather the 200 prospective jurors in one room, let them pick their individual juror number (1-200) and then let's do the selection:

Set up a Model I and run the following Basic program:

```
10 RANDOM
20 DIM J(200)
30 FOR X=1 TO 200:J(X)=X:NEXT
40 FOR X=1 TO 12
50 Y=RND(200)
60 IF J(Y) THEN PRINT J(Y):J(Y)=0 ELSE 50
70 NEXT
```

In one short second we have selected a jury that is ready for ANY trial.

The cost? Well, I guess I could be persuaded to sell the turnkey system for somewhere around $50,000. A great savings for Los Angeles County, wouldn't you say!!!

TRSTimes will be closed for the second half of November to about the 10th of December. It's not that I am going on vacation - rather the unpleasant, but much needed earthquake repairs will finally be made. We've waited 10 months for this — and now I just want to get it over with. I don't know where we'll stay yet, but wherever it is, I'll be sure to bring a couple of computers with me.

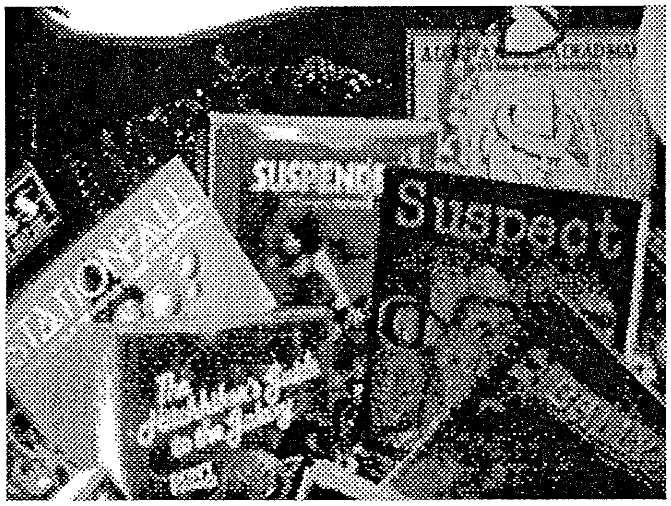
Before closing for this issue, I want to take the opportunity to wish everyone the very best for the holidays and the new year. Keep thinking TRS-80.

And now.....

Welcome to TRSTimes 7.6

BEAT THE GAME

by Daniel Myers



SUSPENDED

"Suspended" is an excellent game and possibly the most advanced game to date for the adventurer. The story is quite simple: You are the failsafe device to protect the surface world of Contra should any emergency develop that would cause the planet control devices to fail (these are underground in a complex where you live suspended in a cryogenic tube, awaiting a disaster). Of course, a disaster develops as soon as you boot up the disk.

You are awakened to find that there has been an earthquake that has damaged the cables in the Primary and Secondary Channels.

You have six different robots at your command. These robots all enjoy different skills and abilities. Each one represents a different sense.

Iris:	The sense of sight
Waldo:	The sense of touch and dexterity
Sensa:	Perceives things magnetic and electronic
Poet:	Perceives things electronic and can diag nose electrical flows
Whiz:	Commands the computer and can do errands
Auda:	The sense of hearing

The trick to winning "Suspended" is assigning the right task to the right robot. Also, the right robot has to be at the right place at the right time. This is called critical path planning, and is the secret of "Suspended."

In the standard game, there are a few real-time events to be aware of:

1) At the 15th cycle, there is another earthquake which causes an acid spill that kills, in short order, any robot that thereafter passes through the Cavernous Room (until the acid is shut off);

2) At the 75th cycle, there is another quake which wrecks the hydroponics and transit equipment on the surface above. These have to be fixed quickly or the game ends swiftly due to the starving populous above;

3) At the 100th cycle, humans enter the complex with the intent of turning you off because, by this time, you are clearly screwing up the assignment. These humans can be the death of you or they can help you by...well, I won't tell you yet.

The following commands get you through the game in less than 70 moves. As a result, they give little aid in the event that you finish the game in more than that time. The game changes considerably after the 75th move. There are many problems that surface after that time, and these clues do nothing to help you.

Let's get started!

#1 POET, GO TO WEATHER CONTROL
#2 SENSA, GO TO SUB SUPPLY ROOM
#3 WHIZ, GO TO SECONDARY CHANNEL
#4 SENSA, TAKE RAMP
#5 SENSA, GO WEST
#6 SENSA, TAKE CONTAINER AND GRASPER
#7 SENSA, GO TO HALLWAY JUNCTION
#8 WALDO, GO TO HALLWAY JUNCTION
#9 AUDA, GO TO GAMMA REPAIR

The above moves set the game up. Poet is needed to turn the weather control off; this minimizes deaths at the surface (your primary goal). Sensa gets the ramp that is needed to allow the robots to go from one level to another. Auda is sent to the Gamma Repair area because she will be needed later and, without the humans coming for a while, there is nothing that can be done with her anyway. Waldo is sent to the Hallway Junction to meet Sensa and take the container and grasper on his way to fixing Iris who is reported to be out of order.

Now, enter your second set of commands:

#10 POET, TURN SECOND DIAL TO 100
#11 POET, GO TO HALLWAY END
#12 IRIS, GO TO MAIN SUPPLY ROOM
#13 SENSA, PUT RAMP AT DROPOFF
#14 AUDA, LISTEN
#15 WALDO, TAKE CONTAINER AND GRASPER
#16 WALDO, GO TO MAIN SUPPLY
#17 WALDO, INSTALL GRASPER
#18 WALDO, TAKE RED IC AND YELLOW IC
#19 SENSA, GO NORTH
#20 SENSA, TAKE RAMP

Once Poet got to the Weather Control, he had to reset the faulty control to 100. This is only a temporary fix because if you let the game go on too long, all hell will break loose with various disasters and accidents occuring which you will not be able to control. Poet is then sent to the Hallway End where he will be used to get the TV camera needed later. Iris is sent to the Main Supply Room where she can be fixed when Waldo arrives. She will also help Waldo repair the machine there. Sensa, upon arriving, puts the ramp in place so that Auda can get to Gamma Repair and so that she and Poet can get to the other level. Waldo is handed the grasper and container which he installs. This is done now to save moves later.

#21 SENSA, GO TO SMALL SUPPLY
#22 WALDO, OPEN PANEL
#23 WALDO, REPLACE ROUGH DEVICE WITH
ROUGH OBJECT
#24 WALDO, CLOSE PANEL
#25 POET, GET IN CAR
#26 POET, GET OUT OF CAR
#27 POET, GO TO BIOLOGY LAB
#28 WALDO, TAKE BURNED AND FRIED CHIP
#29 POET, TAKE CAMERA
#30 SENSA, PUT RAMP AT HOLDER

This stage sets the robots to their major gathering tasks. Waldo has fixed Iris, and is now set on fixing the machine and salvaging its parts. Poet has arrived at the Hallway End, gotten in the car, exited the car, and is now getting the camera. Sensa has arrived at the Small Supply Room to take the cable cutter.

#31 SENSA, GET ON RAMP
#32 SENSA, TAKE CUTTER
#33 SENSA, GET OFF RAMP
#34 SENSA, TAKE RAMP
#35 SENSA, GO TO SLOPING CORRIDOR
#36 POET, GO TO VEHICLE DEBARKATION
#37 WALDO, PUT RED IC IN RED SOCKET
#38 WALDO, PUT YELLOW IC IN YELLOW
SOCKET
#39 POET, GET IN CAR
#40 POET,GET OUT OF CAR

Sensa has completed her task of getting the metal tool that she will need shortly. Poet has gotten the camera and is now coming back, and Waldo is in the midst of fixing and salvaging the machine with Iris in the Main Supply Room.

You're more than half-way to your goal! Now, enter:

#41 POET, GO TO PRIMARY CHANNEL
#42 SENSA, PUT RAMP AT DROPOFF
#43 WALDO, GO TO GAMMA REPAIR
#44 WALDO, PUSH BUTTON
#45 IRIS, TAKE FUSE
#46 WALDO, TAKE CABLE
#47 WALDO, GO TO THE SECONDARY
CHANNEL
#48 IRIS, GO TO MIDDLE SUPPLY
#49 IRIS, TAKE CABLE
#50 IRIS, GO TO MAIN SUPPLY

Poet has been sent to use the camera in the Primary Channel (this is a Kamikaze mission because he has to pass through the Cavernous Room to get there). Sensa has gotten to the Sloping Corridor and reinstalled the ramp so that she and Poet can get to the lower level. Waldo and Iris have gotten the machine fixed and salvaged one of the two needed cables to set the FCS in balance. Waldo is now set on his mission where Whiz will be waiting to install the cable needed in the Secondary Channel.

#51 SENSA, EXAMINE OBJECT
#52 SENSA, TURN FLOWSWITCH
#53 BOTH SENSA AND AUDA, MOVE FRED
#54 SENSA, CUT CABLE WITH CUTTER
#55 POET, PLUG TV1 IN
#56 POET, AIM TV1 AT SIGN

(This is the important "Reset Code." Write it down! It's different for every game.)

#57 SENSA, TAKE CABLE
#58 SENSA, GO TO PRIMARY CHANNEL
#59 IRIS, PUT CABLE IN MACHINE
#60 IRIS, PUT FUSE IN MACHINE

Sensa and Auda salvaged the remaining needed cable to fix the cable in the Primary Channel. Sensa is now on her way. Poet valiantly died trying to work the camera in the Primary Channel after having had corrosive acid spilled on him. Iris has fixed the reset machine which is now only awaiting the installation of the cables in the FCS to reset the systems to set the surface world above right.

#61 WHIZ, GO TO WALDO
#62 WHIZ, TAKE FOURTEEN-INCH CABLE
#63 WHIZ, REPLACE THE NINE-INCH CABLE
WITH THE FOURTEEN-INCH CABLE
#64 WHIZ, DRAG WALDO TO THE EAST END
#65 AUDA, GO TO SLEEP CHAMBER
#66 SENSA, REPLACE FOUR-INCH CABLE
WITH TWELVE-INCH CABLE
#67 IRIS, PRESS ----- CIRCLE
#68 IRIS, PRESS ----- CIRCLE

At this point, the game is over. Only 8,000 are dead and you have succeeded in your mission. It should be noted that this does not answer all the questions and puzzles that are presented in the game, it just tells you how to win the game in the shortest order. Enjoy!

WITNESS

This walkthrough will help you to finish "Witness." This game is a much simpler one than "Deadline," so I hope you won't be disappointed when you see how easy it is to solve this mystery.

Okay, you start South of the house, where you just picked up a matchbook. By the way, none of the stuff that came with the game is really necessary to solving the case. Go North twice to the front door and ring the bell. Phong will let you in. Then just try to go East, and Phong will lead you to the Living Room, where Monica and Mr. Linder are.

Now, wait (get used to doing that, because there's a lot of waiting in this one), and Linder will eventually take you to his office. Sit down in the wooden chair, and Linder will hand you a note. Read it, as it will help waste some time. Now, just do anything (but stay seated!) to make time pass. Show the matchbook to Linder for an interesting reaction, if you like. In any case, you just have to keep waiting.

Eventually, Monica will come in briefly to announce she's going to the movies. This is not what you're waiting for, however! So, keep on waiting, and finally, the murder will occur. Linder will be shot while you sit there, and you can't stop it from happening. Read the description carefully at the moment the shot is fired. There's something odd about it. In fact, the whole thing is a setup.

The first thing to do is stand up, then push the button. Instead of ringing to summon the butler, it causes a strange click to be heard from the clock. At this point, Phong will enter the room. Tell him you want the keys, and he'll hand them over to you. Now, examine the clock. Keyhole seems a little strange, doesn't it? The doorbell rings while you're doing this, so as Phong goes to answer the door, examine the keyhole.

I'll bet you're getting some ideas already! However, you'll need to have the powder analyzed, and Duffy hasn't arrived yet, so wait around until he does. Then get the powder analyzed (you can ignore Stiles, he's only a red herring). While that's being done, examine the window (you can't open the clock yet, it's the one key you don't have). The green wire seems suspicious, so get it for future reference.

Now, go West into the Hallway, then North twice, and open the Butler's door. Go West into the room, and read the mystery book (by the way, you can drop the telegram and note, they aren't important). A gun receipt is used as a bookmark. The purchaser's name is obviously phoney, but hang on to the receipt anyway.

Okay, from the Butler's Room, go East twice to Monica's room, then unlock and open the back door. Go East into the Backyard, then South twice to the office path. Aha, a muddy gun! No fingerprints, alas, but you might want to take it along with you, just in case. Now, go West into the Side Yard.

Hmm, more footprints here, but they aren't quite the same as the ones on the office path. In fact, it looks like someone was standing here for awhile. Wonder who it might have been? (No, *not* Sergeant Duffy!). Anyway, go West again to the driveway, then North and East into the Garage.

Unlock and open both the garage door and the workshop door, then go East into the Workshop. The place looks like an electrician's paradise, and there isn't much you can do here; but, there are spools of wire hanging around. Could it be...? Examine spool, and you have established a link of sorts between this place and the study. The green wire is obviously from this room. Now, all you need is the person who put it there.

You now stand there waiting for Monica. Just keep waiting; she'll arrive (saying "Wait for Monica" is easiest). It will take a while, so if you want to hunt down Phong and ask him about the gun receipt, you have time. When she does get there, she'll fiddle briefly with the junction box (very suspicious! before noticing you). Now, wait until she leaves, then follow her. You *must* use directions here, just saying "Follow Monica" won't work.

Follow her all the way to her room, and wait for her to come back out of the bathroom. When she returns, ask her about Mr. Linder. Her response will establish the motive. Now, wait some more, and she will eventually leave the room. Follow her again, this time to the office.

As soon as you get in there, handcuff her. Somewhere along the way, Sgt. Duffy will have left with the body, so you can't arrest her until he comes back. In the meantime, you have to find some very important evidence. So, first search Monica for the key. When you get it, unlock the clock and open it. She's already removed the gun, but you can search her for that, also.

Now, just wait until Duffy returns, and arrest her for the murder. And that should be about it. By the way, if you try leaving her and waiting in the office (so you can find the gun in the clock), you'll find that, however hard you try, you won't be able to handcuff her (which is necessary so she can be searched). So, you'll just have to wait and follow her. As I said in the beginning, it's a pretty simple game.

TRSTimes on DISK #14

is now available, featuring the programs from the Jul/Aug, Sep/Oct, and Nov/Dec 1994 issues.

U.S. & Canada:	$5.00 (U.S.)
Other countries:	$7.00 (U.S.)

TRSTimes on Disk
5721 Topanga Canyon Blvd. #4
Woodland Hills, CA 91367
U.S.A.

TRSTimes on Disk #1 through #13 are still available at the above prices

MODEL 4/4P/4D OWNERS!

Forget
SYSRES & MEMDISK.

Now there's

QuikDisk

QuikDisk converts the top 64K of your 128K Model 4 to a large disk I/O buffer. Sophisticated data management techniques ensure frequently accessed disk data is almost always ***instantly*** available.

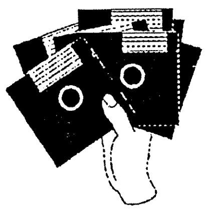
QuikDisk provides ***dramatic*** disk I/O speed increases on both floppy and hard drive systems.

"SmartDrive" is so good, they built it into the latest MS-DOS so no one would be without it. Don't ***you*** be without this ***essential*** type of utility even one day longer.

QuikDisk is only $31.95 +$3 S&H (add $2 outside North America. VA residents please add $1.44 (4 1/2%)). 128K required. Not intended for systems with XLR8er or other large memory expansion boards.

Order **QuikDisk** from:

J.F.R. Slinkman
1511 Old Compton Road
Richmond, VA 23233.



THE TEXAS CITIZENSHIP TEST

By Thomas Pesek and Bruce Wehrle

Not since the days of Santa Anna has the great state of Texas seen an invasion to rival the one which has occured in the 1980 and 90's. Millions of half-frozen Yankees, quiche-eating Californians, Mexicans, Vietnamese and Central American refugees invaded Texas looking for the Promised Land. Many of these people have had a hard time understanding exactly what it means to be a TEXAN. They find it hard to understand the fierce loyality and pride that a TEXAN feels for his state. Well, as one of these fiercely loyal TEXANS, I decided to help out these wayward souls. The best things in life are reserved for the chosen few. Only a few were lucky enough to be born in Texas- to be NATIVE TEXANS. But don't dispair; we have chosen to allow the rest of you poor unfortunate wretches to become the next best thing — A NATURALIZED TEXAN. No longer will you have to lament 'Ah, to be a Texan'. Here is your chance. By passing the following test you can become a NATURALIZED TEXAN. A score of 70% or higher will allow you to hold your head up with pride and say 'I'm a TEXAN'. --- Good Luck!

```
10 '******************************************
20 '**  TEXAS CITIZENSHIP TEST              **
30 '**       VERSION 2.0                    **
40 '**     By: Thomas Pesek                 **
50 '**    and: Bruce Wehrle                 **
60 '** Reference:  The Texas Almanac        **
70 '******************************************
80 '
90 I=1:CORRECT=0:Y$=CHR$(14)Nn$=CHR$(15):
R$=CHR$(16):S$=CHR$(17
100 CLS:PRINT n$
110 PRINT@(7,13),CHR$(151);STRING$(50,131);CHR$(171)
120 PRINT@(8,13),CHR$(149);STRING$(50,32);CHR$(170)
130 PRINT@(9,13),CHR$(149);STRING$(14,32);
"TEXAS CITIZENSHIP TEST";STRING$(14,32);CHR$(170)
140 PRINT@(10,13),CHR$(149);STRING$(50,32);CHR$(170)
150 PRINT@(11,13),CHR$(149);STRING$(20,32);
"VERSION 2.0";STRING$(19,32);CHR$(170)
160 PRINT@(12,13),CHR$(149);STRING$(4,32);
"Prepared by: Thomas Pesek (A Native Texan)";STRING$(4,32);
CHR$(170)
170 PRINT@(13,13),CHR$(149);STRING$(2,32);
"Assisted by: Bruce Wehrle (A Naturalized Texan) ";CHR$(170)
180 PRINT@(14,13),CHR$(149);STRING$(50,32);CHR$(170)
190 PRINT@(15,13),CHR$(149);STRING$(9,32);
"Reference:    The Texas Almanac";STRING$(9,32);CHR$(170)
200 PRINT@(16,13),CHR$(149);STRING$(50,32);CHR$(170)
210 PRINT@(17,13),CHR$(141);STRING$(50,140);CHR$(142)
220 '
230 '
240 '
250 PRINT@(23,24),"Press any key to continue ";Y$;
260 A$=INKEY$: IF A$="" THEN 260
270 CLS:PRINT N$;
290 PRINT@(4,24),"Texas is called:"
300 PRINT@(6,29),"(a) The Cowboy State"
310 PRINT@(8,29),"(b) The Lone Star State"
320 PRINT@(10,29),"(c) The Armadillo State"
330 PRINT@(12,29),"(d) None of the above ";
340 PRINT Y$;
350 A$=INKEY$: IF A$="" THEN 350 ELSE PRINT N$;
360 PRINT@(8,29),R$;"(b) The Lone Star State ";S$
370 PRINT@(18,9),"Texas is often called the Lone Star State
because its flag has a single star."
380 PRINT"This flag was originally the flag of the Republic of
Texas and was approved","on Jan. 25, 1839 by Mirabeau B.
Lamar.";
390 IF A$="b" OR A$="B" THEN GOSUB 4300 ELSE
GOSUB 4350
400 GOSUB 4200
410 PRINT@(4,24),"Texas has been under ___ flags."
420 PRINT@(6,33),"(a) 2":
430 PRINT@(8,33),"(b) 4"
440 PRINT@(10,33),"(c) 6"
450 PRINT@(12,33),"(d) 1 ";Y$;
460 A$=INKEY$: IF A$="" THEN 460 ELSE PRINT N$;
470 PRINT@(10,33),R$;"(c) 6 ";S$
480 PRINT@(18,9),"Six different flags have flown over
Texas during 8 changes of sovereignty."
490 PRINT"The accepted sequence is as follows:
Spanish- 1519-1685; French- 1685-1690;","Spanish-
1690-1821; Mexican- 1821-1836; Republic of Texas- 1836-
1845;","U.S.- 1845-1861; Confederate States- 1861-1865;
U.S.- 1865-Present"
500 IF A$="c" OR A$="C" THEN GOSUB 4300 ELSE
GOSUB 4350
510 GOSUB 4200
520 PRINT@(4,24),"The State Flower of Texas is the"
530 PRINT@(6,29),"(a) Bluebonnet"
540 PRINT@(8,29),"(b) Yellow Rose"
550 PRINT@(10,29),"(c) Daisy"
560 PRINT@(12,29),"(d) Tumbleweed ";Y$;
570 A$=INKEY$: IF A$="" THEN 570 ELSE PRINT N$;
580 PRINT@(6,29),R$;"(a) Bluebonnet ";S$
590 PRINT@(18,9),"The state flower of Texas is the
Bluebonnet, also called buffalo clover,"
600 PRINT"wolf flower, and 'el conejo' (the rabbit). The
```

```
Bluebonnet was adopted as","the state flower by the 27th legislature in 1901 at the request of the","Society of Colonial Dames of Texas."
610 IF A$="a" OR A$="A" THEN GOSUB 4300 ELSE GOSUB 4350
620 GOSUB 4200
630 PRINT@(4,19),"The State Motto of Texas is"
640 PRINT@(6,24),"(a) If you don't have an oil well, get one!"
650 PRINT@(8,24),"(b) Lone Star longnecks forever"
660 PRINT@(10,24),"(c) Friendship"
670 PRINT@(12,24),"(d) The Oil State"
680 PRINT@(14,24),"(e) Yankee go home ";CHR$(14);
690 A$=INKEY$: IF A$="" THEN 690 ELSE PRINT N$;
700 PRINT@(10,24),R$;" (c) Friendship ";S$
710 PRINT@(18,0),"The state motto of Texas is 'Friendship.' The word, Texas, or Tejas,"
720 PRINT"was the Spanish pronunciation of the Caddo Indian word meaning 'friends'","or  'allies.' It was adopted by the 41st legislature in 1930."
730 IF A$="c" OR A$="C" THEN GOSUB 4300 ELSE GOSUB 4350
740 GOSUB 4200
750 PRINT@(4,19),"The State Bird is the"
760 PRINT@(6,24),"(a) Obscene gesture seen on freeways"
770 PRINT@(8,24),"(b) Roadrunner"
780 PRINT@(10,24),"(c) Vulture"
790 PRINT@(12,24),"(d) Whooping Crane"
800 PRINT@(14,24),"(e) Mockingbird ";Y$;
810 A$=INKEY$: IF A$="" THEN 810 ELSE PRINT N$;
820 PRINT@(14,24),R$;" (e) Mockingbird ";S$
830 PRINT@(18,0), "The state bird of Texas is the Mockingbird. It was adopted by the"
840 PRINT"40th legislature in 1927 at the request of the Texas Federation of Women's","Clubs."
850 IF A$="e" OR A$="E" THEN GOSUB 4300 ELSE GOSUB 4350
860 GOSUB 4200
870 PRINT@(4,24),"Texas Independence Day is:"
880 PRINT@(6,29),"(a) January 1"
890 PRINT@(8,29),"(b) March 2"
900 PRINT@(10,29),"(c) July 4"
910 PRINT@(12,29),"(d) June 10 ";Y$;
920 A$=INKEY$: IF A$="" THEN 920 ELSE PRINT N$;
930 PRINT@(8,29),R$;" (b) March 2 ";S$
940 PRINT@(18,0),"On March 2, 1836 Texas' Declaration of Independence from Mexico was adopted"
950 PRINT"at Washington-on-the-Brazos. The convention was chaired by Richard Ellis,","but the Declaration was written almost entirely by George C. Childress.","It was signed by 58 delegates."
960 IF A$="b" OR A$="B" THEN GOSUB 4300 ELSE GOSUB 4350
970 GOSUB 4200
980 PRINT@(4,14),"The distance from one end of Texas to the other is:"
990 PRINT@(6,29),"(a) about 800 miles"
1000 PRINT@(8,29),"(b) about 200 miles"
1010 PRINT@(10,29),"(c) about 2000 miles"
1020 PRINT@(12,29),"(d) about 1 light-year ";Y$;
1030 A$=INKEY$: IF A$="" THEN 1030 ELSE PRINT N$;
1040 PRINT@(6,29),R$;" (a) about 800 miles ";S$
1050 PRINT@(18,0),"The distance from Orange on Texas' eastern border to El Paso on it's west"
1060 PRINT"is 855 miles. A car traveling this distance at 55 mph would require","15.5 hours of non-stop driving to transit Texas. Texarkana, Texas is"
1070 PRINT"closer to Chicago, Ill.(779mi) than to El Paso, Texas (821mi)!","In 1980, there were 70,605 miles of paved roads in Texas."
1080 IF A$="a" OR A$="A" THEN GOSUB 4300 ELSE GOSUB 4350
1090 GOSUB 4200
1100 PRINT@(4,4),"When Texas was annexed, the U.S. government agreed to allow it to"
1110 PRINT@(5,4),"split up at some later date, if desired, into:"
1120 PRINT@(7,29),"(a) up to 10 States"
1130 PRINT@(9,29),"(b) a State and a Republic"
1140 PRINT@(11,29),"(c) up to 5 States"
1150 PRINT@(13,29),"(d) 47 large cattle ranches "; Y$;
1160 A$=INKEY$: IF A$="" THEN 1160 ELSE PRINT N$;
1170 PRINT@(11,29),R$;"(c) up to 5 States ";S$
1180 PRINT@(18,0), "When Texas was annexed by the United States on Dec. 29, 1845, it had a huge"
1190 PRINT"land area and few people. The U.S. agreed to allow Texas to split up into","up to 5 States at some later time when it's population increased,","if it chose to."
1200 IF A$="c" OR A$="C" THEN GOSUB 4300 ELSE GOSUB 4350
1210 GOSUB 4200
1220 PRINT@(4,24),"The current Governor of Texas is:"
1230 PRINT@(6,29),"(a) Howdy Doody"
1240 PRINT@(8,29),"(b) Howard Hughes"
1250 PRINT@(10,29),"(c) Ann Richards"
1260 PRINT@(12,29),"(d) Lyndon Johnson"
1270 PRINT@(14,29),"(e) Dead, but not yet buried "; Y$
1280 A$=INKEY$: IF A$="" THEN 1280 ELSE PRINT N$;
1290 PRINT@(10,29),R$;"(c) Ann Richards ";S$
1300 PRINT@(18,0),"What can we say, if you picked (e) give yourself half credit. She's not dead,"
1310 PRINT"it just seems that way at times."
1320 IF A$="c" OR A$="C" THEN GOSUB 4300 ELSE GOSUB 4350
1330 GOSUB 4200
1340 PRINT@(4,24),"The State Capitol is located in:"
1350 PRINT@(6,29),"(a) College Station"
1360 PRINT@(8,29),"(b) Austin"
1370 PRINT@(10,29),"(c) Luckenbach"
1380 PRINT@(12,29),"(d) Houston"
1390 PRINT@(14,29),"(e) Montrose ";Y$;
1400 A$=INKEY$: IF A$="" THEN 1400 ELSE PRINT N$;
1410 PRINT@(8,29),R$;"(b) Austin ";S$
1420 PRINT@(18,0),"A site on the Colorado river was selected in 1839 by the Capital Commission as"
1430 PRINT"the future capital of Texas. It's selection was confirmed by Congress in 1840.","The city was named in honor of Stephen F. Austin and the
```

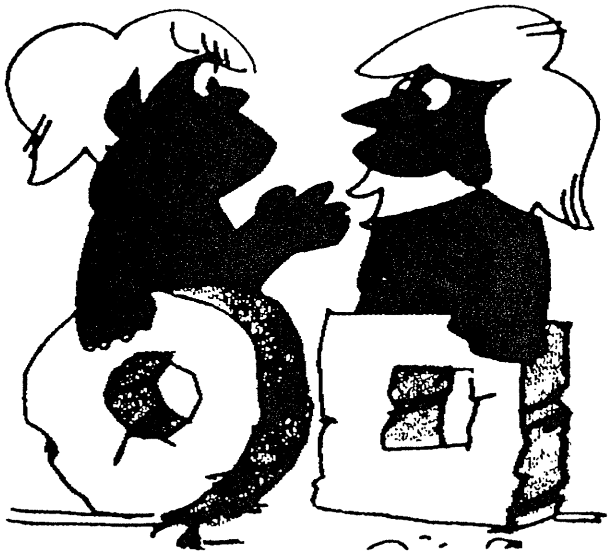
```
government moved","to Austin from Houston in 1840."
1440 IF A$="b" OR A$="B" THEN GOSUB 4300 ELSE GOSUB 4350
1450 GOSUB 4200
1460 PRINT@(4,14),"The first elected President of the Republic of Texas was:"
1470 PRINT@(6,29),"(a) Davy Crockett"
1480 PRINT@(8,29),"(b) Sam Houston"
1490 PRINT@(10,29),"(c) Santa Anna"
1500 PRINT@(12,29),"(d) Ima Hogg"
1510 PRINT@(14,29),"(e) Jim Bowie ";Y$;
1520 A$=INKEY$: IF A$="" THEN 1520 ELSE PRINT N$;
1530 PRINT@(8,29),R$;"(b) Sam Houston ";S$
1540 PRINT@(19,1),"David G. Burnet was named provisional president on Mar. 2, 1836, when Texas"
1550 PRINT"declared itself independent. After the victory at San Jacinto, Sam Houston","was elected president in the first national election in September 1836","defeating Stephen F. Austin and Henry Smith."
1560 IF A$="b" OR A$="B" THEN GOSUB 4300 ELSE GOSUB 4350
1570 GOSUB 4200
1580 PRINT@(4,19),"The infamous Chicken Ranch was:"
1590 PRINT@(6,24),"(a) A fast food restaurant in Austin"
1600 PRINT@(8,24),"(b) A cock fight in Houston"
1610 PRINT@(10,24),"(c) A whorehouse in La Grange"
1620 PRINT@(12,24),"(d) A party house in Marvin Zindler's back yard"
1630 PRINT@(14,24),"(e) A bogus poultry farm run by Billy Sol Estes ";Y$;
1640 A$=INKEY$: IF A$="" THEN 1640 ELSE PRINT N$;
1650 PRINT@(10,24),R$;"(c) A whorehouse in La Grange ";S$
1660 PRINT@(18,0),"The `Chicken Ranch' was the `Best Little Whorehouse in Texas.' It operated just"
1670 PRINT"outside La Grange, Texas for over 100 years until it was closed by Gov.","Preston Smith after a TV expose by Marvin Zindler. The episode inspired","the popular Broadway Play and Movie `The Best Little Whorehouse in Texas'."
1680 IF A$="c" OR A$="C" THEN GOSUB 4300 ELSE GOSUB 4350
1690 GOSUB 4200
1700 PRINT@(4,14),"At the Battle of San Jacinto, what was Santa Anna doing"
1710 PRINT@(5,14),"when the Texans attacked?"
1720 PRINT@(7,24),"(a) Eating tacos & drinking Carta Blanca"
1730 PRINT@(9,24),"(b) Engaged with a prostitute"
1740 PRINT@(11,24),"(c) Shooting pool at Rosa's Cantina"
1750 PRINT@(13,24),"(d) plotting his attack on the Texans ";Y$;
1760 A$=INKEY$: IF A$="" THEN 1760 ELSE PRINT N$;
1770 PRINT@(9,24),R$;"(b) Engaged with a prostitute ",S$
1780 PRINT@(18,0),"On the afternoon of April 21, 1836, the Texans attacked Santa Anna's army"
1790 PRINT"during siesta time. Santa Anna was in his tent with a prostitute. The Mexicans","were routed, with 630 killed, 280 wounded, and 730 captured. The Texans had","9 killed and 30 wounded. Santa Anna fled but was caught the next day,"
1800 PRINT"and forced to surrender to the Texans."
1810 IF A$="b" OR A$="B" THEN GOSUB 4300 ELSE GOSUB 4350
1820 GOSUB 4200
1830 PRINT@(4,14),"If Texas were an independent nation, it would be the"
1840 PRINT@(5,14),"world's _____ largest oil producing nation. (1992 data)
1850 PRINT@(7,29),"(a) first"
1860 PRINT@(9,29),"(b) sixth"
1870 PRINT@(11,29),"(c) twelfth"
1880 PRINT@(13,29),"(d) fortieth ";Y$;
1890 A$=INKEY$: IF A$="" THEN 1890 ELSE PRINT N$;
1900 PRINT@(9,29),R$;"(b) sixth ";S$
1910 PRINT@(18,0),"Texas produced 925,296,000 barrels of crude oil in 1992. If Texas were "
1920 PRINT"an independent nation this would rank it 6th in world production behind","Russia (4,471,250,000 bbls); Saudia Arabia (2,309,435,000 bbls);","U.S. w/o Texas (2,231,419,000 bbls); Iran (1,022,000,000 bbls); and"
1930 PRINT"Mexico (1,002,430,000)."
1940 IF A$="b" OR A$="B" THEN GOSUB 4300 ELSE GOSUB 4350
1950 GOSUB 4200
1960 PRINT@(4,9),"What was the name of Bob Wills' famous Texas country band?"
1970 PRINT@(6,24),"(a) The Lone Stars"
1980 PRINT@(8,24),"(b) Bob Wills & his Wild West Boys"
1990 PRINT@(10,24),"(c) Montezuma's Revenge"
2000 PRINT@(12,24),"(d) The Armadillos"
2010 PRINT@(14,24),"(e) Bob Wills & The Texas Playboys ";Y$;
2020 A$=INKEY$: IF A$="" THEN 2020 ELSE PRINT N$;
2030 PRINT@(14,24),R$;"(e) Bob Wills & The Texas Playboys ";S$
2040 PRINT@(18,0),"Bob Wills and his Texas Playboys were one of the most famous country "
2050 PRINT"bands of the post-war era. Their music inspired modern Texas musicians like","Willie Nelson, Jerry Jeff Walker, and Asleep at the Wheel. Bob Wills' albums ","are still sold today."
2060 IF A$="e" OR A$="E" THEN GOSUB 4300 ELSE GOSUB 4350
2070 GOSUB 4200
2080 PRINT@(4,32),"True or False:"
2090 PRINT@(6,14),"Houston was founded by the Allen brothers in a scheme"
2100 PRINT@(7,14),"to sell swamp land to Yankees. "; Y$;
2110 A$=INKEY$: IF A$="" THEN 2110 ELSE PRINT N$;
2120 PRINT@(9,38),R$;"TRUE! ";S$
2130 PRINT@(18,0),"The Allen brothers came to Texas to make their fortunes. They came to"
2140 PRINT"the area along Buffalo Bayou and founded Houston. Although the land was swampy","and mosquito infested, they placed ads in Eastern newspapers selling land in","Houston and touting its healthful climate. One of the brothers died of Malaria"
```

```
2150 PRINT"and the other brother profited little from the scheme, but Houston was founded."
2160 IF A$="t" OR A$="T" THEN GOSUB 4300 ELSE GOSUB 4350
2170 GOSUB 4200
2180 PRINT@(4,9),"The Battleship Texas is located in a park at the site of a"
2190 PRINT@(5,9),"famous battle of the Texas Revolution.  This battlefield is:"
2200 PRINT@(7,29),"(a) The Alamo"
2210 PRINT@(9,29),"(b) Goliad"
2220 PRINT@(11,29),"(c) San Jacinto"
2230 PRINT@(13,29),"(d) Rosie's Cantina ";Y$;
2240 A$=INKEY$: IF A$="" THEN 2240 ELSE PRINT N$;
2250 PRINT@(11,29),R$;"(c) San Jacinto ";S$
2260 PRINT@(18,0),"The battleship Texas was bought by the school children of Texas when the"
2270 PRINT"Navy mothballed her. She was moved to her present location along the Houston ","Ship Channel in 1948, after serving in WW I and WW II. She is moored in the"
2280 PRINT"San Jacinto Battleground State Park, site of the victory of the Texans over ","Santa Anna on April 21, 1836."
2290 IF A$="c" OR A$="C" THEN GOSUB 4300 ELSE GOSUB 4350
2300 GOSUB 4200
2310 PRINT@(3,14),"Which of these cities was NOT the Capitol of Texas"
2320 PRINT@(4,14),"during the Texas Revolution?"
2330 PRINT@(6,24),"(a) Velasco"
2340 PRINT@(8,24),"(b) Galveston Island"
2350 PRINT@(10,24),"(c) Washington on the Brazos"
2360 PRINT@(12,24),"(d) Harrisburg"
2370 PRINT@(14,24),"(e) San Antonio ";Y$;
2380 A$=INKEY$: IF A$="" THEN 2380 ELSE PRINT N$;
2390 PRINT@(14,24),R$;"(e) San Antonio ";S$
2400 PRINT@(18,0),"After the Declaration of Independence was signed at Washington on the Brazos,"
2410 PRINT"the government of Texas was set up in Harrisburg. As Santa Anna approached,","they fled to Galveston Island. The Treaty ending the war was signed at Velasco."
2420 PRINT"After the war, the Capital was moved to Columbia and then later to Houston,","and finally to Austin.  San Antonio has never been the Capital of Texas."
2430 IF A$="e" OR A$="E" THEN GOSUB 4300 ELSE GOSUB 4350
2440 GOSUB 4200
2450 PRINT@(4,14),"The famous East Texas oil field where the discovery of oil"
2460 PRINT@(5,14),"led to the Texas Oil Boom is called:"
2470 PRINT@(7,29),"(a) The Mother Lode"
2480 PRINT@(9,29),"(b) Spindletop"
2490 PRINT@(11,29),"(c) The Big Thicket"
2500 PRINT@(13,29),"(d) Southfork Ranch ";Y$;
2510 A$=INKEY$: IF A$="" THEN 2510 ELSE PRINT N$;
2520 PRINT@(9,29),R$;"(b) Spindletop ";S$
2530 PRINT@(18,0),"Jan 10, 1901 is the most famous date in Texas oil history. The great Spindletop"
2540 PRINT"gusher erupted at a well near Beaumont drilled by Capt. A. F. Lucas. This well","was the first salt dome oil discovery. By 1902 the field was producing "
2550 PRINT"17,421,000 barrels per year or 94% of the state's production. The resulting","oil glut drove crude oil prices down to 3 cents a barrel, an all time low."
2560 IF A$="b" OR A$="B" THEN GOSUB 4300 ELSE GOSUB 4350
2570 GOSUB 4200
2580 PRINT@(4,19),"The speed limit in Texas is:"
2590 PRINT@(6,24),"(a) 55 MPH"
2600 PRINT@(8,24),"(b) As fast as your car will go"
2610 PRINT@(10,24),"(c) 105 for pickups only"
2620 PRINT@(12,24),"(d) Who cares?"
2630 PRINT@(14,24),"(e) 55 MPH in the country, 85 MPH in Houston ";Y$;
2640 A$=INKEY$: IF A$="" THEN 2640 ELSE PRINT N$;
2650 PRINT@(6,29),R$;"(a) 55 MPH ";S$
2660 PRINT@(18,0),"In 1974, the Federal Government forced Texas to lower the state speed limit to "
2670 PRINT"55 MPH. This has never been a popular law in Texas and as you know if you drive","in Texas, has never been widely obeyed. You might give yourself half credit for","any of the other answers."
2680 "
2690 IF A$="a" OR A$="A" THEN GOSUB 4300 ELSE GOSUB 4350
2700 GOSUB 4200
2710 PRINT@(4,19),"A longneck is a:"
2720 PRINT@(6,24),"(a) Civil War Texas Soldier"
2730 PRINT@(8,24),"(b) freak in a circus sideshow"
2740 PRINT@(10,24),"(c) Type of whooping crane that winter in Texas"
2750 PRINT@(12,24),"(d) returnable beer bottle ";Y$;
2760 A$=INKEY$: IF A$="" THEN 2760 ELSE PRINT N$;
2770 PRINT@(12,24),R$;"(d) returnable beer bottle ";S$
2780 PRINT@(18,0),"If you missed this question, we have a field trip for you. Go to any Country &"
2790 PRINT"Western bar or ice house and yell out loudly: `What the hell are longnecks and","what kind of wimps drink them?' Some fine young cowboy will give you a free","view of a longneck as he shows you some of its many uses."
2800 PRINT"Note: The authors of this test are not responsible for injuries!"
2810 IF A$="d" OR A$="D" THEN GOSUB 4300 ELSE GOSUB 4350
2820 GOSUB 4200
2830 PRINT@(4,32),"True or False:"
2840 PRINT@(6,9),"The borders of Texas once extended to near the Canadian border."
2850 PRINT@(7,9),"When it was annexed by the U.S., the Republic of Texas sold"
2860 PRINT@(8,9),"part of its land to the U.S. for 10 Million dollars.  This"
2870 PRINT@(9,9),"land later became parts of New Mexico, Colorado, Oklahoma,"
2880 PRINT@(10,9),"Wyoming, & Kansas. ";Y$;
2890 A$=INKEY$: IF A$="" THEN 2890 ELSE PRINT N$;
2900 PRINT@(12,38),R$;"TRUE! ";S$
2910 PRINT@(18,0),"In the Compromise of 1850, Texas
```

```
accepted 10 million dollars from the U.S."
2920 PRINT"in return for relinquishing its claim to all lands north and west of its","current borders. Texas used the money to pay off its war debts and to","establish the permanent school fund."
2930 IF A$="t" OR A$="T" THEN GOSUB 4300 ELSE GOSUB 4350
2940 GOSUB 4200
2950 PRINT@(4,19),"A Wetback is:"
2960 PRINT@(6,24),"(a) A pledge in an Aggie fraternity"
2970 PRINT@(8,24),"(b) A type of seagull on Padre Island"
2980 PRINT@(10,24),"(c) An illegal alien"
2990 PRINT@(12,24),"(d) a stolen car with a fresh paint job ";Y$;
3000 A$=INKEY$: IF A$="" THEN 3000 ELSE PRINT N$;
3010 PRINT@(10,24),R$;"(c) An illegal alien ";S$
3020 PRINT@(18,0),"Thousands of illegal aliens enter Texas each year by swimming or wading"
3030 PRINT"across the Rio Grande, hence the term `wetback'. The influx of cheap labor","has been both a blessing and a curse to Texas. The problems created by these","aliens are among the most difficult facing Texas the remainder of this century."
3040 IF A$="c" OR A$="C" THEN GOSUB 4300 ELSE GOSUB 4350
3050 GOSUB 4200
3060 PRINT@(4,9),"In the early days, the only law in West Texas near Pecos was:"
3070 PRINT@(6,19),"(a) Judge Roy Bean & an occasional Texas Ranger"
3080 PRINT@(8,19),"(b) a fast gun & a fast woman"
3090 PRINT@(10,19),"(c) Tortilla justice & Montezuma's Revenge"
3100 PRINT@(12,19),"(d) The Giant Armadillo ";Y$;
3110 A$=INKEY$: IF A$="" THEN 3110 ELSE PRINT N$;
3120 PRINT@(6,19),R$;"(a) Judge Roy Bean & an occasional Texas Ranger ";S$
3130 PRINT@(18,0),"Judge Roy Bean's saloon and court house are located in Langley, Texas"
3140 PRINT"near Pecos. In early Texas, this lawless area was protected by only a few","Texas Rangers. The area was overrun by bandits and outlaws. To tame the","lawlessness the Rangers often dispensed justice with their 45's or they took"
3150 PRINT"their prisoners before Judge Bean and his swift `hang em high' justice."
3160 IF A$="a" OR A$="A" THEN GOSUB 4300 ELSE GOSUB 4350
3170 GOSUB 4200
3180 PRINT@(3,29),"Pronunciation Test:"
3190 PRINT@(5,9),"If a Texan tells you he needs to go to town to buy some `BOB WAHR',"
3200 PRINT@(6,9),"he is going to buy:"
3210 PRINT@(8,24),"(a) a hooker for the evening"
3220 PRINT@(10,24),"(b) a pickup truck"
3230 PRINT@(12,24),"(c) barbed wire for cattle fencing"
3240 PRINT@(14,24),"(d) a mean bull ";Y$;
3250 A$=INKEY$: IF A$="" THEN 3250 ELSE PRINT N$;
3260 PRINT@(12,24),R$;"(c) barbed wire for cattle fencing ";S$
3270 PRINT@(18,0),"Barbed wire, invented in 1873, was first used in Texas about 1879."
3280 PRINT"It spread throughout the range by 1883. Strife arose as open land was","fenced and fence cutting became wide-spread. In 1884, the legislature made","fence cutting a felony- a law which has remained on the books to modern times."
3290 IF A$="c" OR A$="C" THEN GOSUB 4300 ELSE GOSUB 4350
3300 GOSUB 4200
3310 PRINT@(4,32),"True or False:"
3320 PRINT@(6,9),"Citizens of Austin once seized state documents and held them "
3330 PRINT@(7,9),"in an attempt to prevent the capital from being moved to Houston. ";Y$;
3340 A$=INKEY$: IF A$="" THEN 3340 ELSE PRINT N$;
3350 PRINT@(11,38),R$;"TRUE! ";S$
3360 PRINT@(18,0),"When the Mexicans invaded Texas in 1842, President Houston ordered"
3370 PRINT"the government moved to Houston. Citizens of Austin, fearing that Houston","would be partial to the city which bore his name, seized state papers to","prevent the move. Houston finally agreed to move the government to Washington"
3380 PRINT"on the Brazos until the crisis had passed."
3390 IF A$="t" OR A$="T" THEN GOSUB 4300 ELSE GOSUB 4350
3400 GOSUB 4200
3410 PRINT@(4,32),"True or False:"
3420 PRINT@(6,19),"For 6 days in 1865, a jackass named Betty was"
3430 PRINT@(7,19),"the governor of Texas. ";Y$;
3440 A$=INKEY$: IF A$="" THEN 3440 ELSE PRINT N$;
3450 PRINT@(11,38),R$;" FALSE! ";S$
3460 PRINT@(18,0),"Although many people have accused various governors of being jackasses,"
3470 PRINT"there are no historical records to substantiate these claims. However, in 1873,",
"democrat Richard Coke defeated republican governor Ed Davis ending","Reconstruction in Texas. Davis refused to vacate the Capitol and a brief armed"
3480 PRINT"clash occured between followers of Davis and Coke in the Capitol building."
3490 IF A$="f" OR A$="F" THEN GOSUB 4300 ELSE GOSUB 4350
3500 GOSUB 4200
3510 PRINT@(4,19),"The Yellow Rose of Texas was:"
3520 PRINT@(6,24),"(a) a famous Texas mixed drink"
3530 PRINT@(8,24),"(b) the State Flower from 1857-1872"
3540 PRINT@(10,24),"(c) a famous prostitute in early Texas"
3550 PRINT@(12,24),"(d) an exotic type of fajitas ";Y$;
3560 A$=INKEY$: IF A$="" THEN 3560 ELSE PRINT N$;
3570 PRINT@(10,24),R$;"(c) a famous prostitute in early Texas ";S$
3580 PRINT@(18,0),"The song `The Yellow Rose of Texas' was written about a beautiful and"
3590 PRINT"famous prostitute in early San Antonio. Legend has it that she was with","Santa Anna at San Jacinto when the Texans attacked. The battle
```

```
ultimately","changed the course of Texas history."
3600 IF A$="c" OR A$="C" THEN GOSUB 4300 ELSE
GOSUB 4350
3610 GOSUB 4200
3620 PRINT@(4,32),"True or False:"
3630 PRINT@(6,24),"Chili is the State Dish of Texas. ";
Y$;
3640 A$=INKEY$: IF A$="" THEN 3640 ELSE PRINT
N$;
3650 PRINT@(8,38),R$;"TRUE! ";S$
3660 PRINT@(18,0),"Chili was proclaimed the state dish
of Texas by the Texas Legislature"
3670 PRINT"in 1977. As of this writing there was as yet
no official recipe for the ","state dish, and although Lone
Star has tried to proclaim itself the National","beer of
Texas, many people feel that Shiner should have that
right as it is"
3680 PRINT"the only independent locally owned brewery
in Texas."
3690 IF A$="t" OR A$="T" THEN GOSUB 4300 ELSE
GOSUB 4350
3700 GOSUB 4200
3710 PRINT@(4,24),"The State song of Texas is:"
3720 PRINT@(6,29),"(a) The Yellow Rose of Texas"
3730 PRINT@(8,29),"(b) Texas, our Texas"
3740 PRINT@(10,29),"(c) The Eyes Of Texas"
3750 PRINT@(12,29),"(d) Deep in the Heart of Texas"
3760 PRINT@(14,29),"(e) San Antonio Rose ";Y$;
3770 A$=INKEY$: IF A$="" THEN 3770 ELSE PRINT
N$;
3780 PRINT@(8,29),R$;"(b) Texas, our Texas";S$
3790 PRINT@(18,0),"Although all of these other songs
are more widely known, `Texas, our Texas'"
3800 PRINT"is the official state song, selected by the
41st legislature in 1929.","The Eyes of Texas is the
official song of the University of Texas and
is","frequently sung at public gatherings and has a
measure of recognition as an"
3810 PRINT"unofficial state song. I have chosen to use it
for the music in this test."
3820 IF A$="b" OR A$="B" THEN GOSUB 4300 ELSE
GOSUB 4350
3830 PRINT@(23,24),"PRESS ANY KEY TO
CONTINUE";:PRINT@(23,67),"Correct:";CORRECT;
TAB(80)
3840 A$=INKEY$: IF A$="" THEN 3840
3850 CLS
3860 SCORE=INT(CORRECT/30*100)
3870 IF SCORE<70 THEN GOTO 4040
3880 '** YOU PASS THE TEST **
3890 PRINT@(4,31),"CONGRULATIONS !!!"
3900 PRINT@(6,9),"You have answered";CORRECT;
"questions correctly, for a score of";SCORE;"%"
3910 IF SCORE<80 THEN PRINT@(8,11),"You have
passed the Texas Citizenship Test and are now a":
GOTO 3950
3920 IF SCORE<90 THEN PRINT@(8,8),"You have done
well on the Texas Citizenship Test and are now a":
GOTO 3950
3930 IF SCORE<100 THEN PRINT@(8,1),"You have
made an excellent score on the Texas Citizenship Test
and are now a":GOTO 3950
3940 PRINT@(7,2),"You have made a perfect score on the
Texas Citizenship Test and are
worthy":PRINT@(8,33),"of the title":
PRINT@(10,10)," NATURALIZED CITIZEN EMERITUS
OF THE GREAT STATE OF TEXAS ":GOTO 3960
3950 PRINT@(10,15),"NATURALIZED CITIZEN OF
THE GREAT STATE OF TEXAS"
3960 PRINT@(12,11),"and are entitled to all the rights
and priviliges thereof."
3970 PRINT@(15,54),"The Authors---"
3980 PRINT@(17,54),"THOMAS PESEK"
3990 PRINT@(19,54),"BRUCE WEHRLE"
4000 '
4010 PRINT@(23,20)," ===> Press any key to continue
<=== ";
4020 A$=INKEY$:IF A$="" THEN GOTO 4020
4030 GOTO 4180
4040 '** YOU FAIL THE TEST **
4050 PRINT@(4,35),"SORRY !!!"
4060 PRINT@(6,8),"You have answered";CORRECT;
"questions correctly, for a score of";SCORE;"%"
4070 PRINT@(8,8),"You have failed the Texas
Citizenship Test and are not entitled"
4080 PRINT@(10,8),"to be called a citizen of the Great
State of Texas. Go forth to"
4090 PRINT@(12,8),"your nearest library and study, so
that you can pass next time."
4100 PRINT@(14,8),"Lest we become angered and
unleash a pestilence on your house."
4110 PRINT@(16,8),"--or worse yet, give your address to
Aggies and Rednecks!"
4120 PRINT@(18,54)"The Authors---"
4130 PRINT@(20,54),"THOMAS PESEK"
4140 PRINT@(21,54),"BRUCE WEHRLE"
4150 '
4160 PRINT@(23,20)," ===> Press any key to continue
<=== ";
4170 A$=INKEY$:IF A$="" THEN GOTO 4170
4180 CLS
4190 END
4200 '** QUESTION SET UP **
4210 PRINT@(23,0),CHR$(30);:
PRINT@(23,24),"Press any key to continue";:
PRINT@(23,67),"Correct:";CORRECT;
4220 A$=INKEY$: IF A$="" THEN 4220
4230 N=N+1:IF N>6 THEN N=1
4240 CLS
4250 PRINT@(1,29)," QUESTION NUMBER ";I;" ";
4260 PRINT@(23,23),"PRESS LETTER OF CORRECT
ANSWER";
4270 I=I+1
4280 RETURN
4290 '
4300 '** CORRECT ANSWER **
4310 PRINT@(16,22)," CONGRATULATIONS, YOU'RE
CORRECT! ";
4320 CORRECT=CORRECT+1
4330 RETURN
4340 '
4350 '** WRONG ANSWER **
4360 '
4370 PRINT@(16,28)," SORRY, WRONG ANSWER! ";
4380 RETURN
```

PROGRAMMING TIDBITS



Of bits, digits and fingers

FINGER	*Terminating member of the hand.*
DIGIT	*(from Latin digitus=finger) Numeral 0 to 9.*
BIT	*(from BInary digiT) Unit of computer memory able to store one or the other of two alterna tives (... Webster).*

If any reader of my last essay ("A short Boole session", TRSTimes 7.5) tried to verify in Model 4 BASIC my claim that "0 IMP 1 returns 1, but 1 IMP 0 returns 0", he or she would be entirely justified to question my truth in advertising, because...

```
PRINT 0 IMP 1          ...displays -1
PRINT 1 IMP 0          ...displays -2
```

I also claimed that "0 EQV 1" returns 0, even though in TRSTimes 7.3, page 5, the editor made it clear that the XNOR function (same as EQV in Model 4) does not produce a result of 0 or 1. Indeed, it doesn't...

```
PRINT 0 EQV 1          ...displays -2
```

Yet, confusing as they are, all those conflicting claims and computations are correct! The confusion is caused by the difference between binary bits and decimal digits. On the bit level it is always true that "0 EQV 1 returns 0". If a bit in one number is 0, and in the second number the matching bit is 1, then in the result the corresponding bit will be 0. But the result displayed by BASIC (or by a calculator) depends on the values of the original numbers involved in the operation. And so, dear Editor, XNOR or EQV will sometimes produce 0 or 1...

```
PRINT 0 EQV -1         ...displays 0
PRINT 0 EQV -2         ...displays 1
```

This can get even "curioser and curioser"...

```
PRINT 85 EQV -86       ...displays 0
PRINT 84 EQV -86       ...displays 1
```

These and similar shenanigans have to do with the clever way negative integers are encoded in the binary form in most computers, including TRS-80. To understand that, we should first recall how positive integers are encoded.

In the decimal system the positions of digits represent powers of 10 increasing from right to left. The value of the decimal number is the sum of those powers, each multiplied by the corresponding digit. Thus...

decimal 101 = 1*100 + 0*10 + 1*1

Similarly, in the binary system, the positions of bits represent powers of 2 increasing from right to left...

binary 101 = 1*4 + 0*2 + 1*1 = decimal 5

So far so good, but our poor computer knows only "ones" and "zeros". It does not have a "plus" or "minus" sign. To overcome this minor handicap, we have agreed (we, the computerists) that in "signed" binary integers the leftmost bit will function as a sign, sort of. If that bit is 1 then the number is negative. If it is 0 then the number is positive. Thus a decimal value +5 must be properly written 0101.

Unfortunately the sign business is not quite as simple as that. To represent in binary the decimal value -5 we cannot merely change the leftmost bit from 0 to 1. The signed binary integer 1101 would represent the decimal value -3, not -5. This is because the leftmost "sign" bit is not just an arbitrary symbol like our "minus" sign. In a mathematical sense it is a "borrowed" negative quantity equal to the power of two indicated by its position. The values of all other bits are the same as in a positive number. In 1101 the sign bit represents the 3rd power of two or 8, so its value is -8. Since the total value of the other bits is +5, the "net worth" of 1101 is -3.

This binary "sign" trick has one peculiar consequence: unlike digits in decimal numbers, bit patterns are only meaningful in the context of a system in which all values are represented by the same fixed number of bits. For example our 1101 represents -3 only in a system that uses 4 bits. In a 16-bit integer system (like that used in our TRS-80's) the same value -3 looks like this...

```
        11111111 11111101
```

Each time we move the sign bit one position to the left, its negative value doubles. But at the same time the previous sign bit is not the sign anymore, so its value changes from negative to positive, in effect adding double its absolute value to the total. These two "doubles" cancel each other and the "net worth" after each such "extension of the sign" remains the same, no matter how many extra "ones" we add to the left of a negative sign bit.

So, to encode a negative number in a system with a given number of bits, we should first compute the negative value of the leftmost sign bit, and then figure out the bits of a positive value which added to the negative value of the sign yields the desired negative "net worth". A complicated mouthful. Fortunately it's easier to do than it sounds. Let's say we have a 4-bit system and want to find the bit pattern of a negative value -5. The leftmost sign bit is of course 1. Its value is -8 so the total of the other 3 bits must be -5-(-8) or +3. So now we are looking for a pattern of 3 bits that represent +3. There are several ways to do it, but the easiest is a repeated division by 2. If there is a remainder then we know that we have "1" in the rightmost bit. Otherwise we have "0". Why? Because a binary number divided by 2 leaves a remainder 1 only if the rightmost bit is "1" (the values of all other bits are multiples of 2). After each division by 2 all bits "shift" to the right because now each bit must represent half of its previous value. So each division tells us whether the next bit is "1" or "0". To extract the 3 bits in our example we would repeat the division 3 times...

3 / 2 = 1 remainder 1	bit pattern ??1
1 / 2 = 0 remainder 1	bit pattern ?11
0 / 2 = 0 remainder 0	bit pattern 011

Putting this together with the sign bit gives 1011. Obviously this procedure works for any number of bits.

With this background in mind we can write a simple BASIC routine to produce bit patterns of "signed" integers in any bit system up to 32 bits, i.e. up to the so-called "long integers" popular with the C-language crowd. The routine takes a desired number of bits (z) and a value to convert (z#) and returns a string (z$) of 0's and 1's. If the requested values are out of range then "Overflow" is returned instead. Mod-I and Mod-III users replace the exponentiation caret ^ by bracket [(up arrow) in line 130.

```
100 'BITSTR subroutine
110 c# = z#: z$ = "": s$ = "0"
120 if z<1 or z>32 then z$="(bits!)": goto 300
130 i# = -(2 ^ (z-1)): j# = abs(i#) - 1
140 if c#<i# or c#>j# then z$="(value!)": goto 300
150 if c#<0 then c#=c#-i#: s$="1"
160 if z = 1 then 240

200 for x = 1 to (z-1)
210 k# = c#/2: c# = fix(k#)
220 if k#>c# then z$="1"+z$ else z$="0"+z$
230 next
240 z$ = s$ + z$: RETURN

300 z$ = "Overflow " + z$: RETURN
```

The 100-series of lines are preliminaries. In line 130 we calculate the lowest and the highest possible values that can be represented by the specified number of bits. It follows from our preceding discussion that we get the lowest negative value when the "sign" bit is 1 and all the other bits are 0's. The highest positive value is the opposite: the "sign" bit is 0 and all the other bits are 1's. For example in a 4-bit system the lowest value is 1000 or decimal -8, the highest is 0111 or decimal +7. As a general rule, the highest integer in any system is one less than the absolute value of the lowest negative integer.

If the number to convert is negative then in line 150 we compute a positive value which together with the negative value of the "sign" adds up to the number we have just entered. In that case we set the "sign" bit as "1".

The 200-series of lines is a loop that keeps extracting the bits by repeatedly dividing the positive component of our number by 2. Finally in line 240 we put the "sign" together with the rest of the result and return. One special case is when the user specified a 1-bit system. In such a crippled but perfectly valid system there is only a "sign" bit "0" or "1", so the loop was skipped in line 160.

To test the routine run the following program. The loop in the 20-series of lines displays the string of bits. To improve its legibility, in line 22 we insert a blank space every 8 bits.

```
11 clear 1000 '''this line for Mod-I or III only
12 print "How many bits (1-32)? ";
13 line input k$: if k$ = "" then END
14 z = fix(val(k$))
15 print "Enter decimal number: ";
16 line input k$: if k$="" then 12
17 z# = fix(val(k$)): gosub 100

20 for x=1 to len(z$)
21 print mid$(z$,x,1);
22 if (x/8)=fix(x/8) then print " ";
23 next: print
24 goto 15
```

If you play with this program, especially with a larger number of bits, say 16 or more, you might notice an interesting thing. In two numbers that differ only by their sign, several bits (but never all) are often just the opposites (or "complements") of each other. For example...

```
+1234 displays 00000100 11010010
-1234 displays 11111011 00101110
```

These patterns hint at the so-called "two's complement" method used internally by the computer to change positive integers into negative, and the other way around. The method is a two-step process. First all bits in a number are flipped or "inverted": 0's become 1's and 1's become 0's (this first step is called "one's complement"). In the second step 1 is added to that flipped value. Let's try it...

```
+1234         00000100 11010010
flip          11111011 00101101
add 1         00000000 00000001
result -1234  11111011 00101110
```

Adding in binary is pretty simple: 1+1 = 2, in binary 10, so write 0, and carry 1. In this example the carry is added to a 0 in the second bit of the "flipped" number, so write 1, and that's the end of it. In some numbers the carry may have to go back farther than that, but it always stops as soon as a 0 is found in the "flipped" number. The remaining bits in the left part of the final result will remain as "one's complements" of the original bits.

As noted, the method works the same way in reverse. If you take the bit pattern for -1234 and go through the same steps (flip bits, then add 1), you'll get the bit pattern for the positive value +1234.

The rationale behind this mechanical "flip, add 1" method is very simple, though to my knowlege never explained in the computer literature. Consider: a negative number is a positive number subtracted from 0. For example...

$$-1234 = 0 - 1234$$

This equation remains valid if we simultaneously add and subtract 1 on the right-hand side...

$$-1234 = 0 - 1234 + 1 - 1$$

After some re-arranging of the right-hand side we get...

$$-1234 = (-1 - 1234) + 1$$

Using our BASIC program you can see that for any number of bits the value of -1 is always represented by a series of 1's, without any 0's. This is so because, if we can "extend the sign" by adding extra 1's in front of the negative sign bit, we can also "shrink the sign" by deleting any extra "leading" 1's. This reduces 111... to a one-bit system in which only a single "1" remains. Since this lonely "1" is now the sign bit whose value is -1, the value of 111... always boils down to -1, no matter how many 1's were in the original.

The advantage of binary subtracting from a bunch of 1's is that there is never a need to "borrow": 1 minus 1 is 0, 1 minus 0 is 1, and that's it...

```
-1              11111111 11111111
subtract 1234   00000100 11010010
result          11111011 00101101
```

In effect the subtraction from -1 always simply "flips" all bits, which is the first step in the "two's complement" method. To complete the rest of the equation we now add 1, the second step of the method. Similarly, a positive number can be understood as a negative number subtracted from 0, and that's why the method works both ways.

The reverse job of finding the value of a negative signed bit pattern is more straightforward: find the negative value of the leftmost "sign" bit, and add to it the total of positive values of all other bits. As a shortcut we can "shrink the sign", i.e. delete any extra leading 1's. For example...

```
11111111 10011011 or 10011011
```

will yield the same negative value. Bit if this procedure still seems too tedious, try this BASIC program...

```
11 clear 1000 """this line for Mod-I or III only
15 print "Enter binary number: ";
16 line input z$: if z$ = "" then END
17 gosub 500
20 if z$="Overflow!" then print z$ else print z#
22 goto 15

500 'BITVAL subroutine
510 n# = 1: z# = 0: j# = 2 ^ 31
600 for x = len(z$) to 2 step -1
610 k$ = mid$(z$,x,1)
620 if k$=" " then 650
630 z# = z# - n# * (k$="1")
640 n# = 2 * n#: if n#>j# then x=0
650 next
660 if x < 0 then z$ = "Overflow!"
670 s# = n# * (left$(z$,1) = "1")
680 z# = s# + z#: RETURN
```

The subroutine is more or less the reverse of the BITSTR routine. If you enter more than 32 bits then it returns "Overflow" (but spaces typed for legibility between groups of bits are ignored in line 620). To simplify the routine, we assume that any bit which is not "1" means "0", so you get the same value from "1101" as from "11x1", etc.

Now, going back to the question why, for example...

```
PRINT 0 IMP 1                    ...displays -1
PRINT 1 IMP 0                    ...displays -2
```

Recall the "truth table" of IMP discussed last time...

(a)	(b)	(a) IMP (b)
0	0	1
0	1	1
1	0	0
1	1	1

Read the rows of the table: "when the first bit is 0 and the second 0, the result is 1", "first 0, second 1, result 1", etc. So let's see what the truth table does to "0 IMP 1" in BASIC...

```
(a) = decimal 0 00000000 00000000
(b) = decimal 1 00000000 00000001
(a) IMP (b)             11111111 11111111
```

By now we easily recognize that the result represents -1. As for "1 IMP 0"...

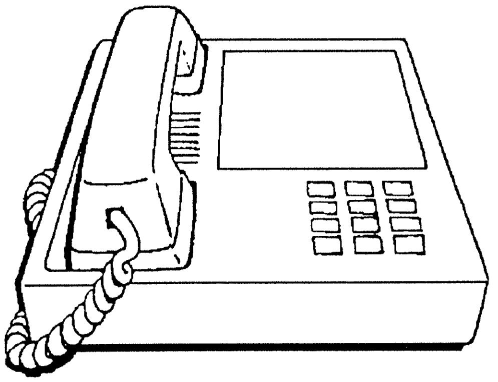
```
(a) = decimal 1 00000000 00000001
(b) = decimal 0 00000000 00000000
(a) IMP (b)       11111111 11111110
```

The result is almost the same as before, except that the rightmost bit is now 0 in accordance with the third row of the truth table (first bit 1, second bit 0, result 0). Since the value of that missing rightmost bit was +1, the result is one less than before, i.e. -2. Strange but true. Similar analysis would show why "0 EQV 1" displays -2, or why "85 EQV -86" displays 0, etc, regardless of the number of bits used by the system.

If I got carried away today, it's because I always find binary systems more fun than decimal. I suspect that God must be of the same opinion. He created male and female, not 10 genders; right and wrong, not 10 excuses; true or false, not 10 maybes; left and right hand, not 10 hands. But man, greedy as usual, demanded more of everything. As a punishment God gave him 10 fingers and ever since, having forfeited the binary bliss, we keep making life more confusing than it was meant to be.

TRSuretrove BBS

8 N 1 - 24 hours
Los Angeles
213 664-5056



where the TRS-80 crowd meets

C PROGRAMMING TUTORIAL

Part 4

by J.F.R. "Frank" Slinkman

Before I get into the guts of this issue's installment, an important word of advice:

As you go through this series, EVERY time a standard library function is referred to or used in a program, you need to look at the documentation for that function, and use the docs as a supplement to the article text. You'll never fully understanding what's going on if you don't do this.

O.K. Let's examine C's powerful, built-in data organization tools.

When it comes to storing data in RAM, BASIC offers only one way (the array) to organize multiple data items into a single logical unit. The limitation of arrays is they can contain only one type of data.

However, if you're sneaky, you can logically combine different data types in RAM by opening a random access disk file, and having two subroutines -- one (subroutine A) to field its buffer with the actual data elements, and one (subroutine B) to field the entire buffer as a single element.

To store data, you would call subroutine A, and fill the buffer with data just as you would for a disk file write. Then call subroutine B, and copy the entire buffer to an element of an array of strings.

The data can later be extracted from the array by calling subroutine B, LSETting an array element into the buffer, calling subroutine A, and accessing the data as if you'd just done a disk file read.

This method treats RAM like a disk file. The array is the "file," the array elements are "records," subroutine B's FIELD command defines the records, and subroutine A's FIELD command defines the fields within the records.

If you can visualize storing data in RAM this way, you're 99% of the way toward understanding how C manages data elements called "structs." Interestingly, and in keeping with the above analogy, structs are sometimes also referred to as "records."

Fortunately, handling structs in C is much easier than the above-described BASIC method.

In addition to arrays and structs, C offers one additional method of data organization, the "union."

A "union" provides a means of looking at the same data in more than one way.

In C, these three methods of data organization -- arrays, unions and structs -- are extremely flexible and versatile.

You can create arrays of structs, for example, or structs of arrays. You can have unions of arrays and structs; structs which contain arrays and unions, and even other structs, just to mention a few of the possibilities.

A "union" of two data elements is declared as follows:

```
union type_name
        {       type    member_name;
                type    member_name;
        } union_name;
```

The variables included in the union are called the union's "members."

"Type_name" is optional, and would normally be used only if more than one of the same type of union is to be declared.

If "type_name" is used, it becomes a "typedef" -- a sort of template which makes it easy, elsewhere in the program, to declare other variables to be unions of the same type. In other words, it defines "union type_name" to be a data type just as "char," "int," and "double" are data types.

Also, "union_name" is optional. If you just want to define the union as a data type for later use, you would define the union similarly to:

```
typedef union type_name
        {       type    member_name;
                type    member_name;
    };
```

Henceforth, anytime you declare a variable to be of type union type_name, the compiler would know the type and size of the data, and how to handle it.

The following program illustrates the use of a data union:

```
/*      prog07.c  */

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

```
#option ARGS        OFF
#option REDIRECT    OFF
#option FIXBUFS     ON
#option MAXFILES    0

union  {        char    string[8];
                        double   value;
                } test;
main()
{
        register int    i;

        printf( "\x1c\x1fval = %13.8f\n\n",
                                test.value = 1.0 / 3.0 );

        test.string[7] -= 8;
        puts( " n\t val * 2^n" );
        puts( "===\t==========" );

        for ( i = -8; i <= 8; )
        {       printf( "% d\t%11.8f\n", i++,
                                        test.value );
                ++test.string[7];
        }
}
```

After the "option" directives, the union "test" is both defined and declared. It contains an 8-element char array named "string" which occupies the same RAM space as the double variable "value."

The size of this union is 8 bytes, since both "string" and "value" are 8 bytes long. If one were longer than the other, the union would take on the size of the larger member.

It's important to realize that both "string" and "value" refer to the same RAM space. Unions do not hold two different values, but merely allow you to look at the same data in more than one way, through the use of two different variable names. This is demonstrated in the main() function.

The first statement in main() uses a new class of variable, namely a "register" variable. Previously, we have used only "auto" and "global" variables.

"Auto" variables are stored on the stack. "Register" variables are stored in CPU registers, and thus can be accessed more quickly than auto variables, but not as quickly as "global" variables (or "static" variables, which will be discussed later).

The next statement assigns the value one-third to the double member of the union. Note how the variable receiving the assignment is specified:

```
    test.value = 1.0 / 3.0;
```

First is the name of the union, followed by a "dot" (period) followed by the name of the member.

Note also how it's legal to combine statements in C. This line is really a combination of two lines:

```
    test.value = 1.0 / 3.0;
    printf( "...", test.value );
```

and executes the same (although more efficiently), as if the two parts were written on two separate lines.

This ability to combine statements is both an asset and a liability. The more statements you combine, the more efficient your program becomes, and the harder it becomes to make sense of your program listing.

I recommend you initially write all statements on separate lines. Then after you've got the program debugged and working properly, go back and combine as many statements as is reasonable as part of the final optimization before final compilation.

The next line, "test.string[7] -= 8" subtracts 8 from the value held in the 8th byte of both "string" and "value."

This is one of the nifty operators in C that saves a lot of key strokes compared to coding in, say, BASIC. For example, the line

```
    burp -= 8;
```

is identical to the BASIC line

```
    BURP = BURP - 8;
```

Some other examples of this type of operator are:

C	BASIC
burp += 8;	BURP = BURP + 8
burp *= 8;	BURP = BURP * 8
burp /= 8;	BURP = BURP / 8
burp %= 8;	BURP = BURP MOD 8
burp &= 8;	BURP = BURP AND 8
burp |= 8;	BURP = BURP OR 8
etc.	

Of course, you COULD have written "burp = burp - 8;" but why?

What we are doing is altering the exponent of the double precision value stored in the "value" member of the union. This has the effect of dividing "value" by 2 to the 8th power.

The next two lines print a header to explain the data which will be displayed.

The "for" loop displays values of the variables "i" and "test.value," and then increments the exponent of test.value, the same as multiplying test.value by two.

Type in, compile and run prog07.c, and you will see the effects of manipulating the exponent of the double-precision "value" member of the union "test."

Actually, this could have been done another way, without the use of a union.

The 8th byte of the double can also be accessed by declaring "string" to be a pointer to char, and then loading it with the RAM address of "value."

Then any one of the eight bytes of the double precision "value" can be addressed as one of the elements of a char array, as follows:

```
double  value = 1.0 / 3.0;
char    *string;

string = (char *)&value;
string[7] -= 8;
```

Casting the address of "value" to a pointer-to-char is identical to the effect of including both in a union, but the use of a union is just as efficient, and helps keep program listings simpler and clearer.

In other words, all a union does is assign the same address to two variables, and tell the compiler to treat the object at those addresses the desired ways.

A good example of a union is found in the MC-provided header file, Z80REGS/H. If you'll LIST that file to your printer, you'll see that it's a union of two structs, one of which contains seven short ints, and the other which contains 14 chars.

This union provides the programmer an easy way to load and/or read the various Z80 registers when making use of the non-standard call() function.

When loading the 16-bit register pairs, the 14-byte space is addressed via the struct which treats it as seven 16-bit short ints. When loading single 8-bit registers, the space is addressed via the struct which treats it as fourteen 8-bit chars.

Notice, too, how #defines are used to further simplify the task of accessing this space.

You may also notice that structs are defined very similarly to unions, namely:

```
struct type_name
        {       type    member_name;
                type    member_name;
                .....
                type    member_name;
        } struct_name;
```

Again, "type_name" is optional, and serves as a typedef to make it easy to declare other variables as being of type "type_name" elsewhere in the program.

"Struct_name" is also optional, if you just want to define the structure via a typedef, as discussed above for unions.

You can use any data type as a struct member, except a struct of the same type. Think about that for a minute, and then think about the number of reflections created by two mirrors facing each other.

However, structs can (and often do) contain POINTERS to structs of the same type (the structs used in the programs unhuf.c and dohuf.c in TRSTimes Vol 7 No 2 being good examples of this).

Also, a struct can have as many members as you care to give it. In some cases, it's even useful to have a struct with only one member.

Frankly, you probably won't have too many occasions when you need to use unions, but structs are so powerful and so useful that you'll soon come to wonder how you ever got along programming in a language like BASIC, which doesn't have them.

The following program demonstrates the use of structs, and gets us deeper into the concept of indirection.

```
/*      prog08.c */

#include        <stdio.h>

typedef struct entry
        {       int             position;
                char    *message;
        };
/*=*=*=*=*=*=*/

main( argc, argv )
int argc; char **argv;
{
        register int    i;
        struct entry    *msg;
        msg = calloc( argc, sizeof(struct entry) );
        if ( !msg )
        {       perror( "calloc()" );
                exit(EOF);
        }

        for ( i = 0; i < argc; )
        {       msg[i].message = argv[i];
                msg[i].position = ++i;
        }

    qsort( msg, argc, sizeof(struct entry), compare );

        for ( i = 0; i < argc; )
        {       printf( "%2d  ", msg[i].position );
                puts( msg[i++].message );
        }

        free( msg );
}
/*=*=*=*=*=*=*/

int compare( a, b )
struct entry *a, *b;
{
        return strcmp( a->message, b->message );
}
```

Notice we didn't use any of the "#option" lines discussed in previous programs.

We didn't use "INLIB" because we are using none of the special functions in IN/REL.

We didn't use "ARGS OFF" because, for the first time, we have a program which has command line arguments (well, sort of).

We didn't use "REDIRECT OFF" because we do want the ability to use standard I/O redirection.

And we didn't use "FIXBUFS ON" because we will be using dynamic memory reallocation, and we didn't use "MAXFILES" because it's only desirable to use that if "FIXBUFS" is "ON."

The first thing we do after including STDIO/H is define a structure type. Notice we did not allocate any RAM to hold structures of this type -- we only told the compiler how to build and use this type of structure in the future.

Structures of "type entry," have two members: a short signed int named "position;" and a pointer to char named "message."

Since both short ints and pointers have 2 bytes on our hardware (on some larger machines, pointers are 4 bytes), the size of a structure of type entry is four bytes.

Now look at the start of the main() function. We see, for the first time, a main() function with arguments. Unless we've used the non-standard option "ARGS OFF," main() always takes exactly two arguments. In the past, we've never had occasion to use them; so we simply ignored them.

The "argc" and "argv" names for these arguments have become a de facto standard. "Argc" (argument counter) is an int containing the number of strings on the command line, and "argv" (argument vector) is a pointer to an array of pointers to char which point to these strings.

Since, in C, strings are arrays of chars, "argv" is a pointer to an array of "argc" number of pointers, each of which in turn point to ASCII data from the command line.

Here we have defined "argv" as "char **argv;". Just as one asterisk indicates a pointer, a double asterisk indicates a pointer to a pointer. Often, you will see "argv" defined as "char *argv[]."

We're now deeper into the concept of indirection than we've ever been before, and it's very important that you understand why "char **argv" means exactly the same thing as "char *argv[]."

Prog08.c can have as arguments will fit on the 80-character command line. For the sake of example, let's say we invoke prog08 with the following syntax:

```
prog08 how now brown cow
```

This syntax produces five arguments for main(), namely the string "prog08," the string, "how," and the strings "now," "brown," and "cow."

The system creates a char array to hold each string on the command line, counting the number of strings as it goes along. Pointers to these strings are held in an array of pointers-to-char.

Thus, in this case, the int "argc" will equal five.

The following may help your understanding:

```
char    string0[7] = { "prog08" };
char    string1[4] = { "how" };
char    string2[4] = { "now" };
char    string3[6] = { "brown" };
char    string4[4] = { "cow" };
```

```
        char*   burp[5];

        burp[0] = string0;
        burp[1] = string1;
        burp[2] = string2;
        burp[3] = string3;
        burp[4] = string4;
```

The above code creates the "burp" array of pointers to char identical to "argv" with the example syntax.

The first 5 declarations cause "string0" to contain the address of the string "prog08," and "string1" to contain the address of the string, "how,", etc.

The sixth declaration sets up the variable "burp" as an array of five pointers to char.

Notice how "char* burp[5]" is used to declare "burp." "Char *burp[5]" would also work, but this form makes it clearer to someone reading the listing that "burp" is an array of pointers and not an array of chars.

Remember, the compiler ignores white space characters; so it sees both forms as "char*burp[5]." Thus this is a convention strictly for the benefit of humans, not of computers or compilers.

Also note that "burp" is an address, not a value. Specifically, it is the RAM address of the first of its five array elements. Thus "burp" is actually a POINTER to its first element.

The address of the string "prog08" is loaded into the first element of the "burp" array and the address of the string "how" is loaded into the second, etc.

Because all of "string0" through "string4" are pointers to char, and the variable "burp" is a pointer to the first of five pointers to char, then "burp" is a pointer-to-pointer-to-char. Thus "char *burp[]" can also be be written as "char **burp."
For the visually oriented reader, the following little diagram may help:

```
        burp -> string0 -> "prog08"
                string1 -> "how"
                string2 -> "now"
                string3 -> "brown"
                string4 -> "cow"
```

If you're struggling with this, don't despair. People even brighter than you (assuming that's possible) have had trouble with it. But once you "see" it, it'll seem so clear and obvious you'll laugh at yourself for not seeing it sooner.

And please don't read any further until you fully understand why "burp" and "argv" are pointers to pointers, and how "argc" and "argv" are used to access the various parts of the command line.

Now, inside main(), we declare the register int variable "i," and a pointer to struct of type entry, "msg." This doesn't reserve any RAM to hold the structure itself. It merely reserves two bytes of RAM to hold a POINTER to a struct of type entry.

The next statement uses the standard calloc() function to allocate enough RAM to hold "argc" number of structs of type entry.

In our example, "argc" will be equal to 5; so calloc() will attempt to reserve a zeroed block of 20 (5 structs times 4 bytes per struct) bytes of RAM.

If calloc() succeeds, it returns a pointer to the first byte of the RAM it allocated. If it fails (i.e., there wasn't enough free RAM available), calloc() returns a NULL pointer.

The next statement tests for calloc() failure by testing the value of "msg." If non-zero, everything is O.K. But if "msg" is zero, then calloc() failed; so we display an error message via the perror() function, and exit the program via the exit() function with a non-zero argument.

Normally, for efficiency, these two statements would be combined as:

```
if (!(msg = calloc(argc, sizeof(struct entry))))
        etc.
```

but this way works, too.

Assuming calloc() succeeds, we now use "argc" and "argv" to initialize the array of structs. Each "message" member is assigned a pointer to a string from the command line, and each "position" member is assigned a number equal to the order of appearance of the string.

For example, the first string on the command line must be "prog08"; so the first structure will have its "message" member loaded with a pointer to that string, and have its "position" member loaded with the value one. (Why one and not zero? Because it's "++i," not "i++.")

After the array elements have been loaded, we use the standard qsort() function to sort the struct array elements in alphabetical order.

The next "for" loop displays the command line strings in their new (sorted) order, showing both their original position and their contents.

We have no further need of the struct array; so the last statement in main() releases the RAM allocated by calloc(), making it available for other uses.

This statement isn't actually necessary in this case, since the next thing which will happen is a return to LS-DOS Ready; but it demonstrates how the free() function is used to release allocated memory. Also, it's a good idea to adopt the habit of releasing all allocated memory when you're through with it.

Now we get to the compare() function. This is a requirement for the use of the qsort() and bsearch() functions. You need to look at the documentation for qsort() before you read any further.

The two arguments passed to compare() are pointers to elements of the array being sorted. In this case, therefore, both "a" and "b" are pointers to struct of type entry.

We want the array sorted in alphabetical order; so we don't need to look at the "position" members -- only the "message" members.

But look at how the "message" members of both "a" and "b" are accessed: "a->message," for example, not "a.message."
This is because "a" and "b" are POINTERS to struct of type entry, not the structs themselves.

If you'll refer back to main(), you'll see we were dealing with the structs themselves when we were accessing the "message" members via "msg[i].message."
The rule is: Use the "dot" operator when working with actual structs (or unions). Use the "points to" ("->") operator with working with pointers-to-struct (or pointers-to-union).

Getting back to the compare() function, we pass the "message" members (which are pointers to char) of both structs to the standard strcmp() function, and return the result back to the qsort() function.

If you'll look at the docs for strcmp(), you'll see it returns a negative number if the first string is alphabetically less than the second string; a zero if the strings are identical; and a positive number if the first string is alphabetically greater than the second.

Note the return code for strcmp() exactly fills the requirements qsort() has of the compare() routine; so no additional processing is required.

This makes sorting strings easy, but what do you do when sorting in numerical order?

Also easy. If "a" and "b" were pointers to ints, for example, you'd simply write:

```
int compare( a, b )
int     *a, *b;
{
        return *a - *b;
}
```

Obviously, this returns a positive value if "a" > "b", a zero if "a" equals "b", and a negative value if "a" < "b."

For floats and doubles, it's easiest to just return fsgn(*a - *b) or dsgn(*a - *b), respectively, e.g.:

```
int compare( a, b )
double *a, *b;
{
        return dsgn( *a - *b );
}
```

For longs (signed or unsigned) or unsigned ints, because the result of "*a - *b" could be outside the range of a signed short, the comparison is a little more involved. For example:

```
int compare( a, b )
long *a, *b;
{
        long    temp;

        if ( !( temp = *a - *b ) )
                return 0;
        else if ( temp > 0 )
                return 1;
        else
                return -1;
}
```

O.K. We have now covered all the ways C can store data, all the language's statements, and many of its operators.

Now let's look at a capability of the language which is extremely difficult if not impossible to accomplish in BASIC: recursion.

A recursive function is simply one which calls itself. The following program uses recursion to calculate the factorials 1! through 12!.

A factorial is the product of all integers from one through the number. For example, "three factorial," which is expressed as "3!," is 1 * 2 * 3 = 6.

```
/*      prog09.c  */

#include        <stdio.h>

#option ARGS            OFF
#option REDIRECT        OFF
#option FIXBUFS         ON
#option MAXFILES        0

long    factorial();
char    *format();
/*=*=*=*=*=*=*/

main()
{
    register int        i;

    puts( "\x1c\x1f\t\t\t\tTable of factorials\n" );

        for ( i = 1; i < 13; i++ )
                printf( "\t\t\t%8d! =%s\n", i,
                        format( factorial( i ) ) );
}
/*=*=*=*=*=*=*/

long factorial( n )
int n;
{
        long    retcode;

        if ( n )
                retcode = n * factorial( n - 1 );
        else
                retcode = 1L;

        return retcode;
}
/*=*=*=*=*=*=*/

char *format( arg )
long arg;
{
        static char             f_buf[16];
        char                    *dest;

        memset( f_buf, 0x20, sizeof f_buf );
        dest = f_buf + 15;

        while( arg )
        {       if ( arg >= 1000L )
                        sprintf(dest -= 4,"%04ld",arg
% 1000L);
                else
                        sprintf( dest -= 3, "%3ld", arg
);
                arg /= 1000L;
        }

        while ( dest = memchr( f_buf, 0, 15 ) )
                *dest = ',';

        return f_buf + 1;
}
```

By now, you should be familiar with everything in the program down to the "for" loop in main().

The printf() statement in that loop is actually three statements in one.

First, "i" is passed to the factorial() function.

Second, the return value from factorial() is passed to the format() function.

Finally, the return value from format() is passed to the printf() standard function.

This is one more example of the advantage of being able to combine statements for efficiency. In this case, combining statements has even made it unnecessary to use intermediate variables for the temporary storage of the various returned values.

Now let's look at the factorial() function. It takes an int argument ("n") and returns a long.

First, we declare "retcode" as a class auto long int. Remember, class auto variables are created on the stack.

Second, we check the value of "n". If "n" is greater than zero, factorial() passes "n-1" to ITSELF!

Let's follow the logic assuming the value 4 has been passed to the function by main().

There will be a total of five successive calls to factorial(): the initial call with "n" = 4, then recursive calls with "n" equal to 3, 2, 1, and finally 0.

The following chart shows the value of each variable at each level of recursion:

```
level   n      n - 1  retcode
=====  === ======= =======
 5             0          -
1
 4             1          0
1
 3             2          1
2
 2             3          2
6
 1             4          3
24
```

At level #5, because "n" is zero, the function doesn't calculate "n-1," but returns the value 1.

Upon return to level #4, this return value will be multiplied by the value of "n" in effect at that level of recursion, namely one. The product of 1 * 1 = 1 is assigned to "retcode" and returned to level #3.

Upon return to level #3, the return value of 1 is multiplied times the value of "n" at level #3, namely two. The product of 2 * 1 = 2 is assigned to "retcode," and then returned to level #2.

Upon return to level #2, the return value of 2 is multiplied times the level #2 value of "n" (3) and the product, 6, is returned to level #1.

Upon return to level #1, the return value of 6 is multiplied by the level #1 value of "n" (4), and the product, 24, is returned -- this time to main().

Obviously, as written, this function simply could not be written as a BASIC subroutine. The variables "n" and "retcode" would keep being overwritten with the new values, and the concept just wouldn't work.

So how can C do this when BASIC can't? The answer lies in the fact that both function arguments and auto class variables are stored on the stack.

Therefore, the only way a function can access such variables is via their locations relative to the CPU's stack pointer (SP) register.

When the factorial() function is called, the calling program first PUSHes the 2-byte int argument onto the stack, then makes the call, which PUSHes a 2-byte return address onto the stack.

The very first action factorial() takes is to reserve the next 4 bytes of stack space for the long (4-byte) variable "retcode." It does this by PUSHing the AF register twice.

Thus, in the initial call, factorial() would see the SP register pointing to the following:

```
SP+6   "n" (the argument)
SP+4   return address to main()
SP+0   "retcode" (the auto variable)
```

The ONLY things factorial() knows about the stack is that the long int "retcode" exists at SP+0, the return address is at SP+4, and the short int "n" exists at SP+6. It doesn't know or care about anything else having to do with the stack.

Now factorial() calls itself with an argument equal to "n-1." It does so by PUSHing this argument and then making the call. Again, the first thing the function does is reserve 4 stack bytes for "retcode."

Thus, at this level, factorial() would see the SP register pointing to the following:

```
SP+14  previous "n"
SP+12  return address to main()
SP+8   previous "retcode"
SP+6   new "n"
SP+4   return address to factorial()
SP+0   new "retcode"
```

But, as far as factorial() is concerned, the ONLY "retcode" exists at SP+0, and the ONLY "n" exists at SP+6. To repeat, the function doesn't know or care what else may or may not be on the stack.

Each time factorial() calls itself, eight more bytes of stack space are used to create new "n" and "retcode" variables and hold the return address to the calling routine.

Return from a function handles the stack in the opposite manner.

The last thing factorial() does before returning is to POP the 4-byte "retcode" variable into the DE and BC registers, which is how functions return long ints.

This causes SP to point to the previous SP+4.

The return is accomplished by POPping the return address into the PC register, leaving SP pointing to the previous SP+6, the "new 'n'."

Immediately upon return, the calling routine POPs the 2-byte "n" argument into the AF register to clear the stack, leaving SP pointing to the previous SP+8, the "previous 'retcode'."

Now, the previous SP+8 is the current SP+0; so this level of recursion sees its original "retcode" at SP+0 and its original "n" at SP+6, and doesn't know or care that SP ever pointed to anything else.

In this way, the variables "retcode" and "n" remain unique for each level of recursion.

Pretty tricky, eh? And very useful for solving certain types of problems.

For example, if you wanted to write a program to play checkers, recursion would be a good way to analyze all the possibilities of future moves.

The program could look at each possible move in turn, and recursively test each one of the opponent's possible responses.

It could then recursively test each possible counter-response, and so on, until all possibilities are exhausted.

It could, for example, consider a win to be a positive response, and a loss or deadlock to be negative, and rate each move on the basis of the fewest number of moves to the end of the game.

Thus the move which generates the smallest positive value (i.e., the fewest number of moves which result in a win) would be the best, and the move which generates the smallest negative value (i.e., the fewest moves to a loss or deadlock) the worst.

Admittedly, such a program would not be fast, but I sure wouldn't want to have to play against it, since it would always play every situation perfectly.

The down side to using recursion is that it can be a bit slow, especially when the recursion goes very deep, i.e., to a great many levels.

Recursion requires the use of many PUSHes and POPs, which are rather time-consuming operations, not to mention the time consumed by many CALL and RETurn instructions.

This high processing "overhead" is why programmers have developed techniques which do the same thing as recursion through such means as keeping the variables in arrays which are indexed by what would be the recursion level.

Needless to say, this approach requires the programmer to either know the maximum possible depth of recursion, or to set an arbitrary limit on how deep recursion is permitted to go.

However, in many situations, such has the Huffman compression and decompression tree navigation process described in the article "Graphics Image Compression," TRSTimes Vol. 7, No. 2., recursion is the best and just about the only practical way to go.

Finally, let's look at the format() function. This is the first function we've written which returns a pointer to char. It also takes a long argument, named "arg."

C has no equivalent of BASIC's PRINT USING command; so we have to create our own formatting routines. What his routine does is insert a comma between groups of 3 digits; so that the value one million, for example, will be displayed as "1,000,000" instead of "1000000."

Here we introduce a new class of variable -- the "static" variable. If you'll remember, global variables are stored in permanently allocated RAM; auto variables are stored on the stack; and register variables are stored in CPU registers.

Static variables are stored just like global variables -- in permanently allocated RAM -- except they can only be accessed directly by the function in which they are declared.

In this case, even though "f_buf" takes 16 bytes of permanently allocated RAM, no function other than format() knows this storage area even exits.

The main reason to use statics is when you want the data to be non-volatile (i.e., you want it to survive the exit of the function in which it is declared.) Either you need it to be there, unchanged, the next time the function is called, or you need to make it available in some way to another function.

There is also another reason to use statics: speed. Statics and globals are accessed faster by the program than by either register or auto variables.

Here we declare "f_buf" to be static because we want its data to be available to the main() function. No, main() cannot directly know "f_buf" even exists, but it can learn of it indirectly, by receiving a pointer to its data, which the format() function provides to main() as its return code.

We also declare "dest" to be an auto pointer to char.

The first line of executable code uses the standard memset() function to fill "f_buf" with spaces.

Next we point "dest" to the last element of the "f_buf" char array.

In the first "while" loop, the "if" statement tests the value of "arg."

If "arg" exceeds 999, then "dest" is reduced by 4, and the value sent to sprintf() is calculated by modulo division, limiting the argument to a maximum value of 999 ("%" is the modulo operator and "arg % 1000L" is the same as "ARG MOD 1000" in BASIC).

The "%04ld" control string specifies that the string written is to be four characters long and be left-padded with zeroes. For example, the value one would be written "0001" and the value 999 would be written "0999."

If "arg" is less than 1000, then "dest" is reduced by 3, and sprintf() is used to write a three character string with leading spaces, not leading zeros.

Then "arg" is divided by 1000, and the process continues until "arg" becomes zero.

This works because, unlike Model 4 BASIC, the results of integer division are not rounded. For example, 999 divided by 1000 will produce a quotient of zero, not one.

After this loop is exited, "f_buf" will contain the ASCII digits properly grouped in threes, separated by null characters.

This is because the leading "0" of each 4-byte string will have been overwritten by the terminating null character of the string written to its left.

This is why the second "while" loop uses the standard memchr() function to locate all null chars (except the last), and replace each with a comma.

Finally, the return value (a pointer to the ASCII string which has been constructed) is specified, and returned to main(), which sends format()'s return value to the standard printf() function to display the formatted string.

Well, that wraps it up for this issue. Next time, we'll REALLY get into structs and recursion, and use these tools to solve some real problems and do some real, worthwhile work.

Anybody have a Mac-Inker for sale?
Also interested in 80-US magazines
& early issues of 80-Micro.
Buying Model I/III/4/2000 programs & machines.
Buying Model 100 machines.
Copa International, Ltd.
Newark, IL 60541

FOR EITHER HI-RES BOARD!
free Shipping

HIDEE

to:Andy Miller
602 W. 15th
Sioux Falls,
SD 57104

MODEL 4

Finally! Hi-RESOLUTION Menu's for DIRECT Users! Now you can use Either HI or LOW Res. MENU'S with your DIRECT by Chris.

With HR,CHR,or SHR files you can Create, or with the Samples supplied. This is a SELF-INSTALL file in less than 5 minutes! Also included, Westminster Chimes instead of the usual BEEP. $29.95 no personal checks,please.

The MODEL-4 Now LOOKS like a MAC! With DOCS.

YES, OF COURSE !

WE VERY MUCH DO TRS-80 !

MICRODEX CORPORATION

SOFTWARE

CLAN-4 Mod-4 Genealogy archive & charting *$69.95*
Quick and easy editing of family data. Print elegant graphic ancestor and descendant charts on dot-matrix and laser printers. *True Mod-4 mode*, fast 100% machine language. Includes 36-page manual. *NEW!*

XCLAN3 converts Mod-3 Clan files for Clan-4 *$29.95*

DIRECT from CHRIS Mod-4 menu system *$29.95*
Replaces DOS-Ready prompt. Design your own menus with an easy full-screen editor. Assign any command to any single keystroke. Up to 36 menus can instantly call each other. Auto-boot, screen blanking, more.

xT.CAD Mod-4 Computer Drafting *$95.00*
The famous general purpose precision scaled drafting program! Surprisingly simple, yet it features CAD functions expected from expensive packages. Supports Radio Shack or MicroLabs hi-res board. Output to pen plotters. *Includes a new driver for laser printers!*

xT.CAD BILL of Materials for xT.CAD *$45.00*
Prints alphabetized listing of parts from xT.CAD drawings. Optional quantity, cost and total calculations.

CASH Bookkeeping system for Mod-4 *$45.00*
Easy to use, ideal for small business, professional or personal use. Journal entries are automatically distributed to user's accounts in a self-balancing ledger.

FREE User Support Included With All Programs !

MICRODEX BOOKSHELF

MOD-4 by CHRIS for TRS/LS-DOS 6.3 *$24.95*
MOD-III by CHRIS for LDOS 5.3 *$24.95*
MOD-III by CHRIS for TRSDOS 1.3 *$24.95*
Beautifully designed owner's manuals completely replace obsolete Tandy and LDOS documentation. Better organized, with more examples, written in plain English, these books are a *must for every TRS-80 user.*

JCL by CHRIS Job Control Language *$7.95*
Surprise, surprise! We've got rid of the jargon and JCL turns out to be simple, easy, useful and fun. Complete tutorial with examples and command reference section.

Z80 Tutor I Fresh look at assembly language *$9.95*
Z80 Tutor II Programming tools, methods *$9.95*
Z80 Tutor III File handling, BCD math, etc. *$9.95*
Z80 Tutor X All Z80 instructions, flags *$12.95*
Common-sense assembly tutorial & reference for novice and expert alike. Over 80 routines. No kidding!

Add S & H. Call or write MICRODEX for details
1212 N. Sawtelle Tucson AZ 85716 602/326-3502

A Trip on the Star Princess

by Roy T. Beck

Recently, my wife and I splurged and took a one week cruise on the Star Princess, one of the P&O Line luxury type ships which take you on an eating binge, with a little gambling and sightseeing included. Our trip began and ended in Vancouver, BC, with stops at Juneau, Skagway, and Ketchikan and a visit to Glacier Bay, all under the general heading of an "inside passage" cruise.

The ship is only 5 years old, so the design is specific to cruising service, and is not a "conversion" of a ship designed for some other purposes. Other niceties include nine (9) electric elevators arranged in three banks of three each, one near the bow, one amidships and one near the stern. The official deck numbers ran from 1 to 14, with 13 being non-existent (the old hotel phobia of no 13th floor). Passengers were restricted to decks 2 to 14, which really gave us a lot of room to wander. The ship can cruise at 22.5 knots and displaces 63,500 tons.

Features included a large formal dining room in the stern which seated all 1500 passengers in two sittings. There was also a buffet type dining room on the 12th deck, at which you could optionally dine. Beside the usual breakfast, lunch and dinner, there was an afternoon "tea", and a midnight brunch. Wow!

There were four actual bars, plus alcoholic drinks at the dining areas. The top deck had two swimming pools and three hot tubs. Amidships, there was a fairly large atrium with shops to while away your wallet and purse. Everyone was issued a combination credit and ID card, with which you charged all extra expenses, such as drinks, goods bought in the shops, and photos taken by the ship's photographer. There was also a small library with novels available, a card room, a good sized motion picture theatre, and a live stage theater. Other odds and ends include a youth center, a teen center, exercise room, a pizza parlor, two casinos, and a wine bar. The crew totalled about 630 people.

Our quarters were a twin-bedded stateroom, with a gross floor plan of about 10' x 20', including the closet and bathroom spaces. Cable TV with some satellite signals plus films and some live TV generated on board provided in-room entertainment. Ships announcements were optionally available on one channel. This was of interest in announcements of activities available to the passengers. The principal activity was the 5 meals a day, all included, no extra charge except for alcoholic drinks. Some people actually ate at all five sittings! You should see their waist lines.... I gained only 4 pounds, and Barbara actually lost 1/2 pound! But then, we ate with moderation.

Because this trip was the last of the season, with winter fast approaching, the weather was a little dicey (for a cruise ship) at times. We did avoid the Pacific one day, because of predicted rough weather. This had an interesting side effect. Gambling had to be forbidden for that day, because we were in Alaskan waters instead of international waters, and Alaska does not permit gambling. (It's OK in Canadian waters).

One night we felt some spiral pitching and rolling due to rollers hitting us at an angle. I believe the propellers occasionally came out of water, based on my senses of touch and hearing. The ship being so high out of water, it was somewhat subject to wind loading, even though it was also equipped with stabilizer fins underwater amidships. Each fin was 2.55 meters fore and aft and extended 5.1 meters outward.

The ship was diesel electric powered, with four engines driving generators for a total of 52,000 HP, and two 16,000 HP motors driving the twin screws. For maneuvering, the ship has two bow thrusters which provide about 1200 HP each to move the bow left or right, and one stern thruster with about 1400 HP. The captain did all the maneuvering in port, and handled the ship quite expertly, in my humble opinion. Only in Ketchikan did he call for tugs, and apparently that is necessary there on account of the confined space. The other ships did the same. He didn't use the tugs, however, they just stood by, which was the case with the other ships. (Or it may all be a case of makework union rules....)

Being an engineer by vocation, I was quick to avail myself of an opportunity to visit the bridge while in port. That place looked like a refinery control room! There were two massive consoles running across the ship from side to side, with a small interruption in the middle of the forward one, where the steering wheel was located. Actually, the steering wheel was disappointing in its delicate size, being not much larger than a dinner plate in diameter. Of course the ship has "power steering", so no great physical force was required. At either end of the bridge, there is an extension which extends out over the side of the ship; this is used when docking, and since it extends about 10 feet beyond the side of the ship, the captain and his two officers had a clear view from bow to stern of the dock or whatever else is alongside. This extension is outdoors, so the duplicated compass, throttles, telegraphs, steering controls etc were all exposed to the elements, but were covered by hinged hoods when not actually in use.

Other features were huge display and control panels for operating and displaying the conditions of the ships air conditioning system, its water tight door system, its fire alarm system, and such other niceties as the trim system. There were two bow and two stern tanks, divided either side of the keel, which allowed trimming of the trip bow and stern and port and starboard, by shifting ballast water from tank to tank as required. Since diesel oil was burned at a considerable rate, the ship's trip had to be periodically adjusted for comfort.

Another interesting feature was that the ship's water distillation plant could provide up to 700 tons (max) per day of fresh water, but only while at sea. This indicates the distillers operated from waste heat boilers operating from engine exhaust heat. In port, the crew connected a hose to a special hydrant each day to maintain water supplies. The ship consumes about 600 tons per day (about 160,000 gallons) of fresh water, which is largely used for bathing, kitchens, and the ship's laundry.

Unfortunately (for communications), the captain and all his officers were Italian, and all orders and responses were therefore in italian. My italian is totally non-existent. The captain is fluent in English, and enjoyed giving commentary several times a day over the PA system to the passengers, telling us when and where, and pointing out some of the scenic attractions.

Other features were satellite position determining equipment, multiple radars, depth finders, communications type radio equipment, and an automatic logging system. Even so, a junior officer was always at the captain's side, logging with pencil and paper all commands and the time of their being given. The oldest word processing system still has a place in this world. Another curiousity; since we were almost always in coastal waters, we always had a local pilot on board, either US or Canadian. Somehow, I suspect the captain knew his way around well enough not to need a pilot, but rules are rules.

On our last night out, heading back to Vancouver, Barbara and I finished supper and went back to our cabin. I looked out the window (a real window, mind you, not just a porthole), and immediately realized the ship was doing a hard left turn. As I continued to watch, the ship came to a halt, but continued to turn on its own vertical access.

Realizing something unusual was taking place, I suggested we head for the 14th deck above the bridge. When we got up there, the captain and his coterie of officers were staring off toward Vancouver Island, a searchlight was operating off of each bridge extension, and some smaller boats in the vicinity were also using searchlights. Even though some of us on the 14th deck (the bridge was the 12th), were only 10 feet above the captain we could not be sure of his commands. About that time, the captain came on the PA system and announced we had stopped and were searching for a man overboard. He didn't (then) make it clear if the man was from our ship or some other. The Canadian Coast Guard was also involved.

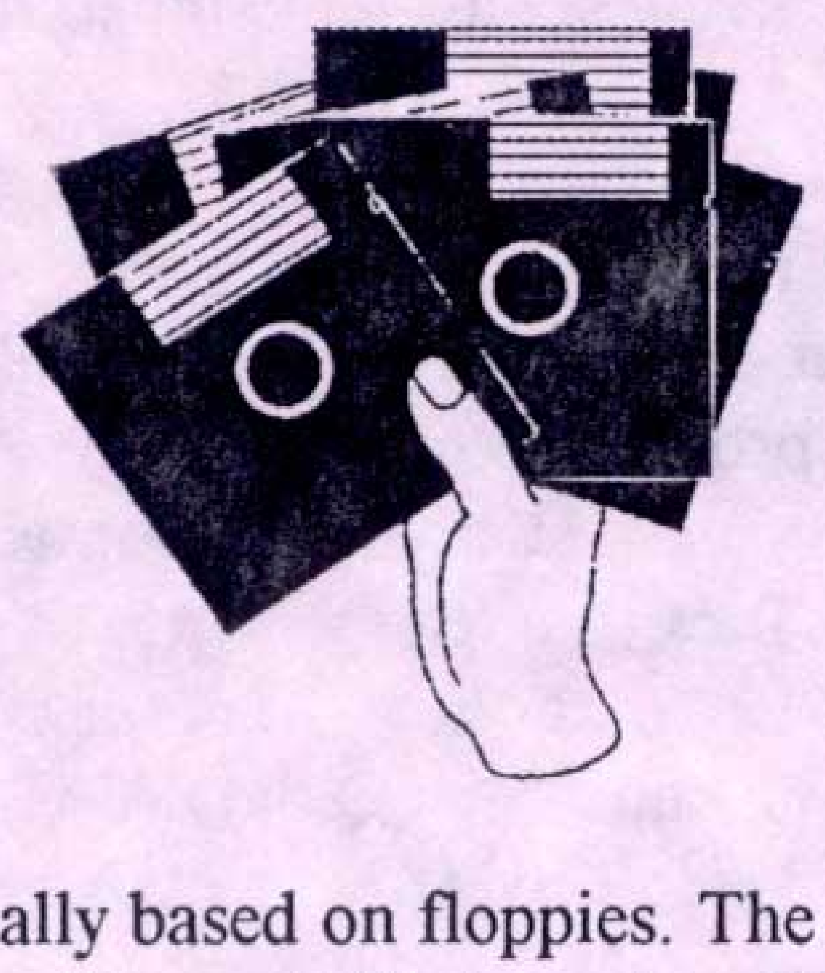
A little later, another announcement made it clear the man was a crewman of our ship, later identified as a Russian national who was a waiter in the dining room. No official reason was ever given for the man being overboard, but the rumor mill had several explanations. One was and attempt at suicide, But I discredit this entirely. My own belief is the man was trying to become a wetback escaping into Canada to set up a new life. The captain said he was suffering hypothermia when found, and we know he was found on the shore of Vancouver Island. He had to swim about a half mile, minimum to get that far from the ship. I am sure he had a lifejacket on, or he could not have survived to swim that far. Anyway, he was hoisted on board from our own life boat, and taken to sick bay. I was most impressed that he was so quickly found, at about 8 or 9 PM, on a dark night. I think the automatic logging system allowed the captain to return to the approximate location of the leap overboard, and that someone had to have seen him leap and reported the fact immediately. I am sure the leaper incurred the captain's wrath, at the least, as we spent about an hour recovering him, and resuming our trip towards Vancouver. It was interesting to watch the captain do a 180 degree turn while standing still in the water. Those bow and stern thrusters sure are neat.

Altogether, the trip itself was interesting and pleasant, and I learned a little about cruise ships. The ship is now in winter service, either along the California-Mexico coast or perhaps by now over in the Caribbean. Versatile.

And now back to attempting to understand and control that TRS-80 beast at the other end of this desk!

QuikDisk

A "QUICK" Review by Allen Jacobs



There is an old saying that goes: "A chain is only as strong as its weakest link." If there is any truth to that, then the weakest link in most personal computers is the floppy drive. That is especially true in the case of the majority of TRS-80's, because their permanent storage is either partially or totally based on floppies. The floppy drive in our systems is also the most likely part to fail. It is the slowest component and certainly the noisiest. Altogether, it's the most irritating part of any computer, "that we can't do without".

Of course, I would not complain this way about floppy drives unless I had, at least, a partial solution for these "floppy drive blues". I do. Or, more precisely, J.F.R. "Frank" Slinkman does. His solution is a program called "QuikDisk". It doesn't eliminate your floppy drive. It just makes it easier to live with. In the bargain, it may just make it last longer by reducing the amount of accessing it does. Even if it only makes your nerves last longer, it's worth it.

How does it work? Essentially QuikDisk is a floppy disk caching program that works on the TRS-80 in much the same way that SMARTDRIVE does on an MS-DOS PC. It uses the upper 64K on a 128K machine to store all the sectors on a disk that have been read by the floppy drive, to the capacity of the two high banks of RAM, which Frank works out to be 248 sectors. While I didn't work that figure out for myself, I'm sure that he did. So, we can take it as a given.

Once a sector has been read into RAM, it is subsequently read from RAM rather than from the floppy disk again if it is re-accessed. We all know that a data transfer from one RAM location to another is always faster and certainly less noisy than a transfer involving a floppy disk.

The same process is invoked to handle writes to the floppy. That way if you load an entire document into Allwrite for example to change a single occurrence of the word "East" to the word "West", and then decide to change it back to the word "East". The entire file would be read into RAM. The first change would be made. Then, only the one granule containing the changed sector would eventually be written out to the disk. If the entire file was then accessed again immediately thereafter, the disk drive wouldn't even turn on, yet the file would be almost instantaneously be reloaded. Then, when the second single change was saved, only that single granule containing the changed sector would eventually be written out to the floppy.

Too theoretical you say? Remind yourself of that thought after you have just bitten through the end of your pencil when you have to change the word "plint" to "print" in a long basic program only to discover, after saving it, that the title of your work which is entitled: "The BIG Program" was saved as "The BUG Program". With QuikDisk, the second load and save to correct this additional error would have been nearly instantaneous. Additionally, your floppy disk drives would not have moved. They would thus be happier because of the reduced wear. This means they would last longer.

"Disk based" database programs wherein records, keys, and other frequently accessed files that change very little or not at all, gain the most from "cached" access. They are effectively turned into "RAM based" database programs. This is true even if your database is larger than RAM memory; because, only a few records within the entire database are actually accessed or altered at any session. Those that are accessed are processed one at a time.

On the road, such high speed/reduced frequency disk access can literally free you from your hard drive..., even if you DON'T have one. How can 64K of RAM hold an entire 180K, 360K, or 720K disk worth of sectors? It can't. However, it does not have to do that in order to be effective. Unless you are backing up an entire disk, the entire disk is virtually never read nor written to at one time.

When you recall that the largest RAM based TRS-80 program can never be larger than about 48K in addition to DOS, its size can only be increased by means of overlays. Since the overlays must be smaller than this, a number of the most frequently used overlays can repeatedly be called from the QuikDisk cache, without those performance robbing floppy disk reads normally required to get at the overlays. The reason that overlays or the portion of some giant overlay read into memory must be SMALLER than memory is that you always need SOMETHING in memory to load and manage the overlays.

When disk activity finally does exceed the capacity of the cache what happens? Nothing, except that the floppy disk has to be read or written to as it normally would be. Everything gets updated: the RAM cache and the disk itself.

How does it work in practice? It appears to work flawlessly. All you have to do is to type:

```
QD (ON)
```

The QuikDisk logo appears and displays its status, at any change in its status. After a period of normal disk activity, your floppy drives start to work faster, more efficiently, and less often upon multiple reads of a file. Multiple floppy drives are automatically cached. Caching is totally automatic, depending upon drive/file use. It feels somewhat like using a hard disk. I can't imagine why such a utility has not been a part of LS-DOS. SMARTDRIVE is a part of MS-DOS.

What are the "negatives"? There are always some. However, in this case more accurately, they are limitations of the system and hardware or just practical constraints.

In order to assure that a floppy drive is "current" you must either access its directory, open or close a file, log the drive off of QuikDisk, or quit QuikDisk. This is important to prevent losing a portion of the disk within RAM. Disk directories are not cached. However reads and writes to a disk directory, as well as cache overflows cause QuikDisk to update the drive. The simplest and most dependable way to assure that a disk is current is to issue the command:

```
QD :d
```

where ":d" is the drive to update before removing the disk. QuikDisk then assumes that the disk in the drive is new.

QuikDisk is not recommended for hard drives, extended RAM disks, Memdisks, or other programs that must use the upper two RAM banks, large memory expansion boards, and XLR8er Boards. This is not truly a limitation because most of these items already have similar drivers and would not benefit from being cached anyway. QuikDisk is mostly "at home" on a "stock" 128K Model 4.

In order to copy or backup a floppy drive, QuikDisk should be disabled. Frank suggests putting commands of this nature into "DO" files that automatically disable QuikDisk during the command and re-establish it on exit. The command to disable QuikDisk is:

```
QD (OFF)
```

The QuikDisk logo appears and announces that it has been disabled while it is removed from memory. It is not always possible to remove QuikDisk if subsequent low memory drivers have been installed after it. The resident RAM driver uses some low memory so that care as to what other drivers or filters are present is important. It should also be loaded last if it will necessary to remove it later in a session.

If QuikDisk is left "ON" during a backup, erroneous results can occur on the backed up disk because a portion of the disk may not be written to the target disk.

QuikDisk can not be installed from the SYSGEN command for the same reason that no high bank memory driver can be installed from that command. What that reason is, however, frankly eludes me at the moment. I just remember that it's true.

Obviously, if QuikDisk is occupying the upper banks, another program that requires them will not be able to utilize them. This is not a serious problem because giving up the Area 2 and Area 3 commands in Allwrite allows caching MORE than two smaller text files through QuikDisk as long as the total size of the files is less than the limit of the cache. In the case of Memdisk, Frank states that QuikDisk is 50% faster and it is obviously easier to use. So why even bother with Memdisk anymore?

Interestingly, by the time I got down to writing "version one" of this review, Frank Slinkman had already come up with Version 2 of QuikDisk, which I used for the review. The documentation is better in Version 2 in that it is more complete. The Version 2 documentation does not specifically state in what other ways it is different from Version 1. Not to worry, it works so well that I am happy with it whatever version it is. When you have a new utility that is genuinely unique and helpful to your system and that flawlessly performs a task that no other utility you have accomplishes, you tend to worry less about what version it is as long as it does the job. QuikDisk DOES THE JOB!

Because of the cache's inherent characteristic of delayed writes, I was initially concerned that QuikDisk would fail to write something to a disk. NOT A PROBLEM! This phobia is as irrational as was my initial fear of using a slide rule. With or without QuikDisk, it is always important to wait until the drive light is out before changing a disk. Also, I realized that I have lost data more often due to other hardware and disk media failures than I have ever lost due to untimely removal of a floppy from a drive. Just developing the good habit of cleanly exiting any program allows me to use this valuable utility with assurance. If you are usually reasonably careful and systematic in your floppy disk handling, you like speed, and you hate disk drive accessing as much as I do, you are going to enjoy this utility.