Add a black, pointed hat....

CLOAD
MAGAZINE, inc
BOX 1267
GOLETA, CA
93017
OCTOBER 1980

About a month ago, a neighbor of ours brought us a large green "squash of some kind". Well, it is getting toward the end of October, and the back of that "squash" is getting rather orange. Oh, you didn't know that a squash had a back? Well, this one has a face etched in its still green front. We're kinda optimistic that by Ghoul's Day, our ugly duckling (squash) will turn into a pumpkin (sorry Cinderella). Then it will be a real Clyde-O-Lantern....

| Side | Title | Turns Count CTR-41 | Turns Count CTR-80 |
|---|---|---|---|
| **** | Block 8 Cover | 12 & 256 | 7 & 150 |
| ** ** | Atomic Table | 41 & 281 | 23 & 166 |
| ** ** | CIA Adventure | 102 & 326 | 59 & 192 |
| **** | Coder (Disk only) | 229 & 426 | 135 & 251 |
| ** | Tiar | 12 & 254 | 7 & 149 |
| *** | Tiar Notes | 72 & 300 | 42 & 177 |
| ** | Load 'N' Save | 134 & 348 | 79 & 205 |
| ** | Escape | 172 & 377 | 101 & 222 |
| **** | Maze (System Maze /) | 225 & 420 | 132 & 248 |

The Block 8 cover draws kaleidoscopic images on the screen. Actually there are three or six mirror images drawn (depending on how you look at it), but it looked like eight to me at first, hence the name. Maybe there are four images...nine...how about five.....

You can list all of the known elements with Atomic Table. Also, the program will break down a given formula into its basic elements. I wonder what Coca-Cola is made of...

Did you drop a ruby? Get the CIA on it immediately! In CIA Adventure, you give one or two word commands to direct the operation. Good luck in getting past the various obstacles in your quest to retrieve the ruby.

And we have just the thing for you CIA types (with a disk system) out there - Coder! This program takes a BASIC file from the disk and creates a carbon copy of that BASIC file, only the copy can't be listed to the screen. You can also decode your previously coded programs. But for those of you who want to decode Coder.... surprise!

Tiar is one of those programs that thoroughly frustrates any reasonably stable person. As you all know, programming a computer takes a marginal amount of logical thinking. Being able to break the problem you want to solve down into its basic components and then being able to sequentially step through each component to write executable code takes a logical mind. But at least you are given the rules (syntax structures) of the language you are programming in before you start. In Tiar, you are not told the rules. By logical deduction, you are supposed to figure out what those rules are. Your goal is to amass a certain number of points, but you are

not told how you can score those points. By logical deduction, you are supposed to figure out how to score points. You are given a certain set of commands, but you are not told what those commands do. By logical deduction, you are supposed to figure out what those commands do.

Had enough? If you are observant, you will have seen by now that following Tiar, there is a little program entitled Tiar Notes. By logical deduction (where have I heard that before?) you will have assumed that Tiar Notes contains the keys to unlocking Tiar. You're an optimist. It's true that you will find a few hints there, but don't expect any great revelations.

Careful! When you list Tiar, there doesn't seem to be anything logical about the end of line 170. Well, Mr. Richmond somehow stuck one of his infamous extra-long lines in there. The end of the line reads something like '....<>A(E2,F2,A(E2,F2,0))THEN@,'. By PEEKing around, the '@,' (which is not consistent since it sometimes lists as a graphic character or as a line feed) turned out to be '100'. Editing the line can be hazardous to the program.

Load 'N' Save is a potentially powerful utility. Say that you are playing a long game like CIA Adventure, and, after five hours of playing, you finally got inside the elevator. But now you have to quit because you are leaving for the weekend. You don't want to leave your computer on, but you would sure hate to start the game over. If you could save the values of all the variables onto tape, and then load them back into the computer on Monday, you could re-initialize CIA Adventure to the point where you left off. That is what Load 'N' Save allows you to do. Sounds too simple? There are catches (like it probably won't work with CIA Adventure in reality)...

1) Although Load 'N' Save is a BASIC program, it loads a machine language routine that must reside permanently in the top of 16K memory. This means that the program from which you want to save the variables can't be so big that it uses all of memory. This also means that you can't have another utility stored in upper memory at the same time.

2) You must know something about the structure of the program that you want to use Load 'N' Save on. For instance, if you stop the program in the middle of a subroutine, save the variables off, and then attempt to jump back into the program right where you left off (after loading the variables back in), you will get an RG (Return without Gosub) error. This happens because the location of the place you were supposed to return to was not saved with the rest of the variables. That location is a system variable located in an array type structure (called a 'stack' - more on that some other time) that only the system uses. In other words, you must have a good understanding of programming techniques in order to use Load 'N' Save on an existing program, or to be able to modify a program so that Load 'N' Save will work on it.

Be sure to CLOAD and RUN Load 'N' Save before loading in your BASIC program (from which you want to have the variables saved or loaded). Note to disk users: this program will not work with DOS BASIC.

Some of you early subscribers will recognize the game Escape (Chase in October 1978). If you enjoyed the original, you will probably love this version (it's FAST). For the uninitiated, Escape puts you in a room full of crazed robots that blindly seek to do you damage. There are also electrical fields scattered about the room which tend to give you that overdone feeling if you touch them. But those fields also tend to scrap robots. And if the robots touch each other, they get a little over-exuberant in shaking hands and a pile of rubbish results. It is a

kind of demolition derby in which you want to be the last one running.

This month's system offering will aMAZE you (alright, no more bad puns... this paragraph). Just to watch it create a maze is worthwhile enough. But then you get to find your way out of it. Believe it or not, there is a way out from any position inside the maze (test it - I did). All you have to do to load and run it is: 1) type 'SYSTEM'<enter>, 2) answer the ?* with 'MAZE'<enter> (Maze now loads), 3) answer the next ?* with '/'<enter> and you're ready to get lost in the maze. Has anyone seen the Minotaur lately?

Hey disk users! You can put 'Organ' (from last month's issue) on disk and run it from disk. Use LMOFFSET if you are using NEWDOS, or TAPEDISK (begin 7000, end 7700, entry 7000) if you are using TRSDOS, to take the program from tape and put it on disk. Add the '/CMD' extension to the file name for convenience (ie: ORGAN/CMD). To make it sound good, you must disable the clock interrupts. So before running Organ, you must go into BASIC and type 'CMD"T"'. If you are using NEWDOS, you can then type CMD"ORGAN" to run. In TRSDOS, you must then type CMD"S" to get back to the DOS level. Then you type 'ORGAN' to run. Kudos to George P. Saladino Jr. for pointing this out.

How many of you got an 'NF IN 980' error from Shopping Spree in last month's issue? Raise your hands. A majority of you from the look of things. That means that the first item that you bought was not an item from your list and then you tried to check out early. Don't ask me how long it took to find exactly when the error occured, let alone how long it took to find what caused it. The culprit was line 460, listed below:

```
460 FORZ=1TO100:X=PEEK(15100):R#=R#+1:
    IFX=0THENNEXTELSEIFX=1THEN800ELSE480
```

Those of you who know BASIC, after studying this line a while, will come to the conclusion that this line is syntactically correct. Yet it causes an error in line 980 -

```
980 FORZY=1TO500:NEXTZY:FORZZ=1TO50:OUT255,0:OUT255,2:NEXTZZ:NEXTJ
```

After breaking apart line 980 into separate lines, the 'NEXTJ' turned out to be baddie. But line 940 had FOR J=1 TO B and no other statement caused a jump to a line between 940 and 980. So the 'NEXTJ' should have worked. Now the 'J' was taken off the 'NEXTJ', and when the 'NEXT' was hit, the program jumped to line 460 (not 940 as it should have had things been working correctly).

If you look again at line 460, the only FOR loop is 'FORZ=1TO100'. Well, at least we now know what caused the error - the FOR Z attempted a match with the NEXT J. But, there is a matching 'NEXT' for the FOR Z in line 460. Then why did the attempted match occur in the first place?

School's in! Clyde Cload's Institute for Higher Misunderstanding brings you a quick and mostly incomprehensible course on the BASIC interpreter.

Let's start at ground zero - the machine. This machine can't read Latin, English, or BASIC. It can only read and act upon a group of switches (8 to a group in this machine) which are either on or off. Since each group of switches can only be arranged on or off in a finite number of ways, we can associate numbers to each arrangement. For human convenience, base 8 (octal), base 16 (hex), or base 10 (decimal) is used. When programs are in this form, they are said to be written in 'machine code' or 'machine language' (imagine that!).

BASIC is not a machine language. It is a human language. For the machine to run a BASIC program, it must first be translated into machine code (which may take more than one translation, but that is another story). There are essentially two methods of translation:

1) Compiling. The program you wrote in BASIC is run through another program called a 'compiler'. This compiler matches FORs with NEXTs, GOSUBs with RETURNs, GOTOs with the correct lines, etc. It does a dozen (million) other things, then spits out another complete program. You would then run this 'new' program, which is the machine language translation of your original program. Our little machines do not have a compiler, although I suppose you can buy one for the Model I TRS-80 somewhere. The compiled machine code program will run FAST (really fast!), but it can take 10 minutes or more to compile some programs. So to make one little change in a program could take a long time since you would have to re-compile the program afterwards.

2) Interpreting. This is what our machine does. The 'interpreter' looks at the first line in your BASIC program and translates it. If the line is completely executable (ie: I=2) it executes it. If it is not fully executable, it does what it can and sets up pointers and tables for later. The tables and pointers are used when the line becomes totally executable (ie: FOR X = 1 TO 100 can't be completely executed until the NEXT X is found). After translating the first line, the interpreter goes on to the next line, and so forth.

Since interpreting code is relatively slow, the interpreter in the TRS-80 is evidently set up to do as little as possible. For instance, if I=0 and the line 'IF I>0 THEN GOTO 100' is interpreted, the 'THEN GOTO 100' part of the line won't even be looked at since the 'IF I>0' is false. Now let us look at line 460 in Shopping Spree.

The matching 'NEXT' for the 'FORZ=1TO100' is located after an 'IFX=0'. Since we know that an error occurs when the first item we pick up is not a valid item and we then check out early, we can assume that the following happens (without really knowing what the program does at this point):

Line 460 is interpreted. The 'FORZ=1TO100' is done first and since it is not totally executable, table entries and pointers are set up in anticipation of the 'NEXT'. Next, the value of X is computed (it is not zero or one) in 'X=PEEK(15100). Then R# is incremented by one in 'R#=R#+1'. Now the 'IFX=0' is evaluated and is found false. The 'THENNEXT' is ignored (aha!), the 'ELSEIFX=1' is found false, and the 'THEN800' is ignored. Finally, 'ELSE480' is done and we are moved up to the checkout counter.

We decide to checkout and the receipt is being printed out. Line 980 is now being interpreted. The 'FORZY=1TO500: NEXTZY' is executed completely. The 'FORZZ=1TO50:OUT255,0: OUT255,2:NEXTZZ' also goes by without a hitch. Then 'NEXTJ' is interpreted and associated with the 'FORZ=1TO100' in line 460 since the 'NEXT' in line 460 was just skipped over. And an error results.

What really happened? The interpreter got confused! The 'NEXTJ' should have been associated with the FOR in line 940. I really don't know why the interpreter blew it, but imbedding a NEXT statement in an IF-THEN-ELSE construction (as in line 460) scrambled something. If somebody out there is in the know, I would appreciate enlightenment! All the same, this is a perfect example of how fancy code writing can cause fancy trouble. Moral: Do it simply!