Newsletter

April's Newsletter

The Winnipeg 99/4 Users Group is a non-profit organization created for users by users of Texas Instruments 99/4A Home Computers and compatibles. The content of this publication doesn't necessarily represent the view of this user group.

Next General Meeting - Date : May 1, 1986
Time : 7:00 P.M.
Place: Winnipeg Centennial Library
2nd Floor, Auditorium

Executive 1986:

President:	Jim Bainard	334-5987
Treasurer:	Bill Quinn	837-7758
Newsletter Editor, Book and User Programs Librarian:	Mike Swiridenko	772-8565
Contributing Editor:	Paul Degner	586-6889
Newsletter Publisher:	Hank Derkson	
Inter-Group Representative:	Dave Wood	895-7067
Systems Co-Ordinator:	Rick Lumsden	253-0794
Educational Co-Ordinator:	Sheldon Itscovich	633-0835
Public Domain Librarian: 822 Henderson Hwy.	Gordon Richards	668-4804
Module Librarian:	Peter Gould	889-5505

Mailing Address:

NEWSLETTER EDITER
WINNIPEG 99/4 USERS GROUP
P.O.B. 1715
WINNIPEG, MANITOBA
CANADA R3C 2Z6

BBS #: (204)-889-1432 SYSOP: Charles Carlson
HOURS: REOPENING IN THE NEAR FUTURE

EDITORIAL COMMENTS:

Mike Swiridenko, Newsletter Editor, is busy writing university finals so Paul Degner will slip in as Newsletter Editor for this issue.

MISCELLANIA:

Miscellaneous news and reminders.

The FEDATA NETWORK provided us information on their FEDATA INFORMATION NETWORK B.B.S. containing a online catalog of documents, reports, handbooks, surveys, and books on and from the United States Government. They have given our user group a free I.D. and password to use their system. If you are interested in trying out this B.B.S then please contact Paul Degner.

Last meeting the tape of the Chicago Faire was shown. Bill Quinn brought his portable VCR. Thanks Bill!

Our hardware *people* were busy last month! Sheldon Itscovich, a proud father of a bouncing baby widget, showed off his four slot darling. He says he can produce these at forty dollars a pop. Steve Zabarylo walked in with a alternative widget, two modules full of GROM games, and the home brew 32K expansion neatly attached to his Corcomp 9900 system. It's truly impressive what TIers can do on winter evenings!

READER RESPONSE:

The following was attained from Charles Carlson. The author is unknown.

Subject: 99/4A P-Box Power Supply Modification

This modification should be performed only by competent electronic technicians.

This modification was performed to allow, first; the console to run cooler, and second; to "beef" up the power supply so it could supply enough power to run two disk drives in the P-Box.

This modification involves removing the exsisting voltage regulators in the P-Box and replacing them higher capacitv units. Also three more voltage regulators will be added as to supply power to the console. Holes will have to be drilled in the rear of the P-Box, to mount the voltage regulators. This is necessary to provide heatsinking for the regulators.

You will need to purchase the following parts:

2-7812HK 12V 5A TO-3 regulator chips.
2-7805HK 5V 5A TO-3 regulator chips.
1-7905T -5V 1A TO-220 regulator chip.
5-0.47 uF/35V tatalum capacitors.
5-2.2 uF/35V tatalum capacitors.
1-4,700 uF/35V electrolytic capacitor.
1-2,200 ohm 1/2 watt resistor.
2-MR501 diodes.
4-TO-3 transistor mounting sockets.
1-set of TO-220 insulating mounting hardware.
1-male and female 15 pin "d" connectors with hoods.
an assortment of colored wire, 22 gauge is ideal.

1. Remove the top cover to the P-Box. Remove the flex-cable P-Box interface card, memory and RS-232 cards and any other cards you may have except the disk controller.
2. Remove the disk drive (two screws on top, two screws on bottom).
3. Remove the disk drive data cable from the disk controller.
4. Remove the disk controller card.
5. Remove the power switch knob (pull it off).
6. Remove the front cabinet from the P-Box (1 screw on left side, 1 screw on right side, top 4 screw on rear, 1 screw beside fuse, 1 screw behind disk drive, 3 screws on right bottom side, 1 bottom front screw, 2 bottom left screws, and one bottom screw about 2 inches back from where the PEB card mounts).
7. Pull the cabinet foward and set it aside. You will see the power supply section on the left side (power transformer, in front of blower, and PC board to the left of the transformer). At this point, you may wish to install a different blower, such as a high quality boxer fan in place of the blower. I replaced mine and it is much quiter and it pulls more air.
8. The PC board is mounted via a plastic holder that angles to the right on the bottom. There are two screws, one at each end of the holder. Loosen them, but do not remove them. There are two connectors that lock on to the PC board on the rear. Push their tabs thru the board and pull the connectors off. There is also another connector on the front. Remove it the same way. Then slide out the PC board.
9. Locate the voltage regulator ICs(in the middle and center rear). Remove the regulators ICs.
10. Solder three different color wires appx 6 inches long to the holes of the voltage regulator. Do this again for the other regulator.
11. Locate D3 and D4. Note the polarity of the diodes. Solder two diodes to the transformer side of the existing diodes. Be sure that the cathodes go towards the transformer side. Solder the anode ends together. Solder a wire appx 8 inches long, to the ends you just soldered on the diodes. Set the PC board aside.
12. Drill the TO-3 mounting holes on the rear of the expansion box. I mounted mine beside the blower, going downward for three of the regulators and mounted the remaining below the blower above the fuse holder. You will have to take a razor blade and cut off part of the serial number tag. You need to do this so the chassis can be used for an heatsink. After drilling be sure to file the holes smooth.
13. Install the 0.47 uf capacitors to the inputs of the voltage regulator (pin 1) and install the 2.2uF capacitor to the outputs (pin 2). Solder the negative side to the chassis/ground lug on the mounting socket(this is only for the TO-3 devices).
14. Mount the P-Box 5V and 12V regulators. Use a good grade of silicone heatsinking compound when you install the regulators.

15. Solder the wires from the circuit board that went to the regulator that use to be on the board. Observe correct wiring.

16. Install the other two TO-3 regulators using heatsinking compound. On the 12V unit jump a wire from the 12V regulators input you just installed. Also jumper a ground wire. Install the capacitors, as before. Do the 5V regulator the same way. Solder a wire appx 15 inches long to the outputs of the second set of TO-3 regulators. Route these wires to go out the card cage, thru an unused P-Box slot.

17. Install the TO-220 -5 Volt regulator. Be sure to use insulating hardware. Solder a wire to the output terminal and solder 2 wires to the ground terminal, appx 15 inches long. Route the wires out, as above with exception to one of the ground wires. Jumper it to a ground on one of the TO-3 regulators. Add the bypass capacitors as above but note the polarity. Solder another wire to the input of the regulator, appx 8 inches long. Route this wire to the front of the P-box near the power switch.

18. Install the PC board back in place. The wire from the rectifiers you installed needs to be soldered to the 4700 uF filter cap. The wire that came from the negative voltage regulator should be solder to the same point. Install a ground wire from the filter cap to any ground point on the PC Board. Again observe polarity.

19. At this point, the box should be wired. Check for proper wiring. Turn on power to the P-Box. Very quickly check the outputs of all regulators for proper operation (one +12 and +5 for the drive, one +12 and +5 and -5 for the computer). If all voltages are OK, then procede with the next step. If you do not a have a voltage check the input to the regulator and work it back. Check for shorts and proper wiring.

20. On the wires that you routed out of the P-Box, you will need to indentify them and solder on a 15 pin "d" connector, female jack. Use a 15 pin "D" so you will have no problems with hookup(the joysticks and cassette ports are 9 pins). I solder pin 7 to +12, +5 to pin 5, -5 to pin 3 and ground to pin 1.

21. Put your P-Box back together. Bring your system up with the computer to make sure its OK. Test a disk drive.

CONSOLE MODIFICATION.

Check the wiring on your 15 pin "d" plug. You need to construct another 15 pin "d" connector male with 4 wires appx 15-30 inches long. After making the cable, hook it up to your system and turn the P-Box on. Use a voltmeter to confirm wiring and mark each wire as to what voltage they are. Hookup is EXTREMELY sensitive at this point. If you wire a power supply to the wrong power buss in the 99/4A computer, I may take its last dying gasp and smoke. There will be no way to repair the damage as it will be extensive. Semiconductors do not like reverse voltages. They usually conduct very heavy and burn. I cannot overstress this point!

1. Open the 99/4A console by removing all the bottom screws. You will see the power supply beside the keyboard and below the computer's PC board. You will see four wires connecting the computer to the existing power supply. With the power supply board exposed, turn on the computer and check the voltages where the wires connect to the power supply. Mark these carefully. Turn off the computer. Connect the power cable from the P-Box to these wires, again noting correct hook up (+12 to +12, +5 to +5, -5 to -5, ground to ground). Remove the 4A's power supply. Keep it for the future.

2. Route the cable outside the existing hole where the power plug use to connect.

3. Take the big step. Hook the computer up to the P Box and the system. Bring the system up. All should be well. If you do not get your title screen shut down, and check connections and pray that you did not burn up anything. You should have no problems a point if you maintained correct wiring.

4. If alls well, shut down the system. I left the original power switch in the 4A computer. I superglued it in the on position. You may wish to take it out and install a system reset switch (I highly recommend as if the system locks up you will be powering down your whole system. You can also use a Widget). Put your case back on and happy computing.

The best benefit is the lack of heat from the console. Normally, when I use my computer I am on for about two hours. Its strange not to feel any heat coming from the cartridge port area. Also I put two half height drives in the P-Box and I have had no problems. If you use the parts I recommended, you will not be able to destroy the power supply. They are internally limited and the P-Box power transformer will burn up before the regulators will. Also, just to relieve your worries, the P-Box can handle a heavy load. It was designed to operate 8 cards.

Here is some checks I did for current draw on the computer.

+12V draws 240 ma
+5V draws 940 ma
-5V draws 132 ma

If you have any questons, feel free to contact me.

There are two schematics for this article called before and after. Please pardon the word processor graphics, but if you can read a schematic you should not have any problem with these. I recommend that you study them before you perform the modifications, so you will understand exactly what is going on. Also take your time and do the job right.

REVIEWS:

This column presents reviews of materials that may be of interest to the user. The views expressed are the opinions of the reviewers, exclusively.

SOFTWARE:

c99 A review by Paul Degner

How can a person really review a computer language? Basically it is only as good as what you can do with it. I will attempt to review this language to the best of my knowledge.

A few month's ago I came across a advertisement in a CIM-99 newsletter for a subset C compiler. Not knowing much about the language I sent off my disk/mailer/money for this tryware package called c99 developed by Clint Pulley. Two weeks later the package arrived. I read the documents, tried the example programs, and filed it away for future use. This is usually the way I tackle any new significant software arrivals.

In the package was a note from Clint that he had just finished a library of graphic functions, a random number generator, and a text formatter written in c99. What he said sounded interesting so I sent another disk/mailer/money to Clint. Another wait of two weeks and I recieved what I wanted. This time I was ready to learn the language. In order to do so I had to have

some kind of learning aid so I decided to drop over to the local book shop and purchase a C book such as Jack J. Purdum's C Programming Guide.

It took a weekend to read Purdum's book from cover to cover which made me very eager to compare my subset C compiler to the various ones described in the book. Looking at the specifications sheet of c99 I found a somewhat complete subset C compiler as compared with Purdum's ideas of what a subset C compiler should be.

Missing in c99 V1.32 were the objects (float, double, enum, and void), data types (multi-dimensional arrays, function returns other than integers, structures, and unions), identifiers (underscore), storage classes (automatic, static, external, and register), attributes, assignment operators (+=, -=, *=, /=, %=, >>=, &=, |=, and ^=), statement keywords (for, switch, case, default, do-while, and sizeof), macro preprocessor and control lines (#undef and #if), and all parametized macros. Not much eh? Clint says he will have these options installed in further releases of c99.

To use this language to do something is another task indeed! The best configuration is to install the c99 compiler, TI assembler, libraries, refs, and a program image disk cataloger such as SD on a drive one diskette while keeping drive two diskette allocated for c99/EDASM source and object files. Next is to develop a program. I eventually decided on a conversion from BASIC to c99 of a program which appeared in a Byte magazine dealing with encoding and decoding of text files using passwords. It took me about two weeks of my spare time to perfect because of the time it takes to get your c99 source to 9900 object code.

I was always hoping someday there would be a good language compiler available for the 99/4A and I think I finally found it. Clint Pulley has put a lot of his time and effort into developing c99 and he asks if you think he has to send him twenty dollars. I think it isn't enough for what he has put into c99 but I can only abide by his wishes and he also guarantees notice of any new updates of c99 to the user.

For more information on c99 write to:

Clint Pulley
38 Townsend Avenue
Burlington, Ontario
Canada L7T 1Y6

416/639-8583
STC TI7395

HELPFUL HINTS AND TIPS!
(FOR THE USERS, BY THE USERS!)

This column feature tips brought to my attention from members of this group, other user group's newsletters, and various other sources. *WARNING:* These hints and tips are to be used at your own risk!

MULTIPLAN:

The following is reprinted from the 99'ER ONLINE February newsletter.

A BUG IN TI MULTIPLAN by Bob Chapman

I have been working on a project to get the old family Grandfather Clock back into operation and ran into a spot of bother with the pendulum length - it wasn't quite long enough to give a beat of one second (ie, a period of two seconds). How long should the effective length be?

Out came the high school physics book and a few minutes research revealed the formula:

T=2*PI*SQRT(L/G)

where T is the period, L is the effective length, and G is gravitational accelleration (32.174 feet/second/second).

Then I moved over to my TI and loaded up MULTIPLAN and set up two columns, one for L (in inches), the other for T. The formula in MULTIPLANESE is:

2*()*SQRT(RC[-1]/12/32.174)

where RC[-1] picks up the value for L. I varied L from 12 to 48 inches and took a look at the results.

Something was wrong; between 38 and 39 inches, the period jumped by a factor of ten! Now, I knew that the length should be about 40 inches, which is what MULTIPLAN was indicating. But for lengths under 38 inches, the results were obviously whacko. On analysis, I found that at this and shorter lengths MULTIPLAN must calculate the Square Root of numbers smaller than .01 but it gets them wrong - by a factor of ten. For example, it tells me the SQRT of .09 is 3 when it should be 0.3.

I tried changing the formula by using an exponential of 0.5 and got the same results. Next I tried it in basic and got the correct answers. On an IBM-PC, using LOTUS 123, I got the correct answers.

Does anyone know why TI MULTIPLAN cannot find the correct roots of number smaller than 0.01? I don't know the answer but I would like to hear your theories or facts!

c99:

```
/* c99 encoding program
** This program was originally developed by ralph roberts which appeared in the
** april 82 issue of byte. The program is really pretty simple--but the code it
** generates is not. The encryption begins as a simple offset. The program
** first reads your password (or passphrase) and sums the ascii values of all
** the letters and spaces. To obtain the offset, the program divides the sum by
** the number of letters and spaces in the password. With an offset of 63, for
** example, every letter is printed 63 characters higher than it actually is
** (with a wrap-around feature to maintain the desired ascii range of 32 to 123
** and an upward shift of one so no space will be printed). In each succeeding
** line, the offset is increased by one. This prevents anyone from breaking
```

```
** your code by analyzing frequency of character appearance. Every single
** letter and space is represented by a different character in every line.
** programmed in the basic language, it has since been translated to c99 by:
**
** Paul Degner
** 1105 Church Avenue
** Winnipeg, Manitoba
** Canada R2X-1G1
**
** (204)-586-6889
*/
#include "DS:1.STDIO"
#define CHARMAX 81
char fn[16],
     inbuf[CHARMAX],
     outbuf[CHARMAX],
     password[CHARMAX];
int ascii,
    ch,
    i,
    inp1,
    inp2,
    leninbuf,
    lenpw,
    offset,
    valpw;
main()
{      while(1)
       { while(1)
         { while(1)
           { puts("\n\n\nThis program was originally developed");
             puts("\nby Ralph Roberts which appeared in the");
             puts("\nApril 82 issue of BYTE as a BASIC");
             puts("\nprogram. Recently this program has been");
             puts("\nconverted to Clint Pulley's c99");
             puts("\nlanguage. This program is designated as");
             puts("\ntryware by :");
             puts("\n              Paul Degner");
             puts("\n              1105 Church Avenue");
             puts("\n              Winnipeg, Manitoba");
             puts("\n              Canada R2X 1G1");
             puts("\n\n\nThe Encoding Program");
             puts("\n==================");
             puts("\n\nPASSWORD <80 max chars> : ");
             gets(password);
             if(!*password) continue;
             else break;
           }
           lenpw = width(password);
           i = 0;
           valpw = 0;
           while(i < lenpw)
           { valpw = valpw + password[i];
             i++;
           }
           offset = valpw / lenpw;
           if(offset < 123) break;
           puts("\n\nSorry PASSWORD phrase too big!\n\n");
         }
         while(1)
         { puts("\nENCODING or DECODING (1 or 2) :");
           ch=getchar();
           if(ch == 49)
           { inp1=getfn("be ENCODED","r");
             break;
           }
           if(ch == 50)
           { inp1=getfn("be DECODED","r");
             offset = offset * (-1);
             break;
           }
         }
         inp2=getfn("receive 'doctored' output","w");
         while(fgets(inbuf,CHARMAX,inp1))
         { leninbuf = width(inbuf);
           i = 0;
           while(i < leninbuf);
           { ascii = inbuf[i];
```

```
            ascii = ascii + offset;
            if(ch == 49)
              if(ascii > 123)
                ascii = (ascii - 123) + 32;
            if(ch == 50)
              if(ascii < 32)
                ascii = (ascii + 123) - 32;
            outbuf[i] = ascii;
            i++;
          }
          outbuf[i] = NULL;
          if(ch == 49)
          { offset = offset + 1;
            if(offset == 124)
              offset = 1;
          }
          if(ch == 50)
          { offset = offset - 1;
            if(offset == (-124))
              offset = (-1);
          }
          puts("\n");
          puts(inbuf);
          puts("\n");
          puts(outbuf);
          fputs(outbuf,inp2);
        }
        fclose(inp1);
        fclose(inp2);
        puts("\n\nContinue (Yy/Nn)? : ");
        ch = getchar();
        if(ch == 89)
          continue;
        else
          if(ch == 121)
            continue;
          else break;
      }
}
width(s) char *s;
{     int i;
      i = 0;
      while(*s++)
      i++;
      return(i);
}
getfn(text,a) char *text,*a;
{     int unit;
      unit = 0;
      while(1)
      { puts("\n\nFilename to ");
        puts(text);
        puts(" : ");
        gets(fn);
        if(unit = fopen(fn,a)) break;
        puts("bad filename-try again\n");
      }
      return(unit);
}
```

There's a good reason why we call it the TI Home Computer.

Because it's for every family member in your home. To help children grow in education and imagination. And to help adults keep growing with improved skills and knowledge.

What makes it so versatile is power — the 16-bit processing power of expensive computers. Combined with a superior amount of usable memory, the TI Home Computer is exceptionally easy to use, since it takes over more work for you than competing 8-bit models.

TI's simplicity gets children into computing — and learning — right away. They don't have to push as many keys as with other brands.

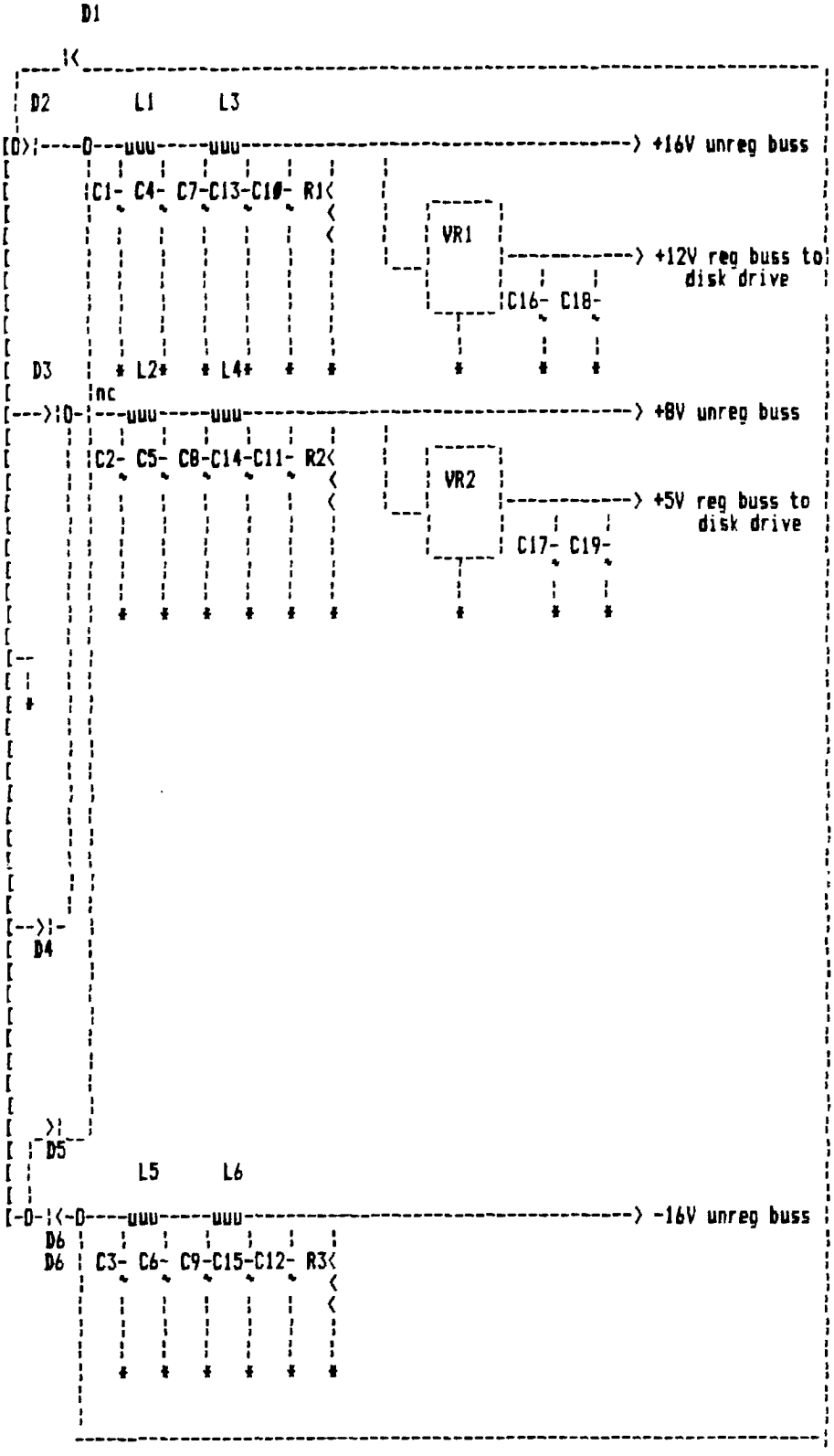
"Him? Computer repairman!"

December 1983 © Creative Computing

LOGO:

The following is reprinted from CIM-99 November newsletter.

```
TO DESSIN
TELL TURTLE CS
HOME MAKE "A 0 MAKE "B 90
REPEAT 36 [HOME RT :A FD 30 RT :B FD 50 MAKE "A :A + 5 MAKE "B :B - 5 ] MAKE "B 90
REPEAT 36 [HOME RT :A FD 30 RT :B FD 50 MAKE "A :A + 5 MAKE "B :B - 5 ] HOME WAIT 300 CS MAKE "A 0
REPEAT 72 [HOME RT :A FD 20 RT 60 FD 34 MAKE "A :A + 5 ] WAIT 300 CS
HOME MAKE "A 90 RT 90
REPEAT 61 [FD 45 HOME RT :A MAKE "A :A - 6 ] WAIT 300 CS
HOME MAKE "A 0
HOME REPEAT 90 [FD :A LT 90 MAKE "A :A + 1 ] WAIT 300 CS
HOME REPEAT 10 [FD 2 RT 90 FD 100 LT 90 FD 2 LT 90 FD 100 RT 90 ] WAIT 300 CS
MAKE "A 40
HOME REPEAT 40 [REPEAT 9 [FD :A LT 40 ] MAKE "A :A - 1 ]
END
```



[= transformer
->|-- = diode
uuu = inductor

|
- = capicitor
|

* = ground

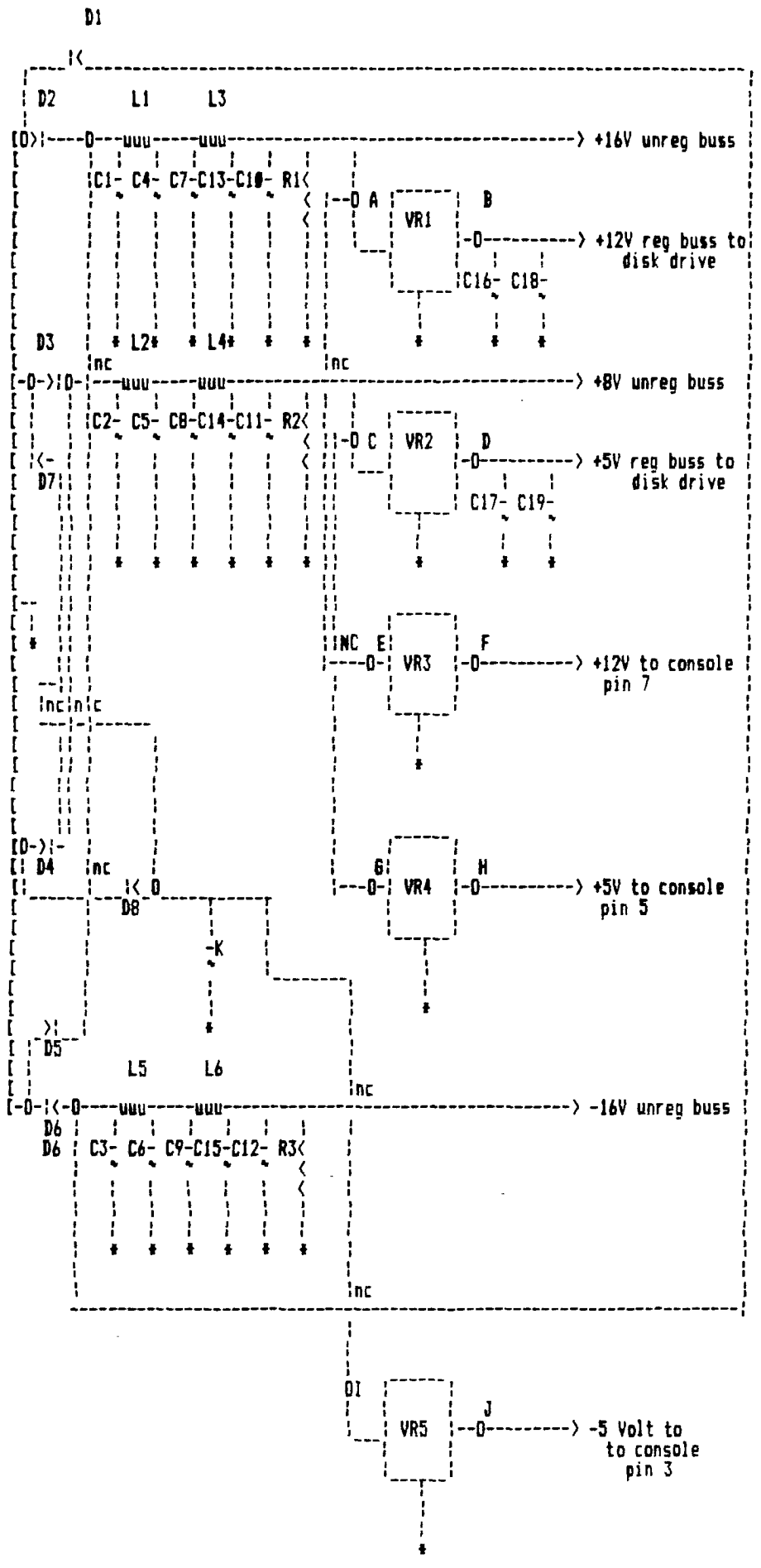
< = resistor

0 = connection
nc = no connection

VR1=7812K
VR2=7805T

Do-It-Yourself Instructions

1. Empty parts on clean workspace.
2. Wire components according to schematic diagram.
3. Plug in and turn on.
4. Write programs.



[= transformer
->|-- = diode
uuu = inductor

| = capicitor

* = ground

< = resistor

0 = connection
nc = no connection

Change VR1 to 7812HK
Change VR2 to 7805HK
add a .47 uF/35V tatalum capacitor to ground at points A,C,E,G,I right at case of regulator
add a 2.2 uF/35V tatalum capacitor to ground at points B,D,F,H,J right at case of regulator
add VR3, 7812HK
add VR4, LM323K
add VR5, 7905T
add D7,D8 MR501
add 4700 uF/35 volt capacitor at point K
mount all VRs to case for a heatsink
! remember polarity on negative 5 volt bus to computer, and capacitors, and insulate VR5 from chassis.
Use a 15 pin "d" connector to supply power to console (don't use a 9 pin, as to avoid confusion with tape and joystick ports).

Article: **SOFTWARE DESIGN**
Author: M. Swiridenko

What is software design?

Software design is the organized method of writing and testing computer programs. Programs written in an unorganized fashion tend to be hard to understand, and difficult to debug. Software design helps the programmer to write programs that will be easier to understand, and get running. Because the programs will be easier to understand, they will be easier to change.

The first step of designing a program is to write down, in plain English, a description of what it is you want your program to do. Writing down a program descrition not only makes clear, in your own mind, what the program is all about, it states for you an objective which you can work towards. Also should you decide to change some aspect of your program you may alter it, easily, by changing the descrition before you actually start writing the program statements. Once you have the program objectives clearly in your mind, you may procede to the next steps of Software Design.

The next steps of program design include writing program 'psuedo-code' and actual program statements, followed by testing and debugging of the program.

Before I get involved further with these steps I will discuss the basic structure that all programs have. All programs usually consist of three parts:

INPUT
PROCESSING
OUTPUT

Each of these parts don't necessarily follow in the order stated, and may be inter-mixed through-out a program. A program will consist of controlling code, and code which perform processing operations. The processing code can be further divided into independent functions, or subroutines. Controlling code directs how processing is to proceed, and thusly determines (by means of logical comparisons) when particular subroutines are to perform their task. Subroutines may be further broken down into controlling code and other processing tasks. These subtasks of a subroutine may be divided into other parts, and so on. The breaking down of a program into controlling code, and independently executable processing code illustrates the hierarchical structure of programs. That is programs can be divided into small parts which are controlled from routines which call those parts. These routines may then be called from other controlling routines, and so on, for as many levels as are needed. A controlling routine which calls a subroutine is said to be a level above the subroutine. A routine which calls the controlling routine is said to be two levels above the subroutine, and so on.

Because programs can be written as subroutines which call subroutines, you may write your program in small parts rather than all at once. Software design methods take advantage of this fact.

There are two basic Software Design techniques that are may applied when you create your program. These are Top-Down design or programming, and Bottom-Up design or programming.

Top-Down design involves the writing of the controlling code before writing the processing subroutines. This means that the higher level routines will be written before the lower level routines. Bottom-Down design is the exact reverse. In Bottom-Down design the lower level routines are written before the higher level controlling routines. Design and testing of programs may use both of these techniques.

There are several advantages/disadvantages to each method. Top-down code is easier to test and get running quickly. Bottom-Up written code can be tested only a subroutine at a time, and often requires special controlling routines (called drivers) for testing purposes. In the Top-Down approach, overall program testing is possible since dummy routines may be substituted for lower level routines. In Bottom-Up programming you may test the low level routines individually, but must wait for the higher level controlling routines to be written before the entire program may be tested. If you want to see your program working before you have finished all of the programming you will use the Top-Down approach. If you want to spend less time adding and testing subroutines you will use the Bottom-Up approach to programming.

Now that you have the choice Bottom-Up or Top-Down designing your program code you will want to start writing your code. A usefull design technique that can help, at this point, is the use of what is known a 'Psuedo-code'. Psuedo-code are descriptive English-like statements that represent program code. Psuedo-code is not restricted by the demands of syntax that normal program code is. English-like wording and lack of syntax restrictiveness make Psuedo-code easy for the programmer to work with and understand. Psuedo-code, because it represents program code is also easily translatable into working program code. The following is an example of some Psuedo-code.

```
display mailing label title
open the mailing file #1.
loop for 1 to 25 while reading and printing a mailing label.
close the mailing file #1.
display the end of program message.
quit the program.
```

The above Psuedo-code is easily translated into the following Basic program.

```
100 PRINT "MAILING LABELS PROGRAM"
110 OPEN #1:"DSK1.MAILLIST",INPUT,INTERNAL,VARIABLE 80
120 FOR I=1 TO 25
130 INPUT #1:NAME$,ADDR$,CITY$,PROV$,PC$
140 PRINT NAME$:ADDR$:CITY$:PROV$:PC$
150 NEXT I
160 PRINT :"END OF MAILING LIST PROGRAM"
170 END
```

To summarize and end what I have said:
Software Design techniques methods take advantage of the hierarchical structure of programs, and facilitates programming by the use of Psuedo-code, and the methods of Top-Down and Bottom-Up programming and testing. More will be said on the subject of program testing and debugging in a future article.

PROGRAMMING HELP FILE:

The purpose of this column is to present, to the user, techniques that will be useful in the writing of programs for the TI-99/4A home computer. I hope that there is something, in what follows, for everyone. If you can provide some programming insight that might be useful to someone, please, feel free to pass it on to me and I'll get it into the next newsletter.

BASIC/EX-BASIC:

A part of BASIC programming that many beginners find difficult to understand is the ARRAY. This month's XBASIC/BASIC help-file discussion will try to explain what ARRAYS are all about, and how this very usefull fundamental data construct can be used.

ARRAYS are basicly lists of related data values. An ARRAY may be a list of number values or a list of character strings.

An example of a number list is a price list for products in a hardware store. If the items in the store are numbered from 1 to 20 then there would be 20 prices in your list, and in your program you would have to DIMENSION your array for 20 numbers. Your DIMENSION statement would look like this:

```
100 DIM  PRICES(20)
```

The default DIMENSION, in TI X/Basic, is 10. If you need more array space than the default of 10 items, you have to use a DIM statement to have the BASIC interpreter reserve more memory. You may also DIMENSION an array, for less then 10 elements, to save memory. An array of one element (DIM ARRAY(1)) is like a simple variable except that it requires an index.

Back to the price list again. If you numbered your products 1 to 20 you would be able to find the price of product numbered 13 as follows:

```
110 PRINT PRICES(13)
```

To change a price, say the item numbered 18, from whatever it was before plus a 6% increase, due to inflation, you would do the following:

```
120 PRICES(18)=PRICES(18)*1.06
```

As you can see an numeric ARRAY differs from an ordinary numeric variable in that it is immediately followed by a set of parenthises which enclose a number. The parenthises indicate that the variable is a list of numbers rather than the single value that a variable is. The number within the parenthises is called the 'index' of the array. The index is used to indicate which element of the array list is to be used. The index may be a numeric variable. Having a variable as an index makes your array more versatile to use. You can now assign values to each element of an ARRAY variable with a FOR-NEXT loop. To get the prices for your 20 hardware items you would do the following:

```
200 REM READ PRICES INTO THE ARRAY.
210 FOR INDEX=1 TO 20
220 READ PRICES(INDEX)
230 NEXT INDEX
235 REM 20 PRICES FOR THE PRICE LIST
240 DATA 1.29,0.99,5.99,1.44,3.89
250 DATA 1.34,9.98,8.69,1.59,1.75
260 DATA 20.35,89.99,45.57,31.13,63.71
270 DATA 13.24,0.59,77.21,345.65,11.25
280 REM END OF PRICES.
```

To see the prices in your array you would also use a loop and a variable index to print the prices to the screen. The following loop will print the prices in tabular form.

```
300 REM PRINT PRICES TO THE SCREEN
310 PRINT "ITEM";TAB(8);"PRICE"
320 FOR INDEX=1 TO 20
330 PRINT INDEX;TAB(8);PRICES(INDEX)
340 NEXT INDEX
350 PRINT "END OF PRICE LIST"
```

Character arrays are also available. An example of a character array is a list, by name, of all 20 products that the hardware store handles. Lets call this array PROD$. You would use indexes to change, read or print items in this list. A DIMENSION satement would also be needed as there would be 20 items in this list. An example of changing a name is:

```
400 PROD$(20)="HAMMER"
```

A revised price list program follows. It will display the name of the items as well as their prices.

```
500 REM PRICE LIST WITH PRODUCT NAMES INCLUDED
505 PRINT "ITEM";TAB(6);"NAME";TAB(23);"PRICE"
510 FOR INDEX=1 TO 20
520 PRINT INDEX;TAB(6);PROD$(INDEX);TAB(23);PRICES(INDEX)
530 NEXT INDEX
540 PRINT "END OF PRICE LIST"
```

If the hardware store decides to expand its product line and now has 30 products rather than the previous 20 items, we would have to change our DIM statements and the loop limits to reflect the increased memory need. If we had used a variable for our loop limits then this change would be simple. Since we did not we will have to change the limits for all of the FOR-NEXT loops in our program(s).

The arrays, PRICES and PROD$, are exaples of single dimensioned arrays. Arrays may be multi-dimensioned. An example of a 2 dimensioned array is a multiplication table. A 10x10 multiplication table is a 10 by 10 array. The dimension statement for such a table would look like this:

```
100 DIM TABLE(10,10)
```

To get at a value in this table you must supply two index values. TABLE(5,5) will index the fifth element of the fifth row, or the value 25, in our table. The TV screen is 32 columns by 24 rows long, and can also be regarded as a two-dimensional table. Multi-dimensioned arrays must always have their indexes seperated by commas. Arrays of more than two dimensions can be thought of simple as lists or pages of two-dimensional tables. Multi-dimensioned character arrays are also possible.

This ends my discussion of arrays. I hope that it has shed a bit of light on the subject. Bye till next time.

ASSEMBLY:

This month I will discuss memory addressing. As this is the heart of what assembly language is all about, I can only hope to cover a small part of it in this short space. What I hope to convey is the sence of how assembly language, and the underlying machine code function.

The first thing that must be understood about how the computer operates is that any program that it executes must operate in a limited amount of memory. The memory that a computer can work with is called its address space. The 9900 CPU has an address space of 64K bytes. Now, 'What does that mean?', you ask. From a previous discusion you know that the computer works with 1's and 0's, bits. These bits are grouped into sets of eight, called bytes, and pairs of bytes called words. The 9900 has 15 lines with which it may address words of memory. Each line may have a + or 0 voltage. Because an address line can have two values and there are 15 of these lines, the 9900 can work with 2^15 words, or 2^16 bytes of memory. Now, 2^16 = 65536 bytes, and 2^10 = 1024, or 1K in computer lingo, so 2^16 = 64x2^10 or 64K.

So now you know that the 9900 can address 64K bytes of memory, but where are these bytes kept and how are they arranged. The 64K address space of the 9900 is composed of ROM, and RAM. The bytes of ROM, and RAM are arranged one after the other in a sequential fashion. Sequential means that byte 16 follows byte 15 and byte 15 follows byte 14, and so on. Thus each byte of memory can be numbered from 0 to 65536. The zero'th byte is addressed when all of the address lines carry zero voltages. Different combinations of voltages on the address lines will access each of computer's memory locations (words).

How does the computer know where a program is in its memory? The computer, when you turn it on, will look for a program starting at memory location zero. This is the first byte of the computer's memory space. The computer will then follow the instructions that it finds at that location. In the TI console location zero is in pre-programmed ROM which contains the console's operating sytem. Because the operating system is at location zero, the operating system will take over control of the computer. The first thing the operating system will do is to perform several checks to see what accesories are attached, and if a cartridge is plugged in. The operating system will then transfer control to the appropriate software, as selected by you the user.

How does a program get to be executed by the computer? The 9900 is kept going by a clock (a quartz crystal), and with a certain number of clock cycles it will fetch an instruction from memory, and execute it. The 9900 will then get the next instruction from memory based on the value of the 9900's address register. The address register is refered to as the program counter or PC. The PC is set to zero when the power switch is turned on so the first instruction that the computer will get is from memory address zero. As an instruction is executed the PC will change value. If the instruction currently executing is two bytes long then the value of the PC will be increased by two, so that it will point to the instruction immediately after the current one. If the instruction was a jump to another memory location the PC will take on the value of the new memory location, and the 9900 will fetch and execute its next instruction from the new location in the PC. This is basically how the computer functions. It gets an instruction, executes it, then gets the next instruction. If for some reason there is no recognizable instruction at the next PC location the computer will stop or 'Lock Up'. A correctly written program will never intentionally 'lock up', and should restore the PC value so that control goes back to the operating system.

When you load and run an assembly program the loader program (in ROM) will put the start address of your program into the PC, and the CPU will fetch the first instruction in your program and execute it. The computer will then be under the control of your program.

How do machine instructions make the CPU do things? The 9900 has 69 machine instructions. Each instruction is represented by a different set of bit patterns. The CPU was built so that it will do different (predefined) operations depending on the bit patterns that are loaded into its instruction register. The instruction register is internal to the CPU and is where an instruction is kept while it is being examined and executed.

The Assembler will allow a programmer to format instructions using easy to remember names rather than the tedious to use bit patterns that the CPU understands. The assembler will change your descriptive English-like assembly mnemonics into machine readable code.

Where is data kept in memory? Data is often kept before the start of a program, or after the last instruction of a program. If data is mixed with the instructions of a program unpredicatable results may occur unless precautions are taken to 'jump' around those data areas. Data may also be stored within a machine instruction. Types of data that can be stored within an instruction are addresses to a data area (or areas), an address (or addresses) to an instruction (or instructions), or even a numeric data value. The type and number of data values that may be within an instruction depend on the format of the instruction.

There eight types of instruction formats. These instruction formats are:

1. Register Direct 2. Register Indirect 3. Register Inderect Autoincrement 4. Memory Direct/"symbolic"
5. Memory Indexed 6. Immediate 7. PC-Relative 8. CRU/Single-bit/Multi-bit

The first five are refered to as the general addressing modes (GAS), and are the most commonly used of the eight.

That is all space will allow this month. I will continue with addressing formats, and how they look in assembler mnemonics, next month. Till then, happy programming!

FORTH:

Rather than discuss a particular aspect of the Forth language I will present some of my own Forth programming attempts. Also if the readers have any particular questions about the Forth language I will try to answer them in this column.

The following screens work with the Hi-Resolution graphics mode, of the 9918A video chip, that is accessable from TI-Forth. The first screen is a Forth version of the Mini-Memory's 'LINES' demo program. The second screen is a word which draws a circle.

To run these screens you must load the TI-Forth options -TEXT, -GRAPH, and -GRAPH2. -GRAPH and -GRAPH2 loads the Forth words that let you access Hi-Res or 'GRAPHICS2' mode, and also has the words to plot points and draw lines in this mode. -TEXT loads the word that will return you from the Hi-Res mode back to normal 'TEXT' mode. After

loading in these options you can then enter 'scr# LOAD' for the appropriate screen and it will run automatically.

After the lines demo is running pressing 'REDO' will quit the current drawing and start another one. Pressing 'CLEAR' will quit the demo and return you to text mode. Any other key will pause the demo. The circle demo merely plots a circle on the screen, waits for a key press, then returns to text mode. No checking is done for a circle exceeding the edges ofthe screen. With a little modification the 'CIRCLE' word could become a general purpose circle drawer.

If you're looking for a quick and dirty way to access and learn about the TI's Hi-Res graphics capabilities then Forth seems to be a good answer. Have fun! I did'

NEWSLETTER EDITER
WINNIPEG 99/4 USERS GROUP
P.O.B. 1715
WINNIPEG, MANITOBA
CANADA, R3C 2Z6

```
SCR #26
 0 ( LINES DEMO PRGM.) : WAIT 20000 0 DO LOOP ; : NEG -1 * ;
 1 0 VARIABLE X0 0 VARIABLE Y0 0 VARIABLE X1 0 VARIABLE Y1
 2 0 VARIABLE XD0 0 VARIABLE YD0 0 VARIABLE XD1 0 VARIABLE YD1
 3 RANDOMIZE : RZ 4 RND 1+ ; : ND RZ 2 RND IF NEG RZ THEN RZ NEG ;
 4 : SDR ND YD0 ! YD1 ! ND XD0 ! XD1 ! ; : RY 40 - RND 2* + ;
 5 : SET SDR 256 RY X0 ! 256 RY X1 ! 192 RY Y0 ! 192 RY Y1 ! ;
 6 : MVENDS X0 @ XD0 @ + DUP 0< SWAP 255 > OR IF XD0 @ NEG XD0 !
 7     THEN Y0 @ YD0 @ + DUP 0< SWAP 191 > OR IF YD0 @ NEG YD0 !
 8     THEN X1 @ XD1 @ + DUP 0< SWAP 255 > OR IF XD1 @ NEG XD1 !
 9     THEN Y1 @ YD1 @ + DUP 0< SWAP 191 > OR IF YD1 @ NEG YD1 !
10     THEN X0 @ XD0 @ + X0 ! Y0 @ YD0 @ + Y0 ! X1 @ XD1 @ + X1 !
11     Y1 @ YD1 @ + Y1 ! ;
12 : DOLINE X0 @ Y0 @ X1 @ Y1 @ LINE ; : HCLR 8192 6144 0 VFILL ;
13 : DOLINES 100 0 DO MVENDS DOLINE ?KEY -DUP IF 6 = IF LEAVE ELSE
14 PAUSE IF TEXT ABORT THEN THEN THEN LOOP ?KEY 0= IF WAIT THEN ;
15 : LINES GRAPHICS2 BEGIN SET DOLINES WAIT HCLR AGAIN ; LINES


SCR #27
 0 ( CIRCLE DRAW PROGRAM)
 1 0 VARIABLE A 0 VARIABLE B 0 VARIABLE RADIUS 0 VARIABLE PH
 2 0 VARIABLE X1 0 VARIABLE Y1 0 VARIABLE PX 0 VARIABLE PY
 3 : INITD 160 80 50 0 0 Y1 ! PH ! DUP X1 ! RADIUS ! B ! A ! ;
 4 : CIRCLE BEGIN  X1 @ Y1 @ < 0= WHILE
 5   PH @ Y1 @ DUP + + 1+ PY ! PY @ X1 @ DUP + - 1+ PX !
 6   A @ X1 @ + B @ Y1 @ + DOT  A @ Y1 @ + B @ X1 @ + DOT
 7   A @ X1 @ - B @ Y1 @ + DOT  A @ Y1 @ - B @ X1 @ + DOT
 8   A @ X1 @ + B @ Y1 @ - DOT  A @ Y1 @ + B @ X1 @ - DOT
 9   A @ X1 @ - B @ Y1 @ - DOT  A @ Y1 @ - B @ X1 @ - DOT
10   PY @ PH ! Y1 @ 1+ Y1 !
11   PX @ ABS PY @ ABS < IF PX @ PH ! X1 @ 1 - X1 ! THEN REPEAT ;
12 : CTEST GRAPHICS2 INITD CIRCLE KEY DROP TEXT ;
13
14   CTEST
15
```

CURIOSITIES AND PASTIMES

This month's BRAIN TWISTER: **FOOTBALL RESULTS**

Near the close of the football season a correspondent informed me that when he was returning from Glasgow after the international match between Scotland and England the following table caught his eye in a newspaper:

	Played	Won	Lost	Drawn	Goals For	Against	Points
Scotland....	3	3	0	0	7	1	6
England	3	1	1	1	2	3	3
Wales	3	1	1	1	3	3	3
Ireland	3	0	3	0	1	6	0

As he knew, of course, that Scotland had beaten England by 3--0, it struck him that it might be possible to find the scores in the other five matches from the table. In this he succeeded. Can you discover from it how many goals were won, drawn, or lost by each side in every match?

For Sale:

HAM-teletype. 110 BAUD. DATA 100, -8 level. $20.00.
Phone: Benny Crooks. Home: 885-3644 or Work: 895-5791.