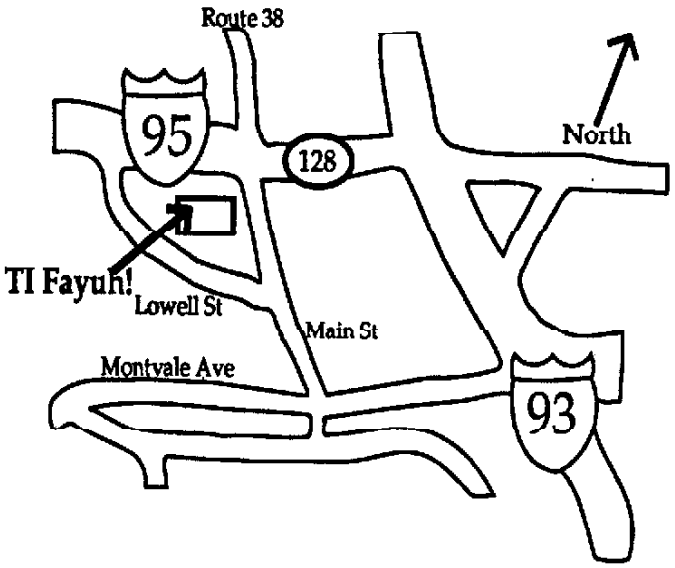
The Boston Computer Society TI-99/4A User Group Meeting Newsletter March 1989

Edited by J. Peter Hoddie

The TI Fayuh 1989!

The Fourth Annual New England TI Fayuh will be held on Saturday April 1 from 10AM to 6PM. The location is the Ramanda Hotel in Woburn Massachusetts. It is located directly off route 128 (I-95) at exit 35. Below is a map for your convenience. The official address of the hotel is 15 Middlesex Canal Park Road in Woburn. Their number is 935-8760. I might add here that it is quite close to my home town (Lexington) which is where previous shows have been held. As a point of reference the Showcase cinemas are located right next to the hotel, and the movie listing are clearly visible from the highway (if you are coming from the south it is a great marker, if you're coming from the north and you see the sign, you went too far - sorry!).

In any case, be at the show. It should once again prove to be an interesting and informative glimpse into the TI world. We are expecting the usual compliment of speakers and vendors. The official list is not yet available however. We have several speakers lined up, but are still looking for more. We also need some volunteers in the morning to set up. If you can help out, be there around 8AM. We will also need a few systems, or even pieces of systems if you can bring them. There will be a sell/swap table so bring your unwanted or unused hardware and legitimate software. Your junk may just be someone elses treasure. We also need some individuals to run the front door (i.e. admission) and a person or five to help out running the various BCS tables. Please let me (Peter) or Justin know what you can do soon so we don't have a last minute panic.



There are still vendor and user group spaces available at the show. If you are interested or know of someone who is interested, give me a call to make the arrangements. There is some official dealer/user group paper work for you to fill out.

If you have questions about the Fayuh or whatever, as usual, my number is (617) 375-6003. Don't call early in the morning, or after about 11 PM. And since I am often out, please be kind to my roommates, none of whom are BCS members or much care about TI computers. As usual the BCS office number is 367-8080.

The April Meeting

The April Meeting of The Boston Computer Society's TI-99/4A User Group will be held on Wednesday April 19 (the real Patriot's day... [spoken like a true Lexington resident!]) at the Massachusetts College of Art on Huntington Avenue in Boston. We should be on the fifth floor somewhere - you can always ask the friendly and helpful guard downstairs!

As always, the April meeting will be an opportunity to discuss the successes and problems with the annual Fayuh. As has been the case for the past few months, I hope to be able to show Press, the much anticipated word processor from Charles Earl. Unfortunately, recent signals from Asgard (the distributor of Press) have pointed to further delays. Paul Charlton has recently announced an extensive development package for the 9640 including an assembler, linker, librarian, and the long awaited technical manual on MDOS. Paul hopes to have these completed in April, so at least a preview of Paul's amazing new development system should be possible. The assembler is several times faster than the TI assembler, as is the linker, and as such this package should interest C programmers as well as assembly language programmers.

This Newsletter

Because I had no time last month to prepare a meeting newsletter, this month the newsletter is a bit larger than normal. There are the monthly columns from Donald Mahler and Ron Williams in addition to an extensive article on printer graphics by Joe Rawlins. The BCS recently recieved three more issues of Jim Peterson's popular Tips from the Tigercub, on disk. I have included the first new issue this month. Others will appear in future issues, space permitting. There is also this rather esoteric article that I wrote somewhere back there. Let me know what you think....

c COLUMN

By Donald L.Mahler

This month we are again working with c99 for the 99/4A rather than the 9640. Unfortunately the floating point functions have not been worked out for the 9640 so Clint and Tom Bentley have not been able to modify the floating point routines written earlier to run on the 99/4A. To use these programs, you need the floating point routines found in our Library and the Math Functions written by Charles Kirkwood for Micropendium. I have not yet placed these in our Library; have to check with Peter and Walt regarding copyrights. *[we have permission to reprint selections from MICROpendium in this newsletter, but I don't know how this applies to the software library -jph]*

```
/* EXPONENT TEST 1 =et2_c */
#define LEN 8
#include dsk2.floati
   /*Bentley */
#include dsk2.mathfn
   /*Kirkwood */
main()
{float base[LEN],ex[LEN];       /*floating
point numbers */
 float ans[LEN]; char buf[5];
  char *c; char op[2];
  char s[LEN];          /* string */
puts(" Enter the base:    ");
fpget(s,base); putchar('\n');
puts(" Enter the exponent:    ");
fpget(s,ex);  putchar('\n');
init();                         /*from mathfn
*/
ax(base,ex,ans);
puts("\n\n");
puts("The answer is ");
fpput(ans,s); }
```

The Henderson-Hesselbach equation is used in chemistry and biology to determine pH when you know the concentration of the acid and the salt; for example, the concentration of carbonic acid and bicarbonate in blood.

```
/*H-H EQUATION            */
#define LEN 8
#include dsk2.floati
#include dsk2.init
/* we are loading individual funct rather
than mathfn */
#include dsk2.natlog
char *fa="0.4343";   /*to convert ln to lg */
main()
{float
pH[LEN],pKa[LEN],acid[LEN],a1[LEN],a2[LEN],a3[LEN];
 float salt[LEN]; char buf[5];
  char *c; char op[2];
  char s[LEN];float k[LEN];
puts("       The H-H equation states:\n");
puts("     pH = pKa + log ([salt] /
[acid])\n\n");
puts(" Enter the pKa:    ");
/* this is a constant for each acid and can
be found in tables */
/* for carbonic acid, it is 6.1 at body temp
*/
fpget(s,pKa); putchar('\n');
puts(" Enter the salt concentration:   ");
fpget(s,salt);  putchar('\n');
 puts("Enter the acid concentration:     ");
fpget(s,acid); putchar('\n');
texp(salt,"/",acid,a1);  /*division */
stof(fa,k);  /* string to fp */
init();
ln(a1,a2);
fexp(a2,"*",k,a3);  /*converts nat log to
common log */
fexp(a3,"+",pKa,pH);
locate(19,7);
puts("The pH is ");
fpput(pH,s);
puts("\n \n\n");}
```

As a check, if the ratio of bicarbonate to carbonic acid is the normal 20/1, the pH is the normal 7.40.

Maybe next month back to c99 for the 9640.

Introduction to the UCSD P-System Using String Data Types

By Ron Williams

Many of the same functions you can use in Extended Basic are available for use in Pascal and are just as convenient to use as in Extended Basic. The string type is used a lot in UCSD Pascal and can be defined in the type section or in the variable section. When you declare a string you can define the string like this:

```
VAR
  word : string[10];
```

This string has a max length of 10 characters and if you try to give the variable a longer length it will chop off the extra characters on the end. The string type also has a default length and that is 80 characters and a string can have a length as short as 1 character and a max length of 255. The procedure readln should be used to import strings in to a program but it may be necessary to make a string another way because using string types is much easier than using a packed array of characters. One other way to make a string is to convert a packed array of characters to a string and that is done like this:

```
word:='          ';
for count:=1 to 10 do
   word[count]:=multichar[count];
```

The variable word is of type string and the variable multichar is a packed array of characters. You must give word a length before assigning the variable one character at a time or you will get an execution error that is why I put the assignment statement before the loop. Another way to give word a new length would be to use a compiler directive to shut off range checking and then assign the new length directly like this:

```
(*$R-*)
word[0]:=chr(10);
(*$R^*)
for count:=1 to 10 do
  word[count]:=multichar[count];
```

This method may be used if you are not sure of the new length of the string and how long to set the loop for. The chr function can except a variable as well as the constant "10" for input.

Some string procedures and functions included in UCSD Pascal are explained below:

The function concat will put together a number of strings and each string must be put in the function separated by commas like this:

```
CONCAT('hello',' how',' are you');
```
It will return:hello how are you

The function copy returns a string of characters starting with a position and a size the function could be used like this:

```
COPY('hello how are you',7,3);
```
It will return:how

The function pos returns a integer value it returns the starting position of a string within a source string. It is used like this:

```
POS('hello how are you','are');
```
It will return:11 This is the first place in the string that 'are' is found.

The function length will return a length of the string it returns an integer value it is used like this:

```
LENGTH('hello how are you');
```
It will return 17 an integer

The procedure delete will remove characters from a string it will remove the characters starting with position and ending with a size. The procedure is used like this:

```
DELETE('hello how are you',1,6);
```
It will remove 'hello' from the string

```
line:='hello how are you';
DELETE(line,1,6);
WRITE(line);   (* will return 'how are you'
*)
```

The procedure insert does just the opposite as delete it will insert a string into a source string it is used like this:

```
line:='hello how are you';
INSERT(line,',hello',6);
WRITE(line)    (* will return 'hello,hello
how are you' *)
```

It will insert the string ',hello' into the variable "line".

The next string functions that I will show you are not found as part of UCSD Pascal but have been added to the user library of the TI 99/4A the additional functions are in the library misc so put uses misc as a library at the beginning of your program.

The function break will return the position of the first character in the source string that matches a character in the second string it will be used like this:

```
thepos:=BREAK('hello','0');
```
The variable thepos will now have the value 5 the string 'o' is the 5th character in 'hello'

The function span will return the first position of a character not found in the source sting it is used like this:

```
thepos:=SPAN('hello','h');
```
The value of 2 will be put into the variable thepos because 'e' is the first character not found.

The function uppercase will return uppercase letters in a new string with the source string having lowercse letters.

```
line:='hello how are you';
UPPER_CASE(line,line2);
WRITE(line2);  (* will return 'HOW ARE YOU'
*)
```

The function will convert the lower case letters in line to upper case and put the string in line2 the source string is not changed and so a new string is created with upper case letters. One more point I would like to make, is make sure you use READLN and not READ with strings the results when using READ can be very bad, things like run time errors and other problems can develop. Well thats it for this month, thanks.

Displaying the IBM Graphic Elements

By Joe T. Rawlins

Several months ago I needed to make an attendance chart for the Sunday School class that I co-teach with my wife. I proceeded to draw one on craft paper only to find that I had skipped a few dates and misspelled a name. I did this twice and decided that there had to be a better way.

I use an IBM PS/2 model 50 at work, and have created

numerous charts and blank forms using the graphic elements contained in the IBM Character Set 1. These characters are located from character 176 thru character 223, and are printable with an IBM compatible printer. These printers usually have DIP switches to select either Standard (Epson) or IBM emulation. When the Standard character set is selected these characters represent Italic characters, however when the IBM mode is selected they represent the predefined line graphics. Most printer manual contain a table of the various character sets available for that printer.

Printing the graphic elements was just a matter of using the transliterate command on various characters below 127, since Funnelweb only recognizes characters in the range of 0 to 127. I tried to pick characters that I thought I would not use in any text that may have to go on a chart. I also tried to use the paired keys ([],(),{},<>) for the corner combinations. There are 45 graphic elements in the range I am covering, however I have only used 19 of them.

This worked fine for printing, but looked a little strange on the screen and you could not be sure how it would look until you ran it thru the formatter. Why not convert the characters that were transliterated to display what they would become? I have changed a few characters in a CHARA1 file with a sector editor, which is OK for a few characters, but to change 19 or so seemed like a lot of work (I like to draw the characters on graph paper and then figure the hex code for each row). I subscribe to GENIAL TRAVelER (everyone should) and in Volume 1 Number 5 there is a program by Wayne Stith called KWIKFONT, which is an all-assembly character definition program. With this program I was able to redefine the characters that I selected, to display the graphic elements desired. After redefining the characters, the ones that have been changed may be saved to a DV-80 file.

Once this is done you have two choices, to use another program by Wayne Stith called KF->CHARA1. or use the information in the file to modify an existing CHARA1 file. I have done both. Wayne's program creates a 9 sector CHARA1 file on DSK1, from the saved DV-80 file and whatever CHARA1 is in memory at that time. I have found this file to be workable from Funnelweb even though it originally came with a 5 sector one. It is possible to use a sector editor to change this from a 9 to a 5 sector file, however that is another article. If you already have a CHARA1 file that has modified characters that you want to keep then the second option is what you want.

The November 1988 issue of MICROpendium contained an article and source code for converting your favorite CHARA1 file into source code DATA statements for editing. The article is by John Birdwell and the source code as published was bug free. What you get is a Charaset file that you may edit with the data obtained from the KWIKFONT DV-80 file. Once the data statements have been modified, this source code is then Assembled to get E/A 3 code. This is then loaded into the Assembler along with the SAVE utility. Execute the SAVE and save the file produced as CHARA1. I saved mine as CHARA2 since I would only be using the graphic elements with the word wrap disabled and in 80 columns (Funnelweb's Program Editor).

The following is the source code for the Charaset that I created. I am including it for those who do not have KWIKFONT and the November Issue of MICROpendium.

```
0001          DEF  SLOAD,SFIRST,SLAST
0002  SLOAD
0003  SFIRST
0004      DATA >0020,>0000,>1824,>2418   * char=>00
0005      DATA >0020,>0008,>1808,>081C   * char=>01
0006      DATA >0020,>0010,>2408,>103C   * char=>02
0007      DATA >0020,>0018,>2408,>2418   * char=>03
0008      DATA >0020,>0014,>141C,>0404   * char=>04
0009      DATA >0020,>001C,>1018,>0418   * char=>05
0010      DATA >0020,>0008,>1038,>2418   * char=>06
0011      DATA >0020,>001C,>0408,>1010   * char=>07
0012      DATA >0020,>0018,>2418,>2418   * char=>08
0013      DATA >0020,>0018,>241C,>0408   * char=>09
0014      DATA >2020,>3800,>1C10,>1C10   * char=>0A
0015      DATA >0040,>0020,>2038,>2438   * char=>0B
0016      DATA >0070,>5070,>4854,>1C14   * char=>0C
0017      DATA >0070,>4070,>001C,>1010   * char=>0D
0018      DATA >0020,>0018,>243C,>2018   * char=>0E
0019      DATA >0040,>0814,>101C,>1010   * char=>0F
0020      DATA >0040,>4040,>1824,>2418   * char=>10
0021      DATA >0020,>2020,>2808,>0808   * char=>11
0022      DATA >0040,>4058,>2408,>103C   * char=>12
0023      DATA >0040,>4058,>2408,>2418   * char=>13
0024      DATA >0040,>4054,>141C,>0404   * char=>14
0025      DATA >0040,>405C,>1018,>0418   * char=>15
0026      DATA >0040,>4048,>1038,>2418   * char=>16
0027      DATA >0040,>405C,>0408,>1010   * char=>17
0028      DATA >0040,>4058,>2418,>2418   * char=>18
0029      DATA >0040,>4058,>241C,>0408   * char=>19
0030      DATA >0040,>4040,>1824,>3C24   * char=>1A
0031      DATA >0040,>4050,>101C,>141C   * char=>1B
0032      DATA >0040,>4040,>1C10,>101C   * char=>1C
0033      DATA >0040,>4444,>041C,>141C   * char=>1D
0034      DATA >0070,>7070,>7070,>7070   * char=>1E
0035      DATA >0040,>4C50,>101C,>1010   * char=>1F
0036      DATA >0000,>0000,>0000,>0000   * char=>20
0037      DATA >0010,>1010,>1000,>1000   * char=>21 !
0038      DATA >0028,>2828,>0000,>0000   * char=>22 "
0039      DATA >0808,>08FF,>00FF,>0000   * char=>23 #
0040      DATA >0808,>0808,>FF00,>0000   * char=>24 $
0041      DATA >0044,>4C18,>3064,>4400   * char=>25 %
0042      DATA >0020,>5020,>5448,>3400   * char=>26 &
0043      DATA >0008,>1020,>0000,>0000   * char=>27 '
0044      DATA >0000,>0000,>0F08,>0808   * char=>28 (
0045      DATA >0000,>0000,>F808,>0808   * char=>29 )
0046      DATA >0044,>287C,>2844,>0000   * char=>2A *
0047      DATA >0808,>0808,>FF08,>0808   * char=>2B +
0048      DATA >0000,>0000,>0030,>1020   * char=>2C ,
0049      DATA >0000,>0000,>FF00,>0000   * char=>2D -
0050      DATA >0000,>0000,>0030,>3000   * char=>2E .
0051      DATA >1414,>1414,>F414,>1414   * char=>2F /
0052      DATA >003C,>4C54,>6444,>3800   * char=>30 0
0053      DATA >0010,>3010,>1010,>3800   * char=>31 1
0054      DATA >0038,>4408,>1020,>7C00   * char=>32 2
0055      DATA >0038,>4418,>0444,>3800   * char=>33 3
0056      DATA >0008,>1828,>487C,>0800   * char=>34 4
0057      DATA >0078,>4078,>0444,>3800   * char=>35 5
0058      DATA >0038,>4078,>4444,>3800   * char=>36 6
```

```
0059      DATA >007C,>0408,>1020,>2000   * char=>37 7
0060      DATA >0038,>4438,>4444,>3800   * char=>38 8
0061      DATA >0038,>4444,>3C04,>7800   * char=>39 9
0062      DATA >0000,>3030,>0030,>3000   * char=>3A :
0063      DATA >0000,>3030,>0030,>1020   * char=>3B ;
0064      DATA >0808,>0808,>0F00,>0000   * char=>3C <
0065      DATA >0000,>00FF,>00FF,>0000   * char=>3D -
0066      DATA >0808,>0808,>F800,>0000   * char=>3E >
0067      DATA >0000,>00FF,>00FF,>0808   * char=>3F ?
0068      DATA >0038,>4454,>5840,>3C00   * char=>40 @
0069      DATA >0038,>4444,>7C44,>4400   * char=>41 A
0070      DATA >0078,>2438,>2424,>7800   * char=>42 B
0071      DATA >0038,>4440,>4044,>3800   * char=>43 C
0072      DATA >0078,>2424,>2424,>7800   * char=>44 D
0073      DATA >007C,>4078,>4040,>7C00   * char=>45 E
0074      DATA >007C,>4078,>4040,>4000   * char=>46 F
0075      DATA >0038,>4440,>4C44,>3800   * char=>47 G
0076      DATA >0044,>447C,>4444,>4400   * char=>48 H
0077      DATA >0038,>1010,>1010,>3800   * char=>49 I
0078      DATA >0004,>0404,>0444,>3800   * char=>4A J
0079      DATA >0044,>4850,>7048,>4400   * char=>4B K
0080      DATA >0040,>4040,>4040,>7C00   * char=>4C L
0081      DATA >0044,>6C54,>4444,>4400   * char=>4D M
0082      DATA >0044,>6454,>544C,>4400   * char=>4E N
0083      DATA >0038,>4444,>4444,>3800   * char=>4F O
0084      DATA >0078,>4444,>7840,>4000   * char=>50 P
0085      DATA >0038,>4444,>544C,>3C00   * char=>51 Q
0086      DATA >0078,>4444,>7848,>4400   * char=>52 R
0087      DATA >0038,>4430,>0844,>3800   * char=>53 S
0088      DATA >007C,>1010,>1010,>1000   * char=>54 T
0089      DATA >0044,>4444,>4444,>3800   * char=>55 U
0090      DATA >0044,>4444,>4428,>1000   * char=>56 V
0091      DATA >0044,>4444,>5454,>2800   * char=>57 W
0092      DATA >0044,>2810,>1028,>4400   * char=>58 X
0093      DATA >0044,>4428,>1010,>1000   * char=>59 Y
0094      DATA >007C,>0810,>2040,>7C00   * char=>5A Z
0095      DATA >1414,>1417,>101F,>0000   * char=>5B [
0096      DATA >1414,>1414,>1714,>1414   * char=>5C \
0097      DATA >1414,>14F4,>04FC,>0000   * char=>5D ]
0098      DATA >0010,>2844,>0000,>0000   * char=>5E ^
0099      DATA >0000,>0000,>0000,>7C00   * char=>5F _
0100      DATA >0000,>0000,>FF08,>0808   * char=>60 `
0101      DATA >0000,>0038,>4848,>3C00   * char=>61 a
0102      DATA >0020,>2038,>2424,>3800   * char=>62 b
0103      DATA >0000,>001C,>2020,>1C00   * char=>63 c
0104      DATA >0004,>041C,>2424,>1C00   * char=>64 d
0105      DATA >0000,>001C,>2830,>1C00   * char=>65 e
0106      DATA >000C,>1038,>1010,>1000   * char=>66 f
0107      DATA >0000,>001C,>241C,>0438   * char=>67 g
0108      DATA >0020,>2038,>2424,>2400   * char=>68 h
0109      DATA >0010,>0030,>1010,>3800   * char=>69 i
0110      DATA >0008,>0008,>0808,>4830   * char=>6A j
0111      DATA >0020,>2024,>3828,>2400   * char=>6B k
0112      DATA >0030,>1010,>1010,>3800   * char=>6C l
0113      DATA >0000,>0078,>5454,>5400   * char=>6D m
0114      DATA >0000,>0038,>2424,>2400   * char=>6E n
0115      DATA >0000,>0018,>2424,>1800   * char=>6F o
0116      DATA >0000,>0038,>2438,>2020   * char=>70 p
0117      DATA >0000,>001C,>241C,>0404   * char=>71 q
0118      DATA >0000,>0028,>3420,>2000   * char=>72 r
0119      DATA >0000,>001C,>300C,>3800   * char=>73 s
0120      DATA >0010,>1038,>1010,>0C00   * char=>74 t
0121      DATA >0000,>0024,>2424,>1C00   * char=>75 u
0122      DATA >0000,>0044,>2828,>1000   * char=>76 v
0123      DATA >0000,>0044,>5454,>2800   * char=>77 w
0124      DATA >0000,>0024,>1818,>2400   * char=>78 x
0125      DATA >0000,>0024,>241C,>0438   * char=>79 y
0126      DATA >0000,>003C,>0810,>3C00   * char=>7A z
0127      DATA >0000,>001F,>1017,>1414   * char=>7B {
0128      DATA >0808,>0808,>0808,>0808   * char=>7C |
0129      DATA >0000,>00FC,>04F4,>1414   * char=>7D }
0130      DATA >1414,>1414,>1414,>1414   * char=>7E ~
0131  SLAST  END
```

My transliterate file looks like this:

```
0001  .CO IBM GRAPHIC ELEMENTS TRANSLITERATE
0002  .CO FILE WILL PRINT GRAPHIC ELEMENTS
0003  .CO WHEN USED WITH AN IBM COMPATIBLE
0004  .CO PRINTER - any printer capable of
0005  .CO printing the IBM CHARACTER SET
0006  .CO ==IBM double horizontal line
0007  .TL 61:205
0008  .CO ~=IBM double vertical line
0009  .TL 126:186
0010  .CO (=IBM double upper left corner
0011  .TL 123:201
0012  .CO )=IBM double upper right corner
0013  .TL 125:187
0014  .CO [=IBM double lower left corner
0015  .TL 91:200
0016  .CO ]=IBM double lower right corner
0017  .TL 93:188
0018  .CO ?=IBM double line top
0019  .TL 63:209
0020  .CO #=IBM double line bottom
0021  .TL 35:207
0022  .CO \=IBM double vertical line with rgt dash
0023  .TL 92:199
0024  .CO /=IBM double vertical line with left dash
0025  .TL 47:182
0026  .CO -=IBM horizontal line
0027  .TL 45:196
0028  .CO |=IBM vertical line
0029  .TL 124:179
0030  .CO (=IBM single upper left corner
0031  .TL 40:218
0032  .CO )=IBM single upper right corner
0033  .TL 41:191
0034  .CO <=IBM single lower left corner
0035  .TL 60:192
0036  .CO >=IBM single lower right corner
0037  .TL 62:217
0038  .CO `=IBM single line top
0039  .TL 96:194
0040  .CO $=IBM single line bottom
0041  .TL 36:193
0042  .CO +=IBM cross
0043  .TL 43:197
```

The transliterate file should be included in any file that you wish to print. This may be by physically having the listing in your file or by using the ".IF DSKx.filename" formatter command. If you use the latter it should be included before any reference is made to the transliterated characters

With my new CHARA2 file I can display the graphic elements I have converted by selecting the Programer's Editor from Funnelweb as my first Editor choice. If you

select the Word Processing Editor first CHARA1 will be used. If this happens and you need to switch to the CHARA2 file, exit the Editor and select the User List, then <FCT> 9 back and select the Programer's Editor. CHARA2 is now loaded and ready to use.

If there are any graphic elements that I haven't included and you need it is now a simple matter to redefine one with a sector editor on the CHARA1 file or in the Charaset DATA statements. Don't forget to add the transliterate to your transliterate file. If you don't send your printer embedded control codes you may use a lot of the characters below 32, and access them with the <CTL> U function. Do not use character 13 (carriage return) as a transliterate, since it is the one character that is almost always used. Actually that is probably a very good place to place your transliterates if you are selective in what you change and wish to keep all of your printable characters available.

For best printing results I usually print my charts in NLQ along with EMPHAIZED & DOUBLE-STRIKE. On my Seikosha SP-1200AI, NLQ is initiated by sending the printer "ESC,x,1" and MIXED MODE printing of EMPHAIZED & DOUBLE-STRIKE with "ESC,!,character 24". I embed these codes in my file and they are sent to the printer at the time of printing.

Happy screens

Tips from the Tigercub
Number 55
By Jim Peterson

Tigercub Software
156 Collingwood Ave.
Columbus OH 43213

I am still offering over 120 original and unique entertainment, educational and utility programs at just $1.00 each, or on collection disks a $5.00 per disk.
The contents of the first 52 issues of this newsletter are availabl as ready-to-run programs on 5 Tips Disks at $10 each
An my three Nuts & Bolts Disk, $15 each, each contain over 100 subprograms for you to merge into your own programs to do all kinds of wonderful things.
My catalog is available for $1, deductable from your first order (specify TIGERCUB catalog).

TI-PD LIBRARY

I have selecte public domain programs, by category, to fill over 200 disks, as full as possible if I had enough programs of the category, with all the Basic-only programs converted to XBasic, with an E/ loader provided for assembly programs if possible, instructions added and any obvious bugs corrected, and with an auto-loader by full program name on each disk. These are available as a copying service for just $1.50 post-paid in U.S. and Canada. No fairware will be offered without the author's permission. Send SASE for list or $1, refundable for 9-page catalog listing all titles and authors. Be sure to specify TI-PD catalog.

The Tigercub has dipped a cautious paw into the cold dark mysterious waters of asembly, while still keeping a firm grip on trusty old Extended Basic. The result is an XBasic program that writes an assembly program!
The following subprogram, when merged into any program which has reidentified characters, and called after the characters have been reidentified, will write a source code which can be assembled into object code, loaded from XBasic and linked to instantly access the character set.
The source code is based on 2FONTS/S by Barry Traver, who gives credit to Mac McCormick, David Migicovsky and Karl Schuneman.

```
19000 SUB CHARSUB(HX$())
19001 DISPLAY AT(12,1)ERASE
ALL:"Source code filename?":
"DSK" :: ACCEPT AT(13,4)SIZE
(12)BEEP:F$ :: OPEN #1:"DSK"
&F$,OUTPUT
19002 DISPLAY AT(15,1):"LINK
ABLE program name?" :: ACCEP
T AT(16,1)SIZE(6):P$
19003 DISPLAY AT(18,1):"Rede
fine characters from  ASCI
I to ASCII"
19004 ACCEPT AT(19,7)VALIDAT
E(DIGIT)SIZE(3):F
19005 ACCEPT AT(19,21)VALIDA
TE(DIGIT)SIZE(3):T
19006 PRINT #1:TAB(8);"DEF";
TAB(13);P$ :: PRINT #1:"VMBW
 EQU  >2024" :: PRINT #1:"
STATUS EQU  >837C"
19007 NB=(T-F+1)*8 :: CALL D
EC_HEX(NB,H$):: A=768+F*8 ::
 CALL DEC_HEX(A,A$)
19008 FOR CH=F TO T :: IF CH
<144 THEN CALL CHARPAT(CH,CH
$)ELSE CH$=HX$(CH)
19009 IF FLAG=0 THEN PRINT #
1:"FONT";:: FLAG=1
19010 FOR J=1 TO 13 STEP 4 :
: M$=M$&">"&SEG$(CH$,J,4)&",
" :: NEXT J :: M$=SEG$(M$,1,
23)&"  *"&CHR$(CH)
19011 PRINT #1:TAB(8);"DATA
"&M$ :: M$="" :: NEXT CH
19012 PRINT #1:P$;TAB(8):"LI
 R1,FONT" :: PRINT #1:TAB(
8);"LI R0,>"&A$ :: PRINT #
1:TAB(8);"LI R2,>"&H$
19013 PRINT #1:TAB(8);"BLWP
@VMBW":TAB(8);"CLR  @STATUS"
:TAB(8);"RT":TAB(8);"END" ::
 CLOSE #1
19014 SUBEND
```

```
19015 SUB DEC_HEX(D,H$)
19016 X$="0123456789ABCDEF"
:: A=D+65536*(D>32767)
19017 H$=SEG$(X$,(INT(A/4096
)AND 15)+1,1)&SEG$(X$,(INT(A
/256)AND 15)+1,1)&SEG$(X$,(I
NT(A/16)AND 15)+1,1)&SEG$(X$
,(A AND 15)+1,1):: SUBEND
```

Now to try it out. You probably know that CALL CHARSET will restore reidentified characters below ASCII 96 to normal form, but not those above, so let's write a routine to restore those. Clear the memory with NEW, merge in the above, which you should have SAVED with - SAVE DSK1.CHARSUB,MERGEby MERGE DSK1.CHARSUB. Add a line -100 CALL CHARSUB(HX$()) and RUN. Answer the filename prompt with DSK1.OLDLOW/S, the next prompt with OLDLOW and select ASCII 97 to 127.
When done, insert the Editor/Assembler module and its disk Part A. Select Assembler, Y to load assembler, give the source code DSK1.OLDLOW/S, object code DSK1.OLDLOW/O, just press Enter at next prompt, and R for options. You should get 0000 ERRORS.
Now key in this routine to test your program.

```
100 CALL INIT :: CALL LOAD("
DSK1.OLDLOW/O"):: FOR CH=33
TO 126 :: CALL CHAR(CH,"FF81
8181818181FF"):: PRINT CHR$(
CH);:: NEXT CH
101 CALL KEY(0,K,S):: IF S=0
 THEN 101 ELSE CALL CHARSET
102 CALL KEY(0,K,S):: IF S=0
 THEN 102 ELSE CALL LINK("OL
DLOW")
110 GOTO 110
```

Press any key to restore the upper case characters by CALL CHARSET, any key again to use the CALL LINK.
You are now ready to use the routine to copy all kinds of character sets from the programs in your library. You don't have any such programs? Not to worry. You don't have to reidentify characters one by one with one of those graphics editor programs. You can just manipulate the existing hex codes of the normal characters. I have created nearly 50 different character sets by that method!
The space occupied by a character on the screen is really an 8x8 square of 64 tiny dots. Various dots are turned on (colored) and off (transparent) to create a pattern - just the opposite of light bulbs on a scoreboard.
And those on-and-off dots are really the binary numbers which the computer uses. But fortunately the computer lets us use hexadecimal numbers rather than binary. The following will print out a reference chart of decimal to binary to hexadecimal. You can easily convert it to dump to a printer.

```
10 DISPLAY AT(6,1)ERASE ALL:
"DEC BIN HEX"
100 FOR J=0 TO 15 :: CALL DE
C_BIN(J,B$):: CALL DEC_HEX(J
,H$):: DISPLAY AT(J+8,1):J;T
AB(5);B$;TAB(10);SEG$(H$,4,1
):: NEXT J
21020 SUB DEC_BIN(D@,B$):: D
=D@ :: IF D=0 THEN B$="0000"
 :: SUBEXIT
21021 IF D=1 THEN 21022 :: X
=D/2 :: B@$=STR$(ABS(X<>INT(
X)))&B@$ :: D=INT(X):: IF D>
1 THEN 21021
21022 B@$="1"&B@$ :: B$=RPT$
("0",4-LEN(B@$))&B@$ :: B@$=
"" :: SUBEND
21039 SUB DEC_HEX(D,H$)
21040 X$="0123456789ABCDEF"
:: A=D+65536*(D>32767)
21041 H$=SEG$(X$,(INT(A/4096
)AND 15)+1,1)&SEG$(X$,(INT(A
/256)AND 15)+1,1)&SEG$(X$,(I
NT(A/16)AND 15)+1,1)&SEG$(X$
,(A AND 15)+1,1):: SUBEND
```

And this routine will show you how each letter is formed, by binary 0's (off) and 1's (on), for each key you press. I put it in merge format so you can MERGE it into any program and CALL it to examine the characters.

```
17000 SUB CHARVIEW
17001 !programmed by Jim Pet
erson Feb 1989
17002 DISPLAY AT(1,1)ERASE A
LL:"CHARACTERS IN BINARY & H
EX":;:"Press any key to see
the  binary representation
 of thescreen character and
its hexcode."
17003 DISPLAY AT(8,1):"Press
 Enter to see the char-acter
17004 CALL KEY(0,K,S):: IF K
=13 THEN 17005 ELSE IF S=0 O
R K<32 OR K>143 THEN 17004 E
LSE 17007
17005 CALL CHAR(48,"FF"&RPT$
("81",6)&RPT$("FF",9))
17006 CALL KEY(0,K,S):: IF S
<1 THEN 17006 ELSE CALL CHAR
(48,"003844444444444380010301
010101038"):: GOTO 17004
17007 CALL CHARPAT(K,CH$)
17008 R=12 :: FOR J=1 TO 15
STEP 2
17009 H$=SEG$(CH$,J,1):: CAL
L HEX_BIN(H$,B$)
17010 DISPLAY AT(R,8):B$
17011 H$=SEG$(CH$,J+1,1):: C
ALL HEX_BIN(H$,B$)
17012 DISPLAY AT(R,12):B$ ::
 DISPLAY AT(R,18):SEG$(CH$,J
,2):: R=R+1 :: NEXT J :: DIS
```

```
PLAY AT(22,6):CH$ :: GOTO 17
004
17013 SUBEND
17014 SUB HEX_BIN(H$,B$):: H
X$="0123456789ABCDEF" :: BN$
="0000X0001X0010X0011X0100X0
101X0110X0111X1000X1001X1010
X1011X1100X1101X1110X1111"
17015 FOR J=LEN(H$)TO 1 STEP
 -1 :: X$=SEG$(H$,J,1)
17016 X=POS(HX$,X$,1)-1 :: T
$=SEG$(BN$,X*5+1,4)&T$ :: NE
XT J :: B$=T$ :: T$="" :: SU
BEND
```

And to reidentify a character, you just change the numbers and letters in the 16-digit hex code which represents the binary pattern. By writing little routines to switch those digits around, all kinds of things can be done.
For instance, the normal characters always have the top row of dots turned off, to provide spacing between lines of text on the screen. If you want taller characters you will have to double-space the lines, but you can create them by making the numerals and upper case characters consist of the 2nd-7th rows, the 7th row again, and the 8th row - it just happens to work out.

```
18000 SUB HIGHCHAR :: FOR CH
=48 TO 90 :: CALL CHARPAT(CH
,CH$):: CALL CHAR(CH,SEG$(CH
$,3,10)&RPT$(SEG$(CH$,13,2),
2)&SEG$(CH$,15,2)):: NEXT CH
 :: SUBEND
```

I made that a subprogram so you can MERGE it in and use it to modify other character sets.

If we take the hex code apart, 2 digits at a time, and reassemble it backward,

```
100 CALL CLEAR :: FOR CH=33
TO 90 :: CALL CHARPAT(CH,CH$
):: FOR J=1 TO 15 STEP 2 ::
CH2$=SEG$(CH$,J,2)&CH2$ :: N
EXT J :: CALL CHAR(CH,CH2$):
: CH2$="" :: NEXT CH
110 DISPLAY AT(12,1):"?NWOD
EDISPU":"VT EHT DENRUT OHW !
YEH" :: GOTO 110
```

That one was in my first Tips newsletter, years ago, but it is much more effective at assembly speed.

This one shades characters on their left edge by turn- on the pixel to the left of the leftmost "on" pixel, if any. Also try it in combination with HIGHCHAR.

```
18001 SUB NEWCHAR3 :: FOR CH
=48 TO 122 :: CALL CHARPAT(C
H,CH$):: FOR J=1 TO 15 STEP
2
18002 CH2$=CH2$&SEG$("0367CD
EF",POS("01234567",SEG$(CH$,
J,1),1),1)&SEG$(CH$,J+1,1)::
 NEXT J :: CALL CHAR(CH,CH2$
):: CH2$="" :: NEXT CH :: SU
BEND
```

This one uses HIGHCHAR to heighten the character and then blanks out three rows. Try following it with NEWCHAR3.

```
18030 SUB NEWCHAR10 :: A$="0
0" :: FOR CH=48 TO 90 :: CAL
L CHARPAT(CH,CH$):: CH$=SEG$
(CH$,3,10)&RPT$(SEG$(CH$,13,
2),2)&SEG$(CH$,15,2)
18031 CH$=SEG$(CH$,1,4)&A$&S
EG$(CH$,7,2)&A$&SEG$(CH$,11,
2)&A$&SEG$(CH$,15,2):: CALL
CHAR(CH,CH$):: NEXT CH :: SU
BEND
```

The next one, which works only on ASCII 97-122, makes tall characters ridiculously elongated above.

```
18050 SUB NEWCHAR20 :: FOR C
H=97 TO 122 :: CALL CHARPAT(
CH,CH$):: CALL CHAR(CH,SEG$(
CH$,7,2)&RPT$(SEG$(CH$,9,2),
4)&SEG$(CH$,11,6)):: NEXT CH
 :: SUBEND
```

This one has the characters raised by one line, widened one column at left and two columns at right to make a full 8x8 character which must be double-spaced horizontally and vertically.

```
18090 SUB NEWCHAR27 :: FOR C
H=48 TO 122 :: CALL CHARPAT(
CH,CH$):: CH$=SEG$(CH$,3,10)
&RPT$(SEG$(CH$,13,2),2)&SEG$
(CH$,15,2):: FOR J=1 TO 15 S
TEP 2
18091 CH2$=CH2$&SEG$("014589
CD",POS("01234567",SEG$(CH$,
J,1),1),1)&SEG$("0129",POS("
048C",SEG$(CH$,J+1,1),1),1)
18092 NEXT J :: CALL CHAR(CH
,CH2$):: CH2$="" :: NEXT CH
:: SUBEND
```

Those who have my Nuts & Bolts disks will see how valuable this assembly can be to make instantly available the routines for double height and double width characters, etc., etc. And if you have Todd Kaplan's amazing ALSAVE routine from the Genial Traveler Vol. 1 No. 3, you can imbed them in your XBasic program for fast loading.
And you can merge CHARSUB into any character editor or sprite defining program and, with a bit of modification, use

it to convert your creations into fast-loading assembly. These assembly loads are compatible with my BXB, so you can also load character sets into sets 15 and 16, ASCII 144-159. However, the CHARPAT statement cannot access ASCII above 143, so in this case you must dimension an array in the program you are copying from, as DIM HX$(159), and place the hex codes in the array using the ASCII as the sub-script number, such as CALL CHAR(CH+64,CH$) :: HX$(CH+64)=CH$, so that they will be passed to the subprogram. And don't CALL INIT after you have called BXB!
So, now you try creating your own screen fonts!

Memory full,

Jim Peterson

More on PostScript

©1989 J. Peter Hoddie

In the last newsletter (the January Newsletter to be precise) the last page was a picture a small portion of text describing how I managed to print a TI-Artist picture on a PostScript laser printer. This is a very straight forward procedure, but it is rather involved. The results are rather impressive, so I thought it might be useful to publish the approach. The overall result is that it is possible to get extremely high quality output of TI-Artist pictures - provided you have access to a PostScript laser printer, a device which costs about $3500, on the low end.

PostScript is a language, similar in many respects to Forth, which is present in many laser printers. A popular example of such a printer is the Apple LaserWriters. These are often found hooked up to Macintosh computers (as the are in the BCS main office) and so this is what I used in my approach. It is possible to actually hook a TI directly up to a PostScript printer using a serial cable. In this case, the procedure described here would have to be modified somewhat, but the basic ideas still apply.

Printing on a PostScript printer is rather different from printing on most printers. Since PostScript is a language, the printer contains an entire computer with lots and lots of memory (like at least one megabyte!). To print, you send a program to the printer which draws contains a series of commands to draw the page in the printers memory. The printer then copies this image to the paper. Thus rather than just sending data to the printer, as we do on the TI, one must send both data and instructions on how to format that data. The first part of the process is to convert the TI-Artist picture to a format that can be included in a PostScript program. We can then simply insert this data into a program which is the same for every TI-Artist picture we wish to print.

PostScript was designed to be read by humans. It is interpreted rather than compiled. This does not make it particularly fast, but is makes things easy for our purposes. The data in the TI-Artist picture file (_P extension) is simply converted to ASCII characters. This is essentially what the Extended BASIC CALL CHARPAT does. But since we have to do it over and over and over and over again to the characters in a TI-Artist file I wrote an assembly program to handle the details.

The assembly program is really primitive, but it is functional. I wrote it in about 15 minutes, mostly by ripping apart MacFlix. It has no input routine, so you have to change the filenames and reassemble to change the input and output filenames. Sorry but the input routine I usually use is about 200 lines long and would over shadow this article if I printed it too! All the assembly program does is load the TI-Artist picture file and output it to a DIS/FIX 128 file which is just the ASCII equivalent of the numbers contained in the picture. The output file will be twice the size of the input file, since each byte is converted to a two byte ASCII representation. If there is not enough disk space you will be dumped back to the title screen. In any case, below is the program.

```
       DEF  START
       REF  DSRLNK
       REF  KSCAN
       REF  VDPWA,VDPWD
       REF  VSBW,VMBW
       REF  VSBR,VMBR
*
PAB    EQU  >A00
PABBUF EQU  >A80
PICBUF EQU  >1000
*
WS     EQU  >8300
*
HEX    TEXT '0123456789ABCDEF'
*
TIAPAB DATA >0500,PICBUF,0,>2000,10
       TEXT 'DSK2.TIA_P'
       EVEN
PSTPAB DATA >0002,PABBUF,>8080,0,9
       TEXT 'DSK2.POST'
       EVEN
WRITE  BYTE 3
CLOSE  BYTE 1
       EVEN
*
START  LWPI WS
       LIMI 0                  the just in case....
*
       BL   @VMBWD
       DATA PAB,TIAPAB,>40
       BL   @DSR               get the file...
       JMP  ERROR
*
       LI   R0,PICBUF
       LI   R1,>C000
       LI   R2,6144
       BLWP @VMBR              copy picture into cpu
memory...
*
       BL   @OUTOPN            create an output
file....
*
```

```
       CLR  R6                  character row counter (0
to 23)
*
NXT10  CLR  R7                  bit column counter (0 to
7)
*
NXT15  CLR  R0                  character column counter
(0 to 31)
*
       MOV  R6,R5
       SLA  R5,8                *256 bytes per character
row
       AI   R5,>C000            point to the appropriate
character row
       A    R7,R5               get bit column offset
NXT20  MOVB *R5,R1              get next byte
       SLA  R1,4                kill high nibble
       SRL  R1,12
       MOVB @HEX(R1),R0         get low nibble as ascii
       SWPB R0
       MOVB *R5,R1              reget he byte
       SRL  R1,12
       MOVB @HEX(R1),R0         get high nybble as ascii
       AI   R5,8                point to next character
definition....
       BL   @OUTWRD             output this word....
*
       INC  R8                  count a column...
       CI   R8,31               done with this bit row?
       JLE  NXT20                nope...
*
       INC  R7                  increment a bit column
       CI   R7,7                done with this character
row?
       JLE  NXT15                nope... do next bit
column
*
       INC  R6                  next character row....
       CI   R6,23               done with picture?
       JLE  NXT10                nope. keep going...
*
       BL   @OUTCLS             close the file...
SAVERR
ERROR  BLWP @0                  and get out...
*
*
*                                R0=0 or not as well...
*
*      graphics DSR routines mostly
*
OUTOPN CLR  @OUTLOC             pop buffer pointer back
to start...
       CLR  @PSTPAB+6           record number to zero
       MOVB @OUTLOC,@PSTPAB     opcode to open..
*
OUTDSR MOV  R11,@OUTDRT         save return address
*
       BL   @VMBWD
       DATA PAB,PSTPAB,40                    copy the
output pab...
*
       BL   @DSR
       JMP  OUTOPE              error.....
*
       CB   @PSTPAB,@WRITE      are we writing?
       JNE  OUTDRX               nope
       INC  @PSTPAB+6           yes, so skip to next
record.
*
OUTDRX B    @0
OUTDRT EQU  $-2
*
OUTOPE B    @SAVERR             flash the screen and
return to picture...
*
* r0 is the word to send out . . .output buffer
flushed when 64 words
*      are loaded...
*
* uses r0 to r3
*
OUTBUF BSS  128                 output buffer
OUTLOC DATA 0                   current offset into
output buffer
*
OUTWRD MOV  R11,R10
*
       MOV  @OUTLOC,R1          fetch current output
location
       MOV  R0,@OUTBUF(R1)      install output word
       INCT R1                  next output address
       CI   R1,128              do we need to flush
buffer?
       JEQ  OUTWR1               yes
       MOV  R1,@OUTLOC          save new output location
       RT                       and return
*
OUTWR1 CLR  @OUTLOC             new buffer . . .
*
       MOVB @WRITE,@PSTPAB      change to write mode
       BL   @VMBWD              copy data into vdp
       DATA PABBUF,OUTBUF,128
*
       MOV  R10,R11
       JMP  OUTDSR
*
OUTCLS MOV  R11,@OUTCLR
       MOV  @OUTLOC,R1          is output buffer empty?
       JEQ  OUTCL1               yes, so no need to
flush...
       LI   R10,OUTCL1          make sure we return
       JMP  OUTWR1               no, so flush the output
buffer
*
OUTCL1 MOVB @CLOSE,@PSTPAB
       LI   R11,0
OUTCLR EQU  $-2
       JMP  OUTDSR
*
*---------------
*      dsr subroutine...returns $+2 if no error
*---------------
*
DSR    LI   R0,PAB+9
DSRDAT EQU  $-2
       MOV  R0,@>8356
       BLWP @DSRLNK
       DATA 8
       JEQ  DSRNOP              error...
       INCT R11                 no error...
```

```
DSREX  RT                      return to either
DSRNOP LI   R0,PAB+1
       BLWP @VSBR              get error
       SRL  R1,13
       JNE  DSREX              not error zero so return
correctly
       LI   R1,>0007           error 7 if it was error
0
       JMP  DSREX
*
* vmbwd is useful and self explanatory...
*
VMBWD  MOV  *R11+,R0
       MOV  *R11+,R1
       MOV  *R11+,R2
       BLWP @VMBW
       RT
*
       END
*
```

After you have run this program, you have an output file that contains the entire picture as one really long hex string. Imagine it as being the worlds longest CALL CHAR string! I then transferred the file to a Macintosh using an XModem transfer. The file may then be directly opened by any popular Macintosh word processor. I used Microsoft Word Version 3.02, which is probably the most popular and powerful Mac word processor widely available.

When in Word, type in the PostScript program listed below. When you encounter the line the reads "INSERT DATA HERE" include the data file from the TI and then continue with the final line.

```
.page.
/concatprocs
{ /proc2 exch cvlit def
  /proc1 exch cvlit def

  /newproc proc1 length proc2 length add
    array def
  newproc 0 proc1 putinterval
  newproc proc1 length proc2 putinterval
  newproc cvx
  } def

/inch { 72 mul } def
/picstr 3 string def

/imagetia
  { 256 192 1 [ 256 0 0 -192 0 192 ]
    { currentfile picstr readhexstring pop }
    image
  } def

gsave
  3 inch 4 inch translate
  2 inch dup scale

{ 1 exch sub } currenttransfer concatprocs
  settransfer

imagetia

**** INSERT DATA HERE *****

grestore
```

This program was taken from page 149 of PostScript Language Tutorial and Cookbook from Adobe Systems Incorporated. Published by Addison-Wesley.

The program is straight PostScript except for the use of the ".page." command as the first line which is only necessary when printing from Microsoft Word. If using Word this document must be formatted in the "PostScript" style. Once the program is typed in, select the entire document and select "Styles" from the "Format" menu. You want the "PostScript" style, so type that in for the style name. This tells Word to print the document as PostScript program. For details on sending PostScript using Word see pages 279-284 of the "Reference to Microsoft Word" which is shipped as part of the Microsoft Word package. If you don't use the PostScript style sheet, you will just get a print out of the PostScript program rather than the picture.

By messing around with the numbers in the lines

```
3 inch 4 inch translate
2 inch dup scale
```

you can change the size of the picture. For a details explanation of how to stretch, rotate, enlarge, and otherwise the picture, see the appropriate reference manuals from Adobe. They are a bit more clear than the TI Editor/ Assembler manual, although not by all that much!

If you are print from something other than Word, you will probably need to include a "showpage" as the final line in the program. Microsoft warns against doing this in the Word reference, so it is not included above.

While this procedure seems somewhat long and confusing, it is really quite straightforward. If you have access to a PostScript printer and have some patience, I hope you'll give it a try. It is certainly a rather involved procedure. It would be possible to write a more complete program on the 99/4A which would produce a file that includes both the PostScript program and the data, but alas I didn't do that here. If someone does this, please let me know. I would be interested to see the results.

Well. I had intended to end the article at this point, but it seems that there is about a half a page more of space to fill. I don't have any particularly unique ideas on my mind (is is now about 3AM) so I'll just explain a few details of the assembly program. The program actually illustrates a few useful programming techniques.

The first, is the use of two of my favorite subroutines. The first is the ever popular VMBWD which is simply a VMBW that takes its values from the three data words following the

calling instruction. This is really useful for writing certain kinds of code, in particular those that deal with DSR's. I find code written with the VMBWD to be a bit easier to read. Also I have found that there are so many many places in some programs where you just do a LI R0,xxx and so on to prepare for a VMBW that this approach saves a few bytes in larger programs. In some cases I have also written VMBRD routines as well. Enough said on that one.

The DSR subroutine is used to call the DSRLNK for all peripheral access. This has the advantage of putting all the code to load >8356 and do error checking in one place. Error checking is a real hassle, and it should be done after every DSR call to be safe. This routine uses a simple approach. It returns to the next address if there was an error. This address contains a JMP instruction to an error handler. If there is no error, the return address is incremented by 2 and the JMP instruction is skipped, so the program continues on normally. This approach works fine for smaller programs. For a larger program, it would probably be better to have a DATA statement following the DSR call. The data would be the address of an error handler. The DSR subroutine would then branch to this address in case of an error. This has the advantage of allowing for error handlers anywhere in memory, not just within 256 bytes of the subroutine call. Each approach uses about the same amount of memory.

If you study thje program listing you'll note the section titled "graphics DSR routines mostly". These routines handle the output of the DIS/FIX 128 file. I lifted them directly from MacFlix (publishing commerical software source code, see what a BCS membership gets you!!!) with no changes. These routines are very helpful in situations where you need to output bytes to a file composed of fixed length records. The call to the close routine (OUTCLS) makes sure that all data has been output before closing the file.

As a final note (and one in search of a law suit I'm sure) I would just like to mention that I do not expect that Tex Comp will be selling this article or its contents in any form. I mean they wouldn't consider releasing a disk called "PostScript - More" or "More ScriptPost" or event "More PostFlicks" or something equally creative. I don't think it would sell all that well. It certainly wouldn't justify taking out a full page in MICROpendium. But then what do I know about marketing, or originality? Certainly no one could put a price on my lack of knowledge. Or strange writing habits at odd hours.

Just for the record (related to the above paragraph is some strange meta-physical way, but not the rest of the article) my MacFlix program was named by the wife of Scott Darling. She suggested MacFlicks or MacFlicker. In a fit of overwhelming cleverness I changed the "cks" to an "x" thus arriving at the now infamous MacFlix. I do admire the incredible perception of those who managed to reverse my thought processes.