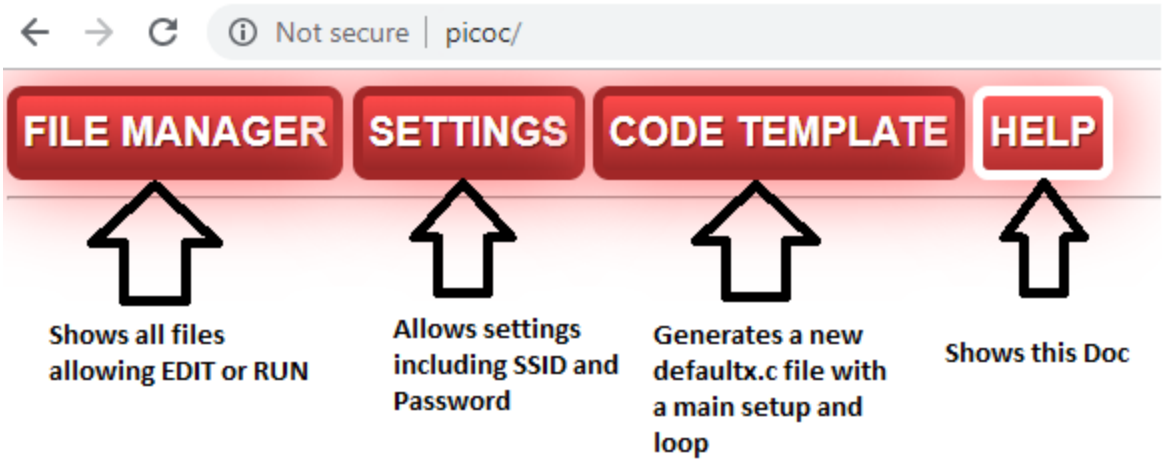
# ESP32 picoc C Language Interpreter

The sketch at https://github.com/rlunglh/IOT_with_your_PC/blob/master/ESP32Program.zip implements a C Language Interpreter on the ESP32 using the Arduino IDE to create and upload the sketch to your ESP32. There are many good explanations of setting up the Arduino IDE for the ESP32; a good place to look for this information is at https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/

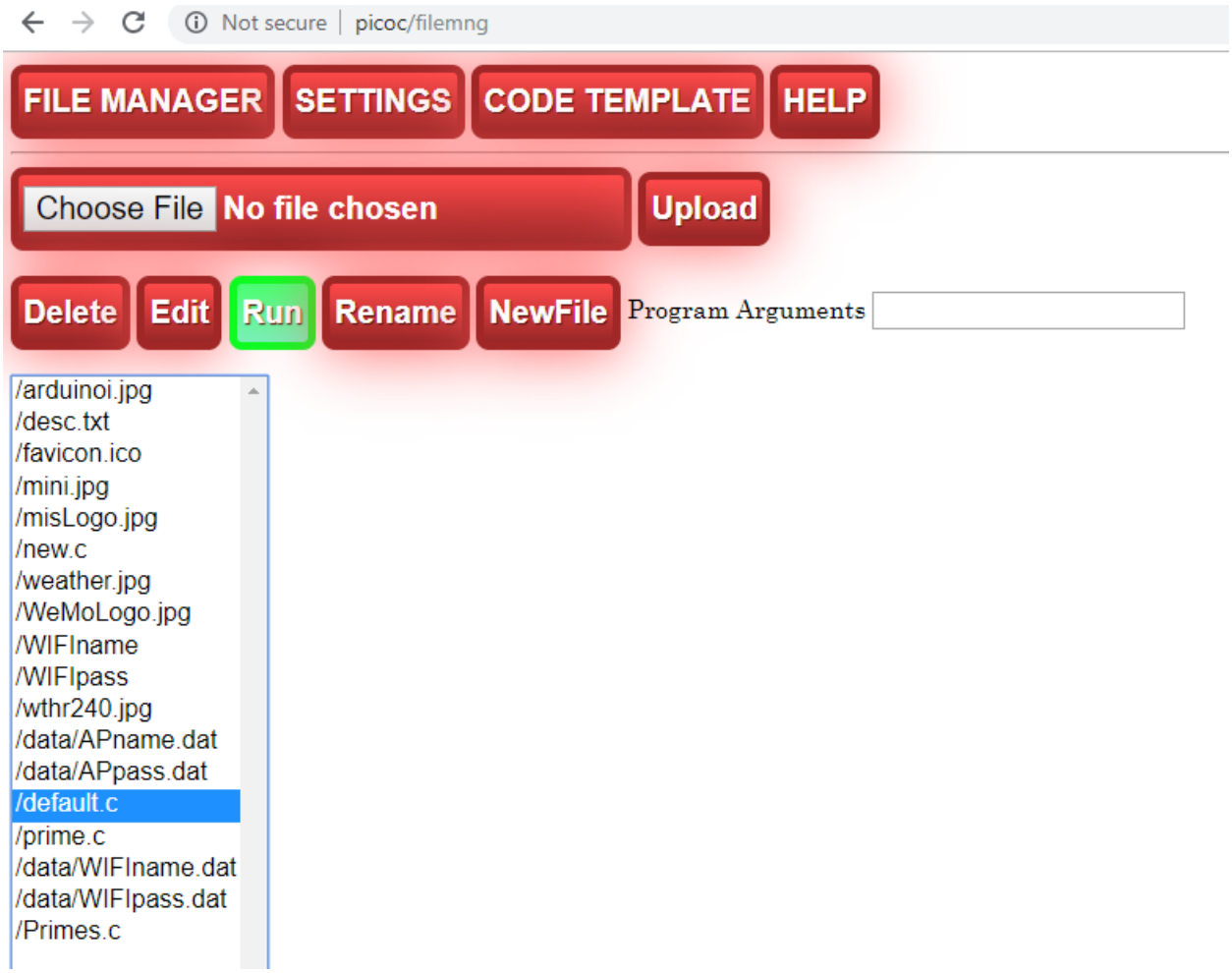
Once the Arduino IDE is installed just unzip the ESP32Program.zip file into your Arduino folder. You need to edit and upload some files to the ESP32 for the sketch to run. First look at the WIFIname.data file in the ESP32Program/data/data folder on your PC. This file supplies the SSID for the ESP32 to use when connecting to WiFi as a station, so change its name to match your WiFi environment. You also need to edit the ESP32Program/data/data/WIFIpass.dat file to hold your WiFi network password for the ESP32 to use when connecting. After editing these files, you can use the Tool/ESP32 sketch data upload menu in the Arduino IDE item to upload the data folder to your ESP32. To install the ESP32 upload tool follow the instructions at https://github.com/me-no-dev/arduino-esp32fs-plugin You may want to make some changes to the ESP32Program file in the Arduino IDE prior to building and uploading your sketch to your ESP32. The first few lines of the sketch are shown below:

```
#define ALWAYS_STATION
// define ALWAYS_STATION to restart EPS32 if it can't connect to the specified SSID
//                       when defined it will not enter AP mode on a Station Connect
//                       failure. It will restart until it successfully connects
//                       to the SSID specified in /data/WIFIname.dat using the
//                       password specified in /data/WIFIpass.dat
//#define TFT
// define TFT to enable use of an attached 320x240 TFT Display
//#define OLED
// define OLED to enable use of an attached 128x64 OLED Display
```

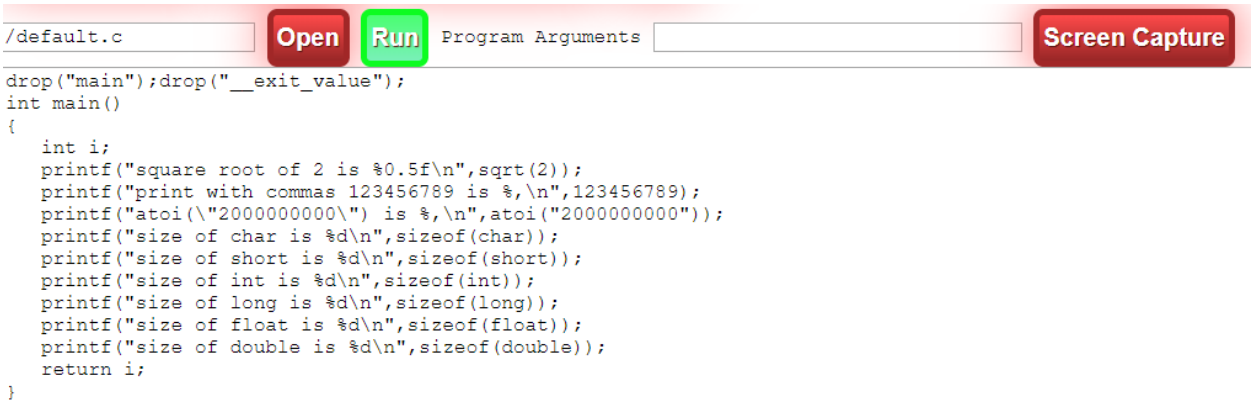
As shown in the code comments ALWAYS_STATION is defined to assure that the ESP32 will not enter Access Point mode. If a station connect fails the ESP32 will repeatedly restart and try to connect to the specified SSID until connection is successful. If you do have a problem connecting, make sure that the WIFIname.dat and WIFIpass.data files in ESP32Program/data/data on your PC have the correct information for your environment. If you want to use the esp32 in AP mode, just put a blank into the WIFIname.data file and comment out the #define ALWAYS_STATION line.
He ESP32PEogram supports an SPI connected TFT color graphic display or a 128x64 OLED display. These are enabled by uncommenting the //#define TFT or //#define OLED lines. You can use OLED and TFT defines concurrently and get a sketch that will work with either an OLED or a TFT display. Once you have the setting needed in the sketch, build and Upload for your sketch to the ESP32. When running, the Serial Monitor (at 1000000 baud) will show the IP address where the picoc Interpreter is running when the ESP32 restarts. In my environment I have added the line 192.168.0.21  picoc to my ..\Windows\System32\drivers\etc\hosts file to allow me to use picoc as an alias for the server.  In a browser, navigate to the ESP32 address to see the first display. An example browser view is show below:



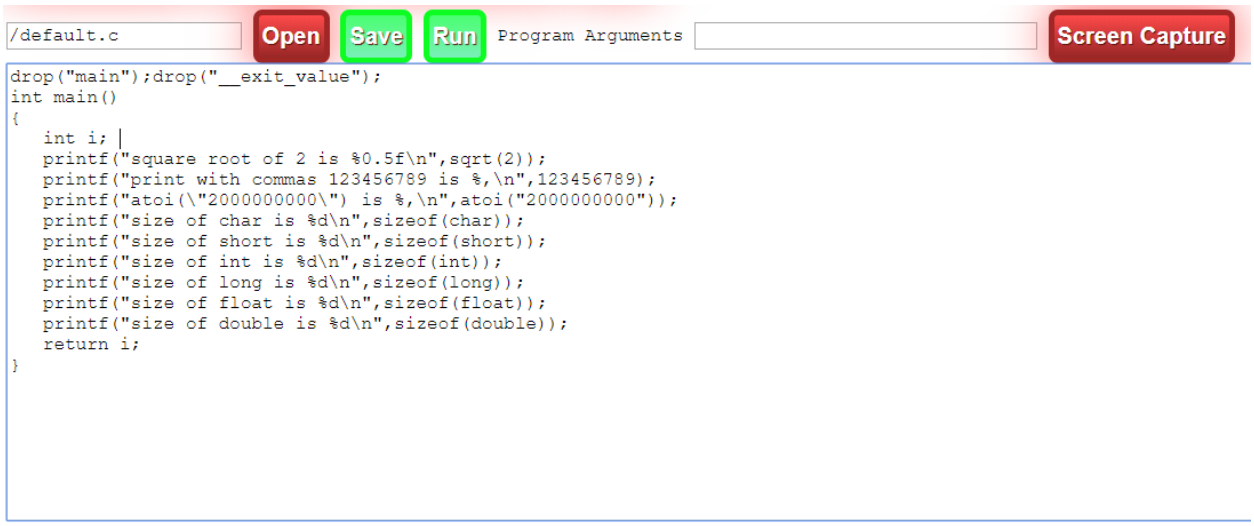

The following paragraphs give a quick tour of operation. Click the File Manager button giving the display shown below:




Select the /default.c line in the select list and click the Edit button giving:




This shows the content of the /default.c file and allows editing the content and saving changes using the Save Button. The Save button will not appear unless you make a change in the edit textarea. When a Save is required, the display will appear as shown in the following figure.

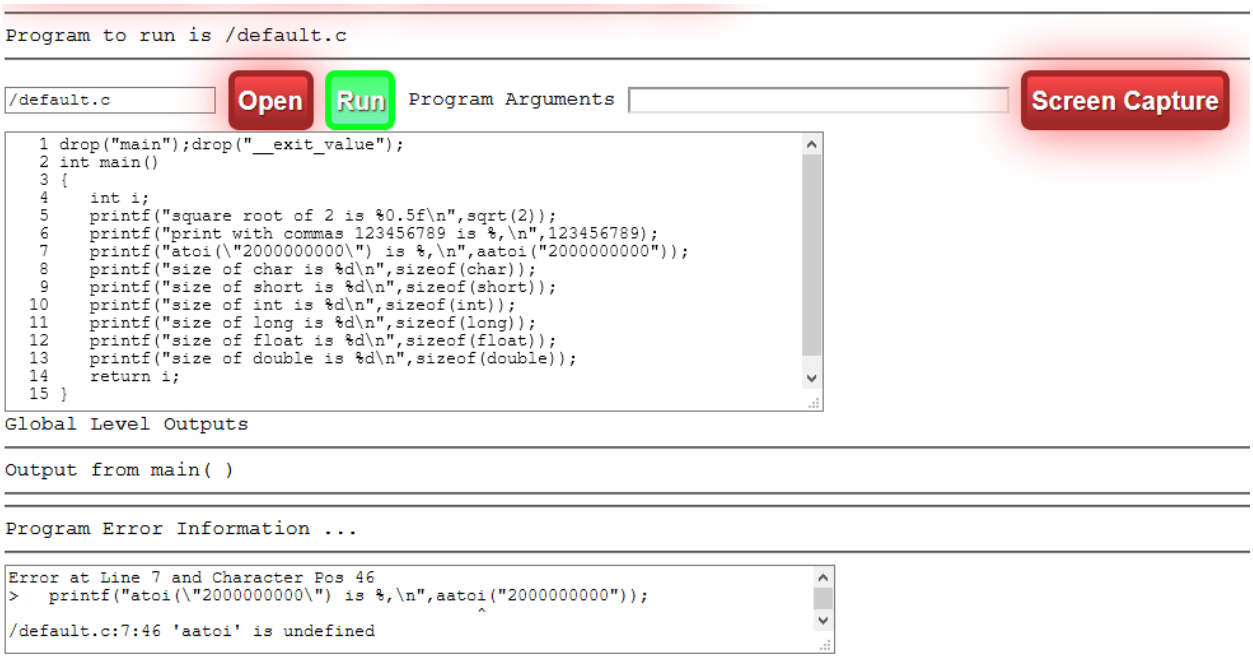



You can also change the edited file by changing the file name in the first text box and clicking Open. If the Save button is available, you can change the file name in the text box and the file will be saved to the named entered. File names can include / to breakout files into different directories and you are free to organize the files any way you choose. The File Manager will list all files showing each file's full path.

This example is done just to show a simple program's operation. Click the Run button giving the output shown in the following figure:


When programs are run, you'll be shown what file is running and any arguments passed along with a listing of the file, Global outputs, the program output, the return value, and the amount of free memory available on the ESP32 when the program completes.. This example was chosen to let you see that int and long types are both 32 bit values, and that the float and double types are both 64 bit values in the Interpreter.

If a running program has a syntax error, you will be shown the Program Error Information. As an example, click the Edit button and change the 2nd occurrence of atoi to aatoi and click the Save button. Now when you click Run you'll see the display shown below:




**Restarting ESP32 .....**

**Redirecting to Edit Page for /default.c in 30 seconds**

**You can also click Open button to go there quicker**

```
-> square root of 2 is 1.41421
-> print with commas 123456789 is 123,456,789
```

The Program Error Information section shows the error, along with the line and character position when the interpreter reached an error condition. It also shows the error cause, in this case, 'aatoi' is undefined. This particular error is well diagnosed and you can verify that clicking Open and changing aatoi to atoi, Saving the file, and clicking Run causes the program to again run without errors.

The handlers around the picoc interpreter eliminate any Interpreter error cleanup problems by simply restarting the ESP32, after a delay, when a program error occurs. You will be returned to the Run page for program you are running after a 30 second delay to allow for ESP32 reboot and its WIFI to connect to the specified SSID. You can also click the Open button on the Run page to get back to the editor more quickly.

The drop(char *) function is included in the Interpreter to allow you to repeatedly run a program. Without the drop function, when you run a program for the second time the Interpreter would throw an error message saying that 'main' was already defined. Programs with a return value also need to drop("__exit_value") before the main function appears in the listing. Programs using arguments need to Drop("__argc") and drop("__argv"). Also any function or variables that appear at Global scope must have a drop statement for their names or the Interpreter will complain that they are already defined.

There are sample programs included in the sketch data directory, and hence uploaded to the ESP32 using Tool/ESP32 Sketch Data Upload menu item ( make sure the Serial Monitor is closed before doing the upload ) that will let you explore some picoc Interpreter basics.

The programs

- default.c
- nested.c
- nestedDebug.c
- prime.c
- Primes.c
- primeWithSubs.c
- tester.c

Can all be run without needing an external TFT or OLED display. Each program has a short synopsis in the following paragraphs.

default.c – you've already been shown this program's operation

nested.c – shows the operation of nested for loops in the interpreter and is really present to set the stage for nestedDebug.c

nestedDebug.c - This program introduces the interpreter's debug features. Its output when run is shown in the following figure.

```
Program to run is /nestedDebug.c

/nestedDebug.c   Open   Run   Program Arguments

   1 drop("main");drop("__exit_value");drop("__argc");drop("__argv");
   2 int main(int argc,char ** argv)
   3 {
   4   Watch("i");
   5   Watch("j");
   6   for (int i=0;i<4;i++)
   7   {
   8     if (i==2) Debug();
   9     if (i==3) stopDebug();
  10     for (int j=0;j<4;j++)
  11       printf("i*4+j = %d\n",i*4+j);
  12   }
  13   return 0;
  14 }

Global Level Outputs

Output from main( )

-> i*4+j = 0
-> i*4+j = 1
-> i*4+j = 2
-> i*4+j = 3
-> i*4+j = 4
-> i*4+j = 5
-> i*4+j = 6
-> i*4+j = 7
   9:  0  >    if (i==3) stopDebug();          |i : int 2 | j : int 4
  10:  0  >    for (int j=0;j<4;j++)           |i : int 2 | j : int 4
  10:  8  >    for (int j=0;j<4;j++)           |i : int 2 | j : int 4
  11:  0  >      printf("i*4+j = %d\n",i*4+j);|i : int 2 | j : int 0
-> i*4+j = 8
  10: 21  >    for (int j=0;j<4;j++)           |i : int 2 | j : int 0
  11:  0  >      printf("i*4+j = %d\n",i*4+j);|i : int 2 | j : int 1
-> i*4+j = 9
  10: 21  >    for (int j=0;j<4;j++)           |i : int 2 | j : int 1
  11:  0  >      printf("i*4+j = %d\n",i*4+j);|i : int 2 | j : int 2
-> i*4+j = 10
  10: 21  >    for (int j=0;j<4;j++)           |i : int 2 | j : int 2
  11:  0  >      printf("i*4+j = %d\n",i*4+j);|i : int 2 | j : int 3
-> i*4+j = 11
  10: 21  >    for (int j=0;j<4;j++)           |i : int 2 | j : int 3
  12:  0  > }                                  |i : int 2 | j : int 4
   6: 19  > for (int i=0;i<4;i++)              |i : int 2 | j : int 4
   7:  0  > {                                  |i : int 3 | j : int 4
   7:  0  > {                                  |i : int 3 | j : int 4
   9:  0  >   if (i==3) stopDebug();           |i : int 3 | j : int 4
   9: 13  >   if (i==3) stopDebug();           |i : int 3 | j : int 4
-> i*4+j = 12
-> i*4+j = 13
-> i*4+j = 14
-> i*4+j = 15

Program returned 0, free memory is 103428
```

You see the program listing as in the program examples shown for default.c. Lines 4 and 5 set watches for variables i and j. Line 8 initiates debug operation when i is equal to 2. Line 9 stops debug operation when i is equal to 3. The interpreter gives you a conditional debugging feature since the Debug() functions can be called with any combination of conditional logic you'd like. The Output from main()

section shows a tabular format. Outputs from the program ( here caused by printf functions ) are entered on a separate line preceded by '-> '. The source lines being executed begin to show after the Debug() function is called. Line 8 of the program causes debug operation to start when i is 2. The display then begins showing debug trace output with a line#: col# > leader and the line being executed next shown with an inverse character at the position where the parser will scan next. Also shown in this table is a second column that presents any Watch variables. At the first trace line 9, I and j are shown to be 2 and 4 respectively. The next line shows that the for j statement is about to be executed. The 10:8 line shows that the for evaluation will next execute the int j=0 assignment. Line 11:0 shows that the printf is the next statement to be executed and that j is now 0. After this the program output form the printf function is printed as i*4+j = 8 . The for loop trace then continues until we get to 9:13 when the stopDebug() function is executed and tracing stops. After that the remaining program outputs are printed up to i*4+j = 15 and the program returns the value 0.

These combinations of Debug, stopDebug, Watch, and stopWatch function, along with conditional execution, give you great flexibility in examining program execution and working out problems in C language code.

The prime.c, Primes.c and primeWithSubs.c all explore computationally demanding application for the ESP32. These programs are all implementations of a prime number sieve that checks for integer divisors of a number with no remainder and if none are found the number is added to the prime number count. Also prime numbers found can be output for each n primes specified as a range. Let's look at the output of prime.c as provided in the stock ESP32Progrma data directory.

/prime.c Open Run Program Arguments Screen Capture

```
 1 drop( main );drop( __exit_value );drop( __argc );drop( __argv );
 2 int main(int argc,char ** argv)
 3 {
 4   int t1,start=1001,end=5000,mcount=0, k, n, lim, sn, p, pc=0, lastp=0, phold=0,rpt=100;
 5   if (argc>=4)
 6   {
 7     argv++;
 8     start=atoi((char *)*argv++);
 9     end=atoi((char *)*argv++);
10     rpt=atoi((char *)*argv++);
11   }
12   n = start; lastp=n; lim = end; sn=n;
13   printf("Finding primes between %, and %,\n\n",n,lim);
14   t1=sysTime();
15   while (n<lim)
16   {
17     k = 3; p = 1; n = n + 2;
18     while ((k * k <= n) && p)
19     {
20       p = n%k; mcount++; k = k + 2;
21     }
22     if (p)
23     {
24       pc = pc + 1; phold=n;
25       if (pc%rpt==0)
26       {
27         printf("prime # %5, is %7, in this range %5.2f %% are primes\n",pc,n,(float)rpt*100.0/(n-lastp));
28         lastp=n;
29       }
30     }
31   }
32   float  runsec=(sysTime()-t1)/8.0;
33   printf("\n\nFound %, primes\n", pc);
34   printf("Last prime was %,\n\n",phold);
35   printf("It took %0.3f seconds to process\n",runsec/100.0);
36   int pers=100.0*mcount/runsec;
37   printf("%, loops were performed, that's %, per second\n",mcount,pers);
38   return pc;
39 }
```

Global Level Outputs

Output from main( )

```
-> Finding primes between 1,001 and 5,000

-> prime #   100 is   1,721 in this range 13.89 % are primes
-> prime #   200 is   2,503 in this range 12.79 % are primes
-> prime #   300 is   3,323 in this range 12.20 % are primes
-> prime #   400 is   4,129 in this range 12.41 % are primes
-> prime #   500 is   4,993 in this range 11.57 % are primes
->
Found 501 primes
-> Last prime was 4,999

-> It took 3.833 seconds to process
-> 19,205 loops were performed, that's 5,011 per second
```

Program returned 501, free memory is 100396

As you can see above, it lists the prime found after each 100 discoveries, and processes over a range of 1001 to 5000. The program also times its execution and counts how many while ( (k*k<=n && p) loops are performed to allow reporting the loops complete per second, in this case, 5,011 per second. I acknowledge that 5,001 loops including a multiplication, two compare, a modulo divide, an increment and an add per second is not exceptional performance. The Primes.c program illustrates how this can be improved by placing the while loop into compiled code that is called by the interpreter. Primes.c Run output is shown in the next figure.

```
Program to run is /Primes.c
```

/Primes.c Open Run Program Arguments Screen Captu

```
  1 drop("main");drop("__exit_value");drop("__argc");drop("__argv");
  2 int main(int argc,char ** argv)
  3 {
  4  int start=2000000001,end=2000100000,rpt=1000;
  5  if (argc>=4){ argv++; start=atoi((char *)*argv++); end=atoi((char *)*argv++); rpt=atoi((char *)*argv++); }
  6  int pc=isprime(start,end,rpt);
  7   return pc;
  8 }
```

```
Global Level Outputs

Output from main( )

Finding primes between 2,000,000,001 & 2,000,100,000

prime #  1,000 is 2,000,021,137 4.73 % are prime
prime #  2,000 is 2,000,042,053 4.78 % are prime
prime #  3,000 is 2,000,062,553 4.88 % are prime
prime #  4,000 is 2,000,083,651 4.74 % are prime
Last prime was 2,000,099,957


4,745 primes found
126,207,107 prime check loops performed in 11.64 sec
That's 10,845,330 per second

Program returned 4745, free memory is 98936
```

Here we ran the prime number sieve over much larger numbers, to get more while loop iterations and showed results for every 1000 primes found. The startling this is that when the Interpreter calls compiled code to do the computationally demanding parts of the program, the while loops soar to close to 11 million per second which is over a 21,000 times improvement. The compiler shouldn't get all the credit because this interpreter is particularly slow – but it does fit nicely on the ESP32. ESP32Program can give you some very rewarding experience if you are new to C programming – just put code that you want to verify into a file on the ESP32 and test away without the Arduino compile and upload delays, and with a conditional trace and watch facility. If you have a section of sketch code that is not operating as you expected or hoped – just paste the appropriate portion of the code into an ESP2 file and run the Debug and Watch functions to see what is really happening without reverting to myriad Serial.print(f) functions.

The primesWithSubs.c program illustrates using a subroutine to perform the while ((k*k<n)  && p) loop.

All three of these programs can be run using program arguments entered, logically enough in the Program Arguments text box. A run of the primeWithSubs.c program with program arguments is shown in the following figure.

```
Program to run is /primeWithSubs.c 1001 3000 100
```

/primeWithSubs.c | Open | Run | Program Arguments: 1001 3000 100 | Screen Capture

```
 1 drop("main");drop("__exit_value");drop("__argc");drop("__argv");drop("isprime");
 2 int isprime(int start,int end,int range)
 3 {
 4   int n,lastp,lim,sn,t1,k,p,mcount=0,phold,pc=0;
 5   n = start; lastp=n; lim = end;
 6   if (n%2==0) n++;n-=2; sn=n;
 7   printf("Finding primes between %, and %,\n\n",n,lim);
 8   t1=sysTime();
 9   while (n<lim)
10   {
11     k = 3; p = 1; n = n + 2;
12     while ((k * k <= n) && p)
13     {
14       p = n%k; mcount++; k = k + 2;
15     }
16     if (p)
17     {
18       pc = pc + 1; phold=n;
19       if (pc%range==0)
20       {
21         printf("prime # %5, is %7, in this range %5.2f %% are primes\n",pc,n,(float)range*100.0/(n-lastp));
22         lastp=n;
23       }
```

```
Global Level Outputs

Output from main( )

-> Finding primes between 999 and 3,000

-> prime #   100 is   1,721 in this range 13.89 % are primes
-> prime #   200 is   2,503 in this range 12.79 % are primes
->
Found 263 primes
-> Last prime was 3,001

-> It took 1.716 seconds to process
-> 8,362 loops were performed, that's 4,872 per second

Program returned 263, free memory is 84812
```

Program tester.c illustrates the built in cat and ls functions. And the program output is shown in the following figure.

```
cat("/tester.c", ...)
   1 drop("main");
   2 //setConsoleOn(1);TFTsetTextSize(1);
   3 ls();
   4 printf("\n");
   5 cat("/tester.c",1);
   6
   7 void main()
   8 { printf("Program Done\n");}
```

```
File name Size
/arduinoi.jpg              19228 bytes
/cursive.hrsh               4379 bytes
/data/APname.dat              10 bytes
/debugPointers.c             425 bytes
/default.c                   553 bytes
/desc.txt                   1159 bytes
/drawing.c                  1782 bytes
/ESP32.jpg                 28113 bytes
/example.c                   579 bytes
/favicon.ico                1150 bytes
/futural.hrsh               3018 bytes
/futuram.hrsh               5282 bytes
/greek.hrsh                 3173 bytes
/hershey.c                  1723 bytes
/ltr.hrsh                   5234 bytes
/mini.jpg                  19478 bytes
/misLogo.jpg               21355 bytes
/misLogoAP.jpg             24862 bytes
/nested.c                    229 bytes
/nestedDebug.c               297 bytes
/prime.c                    1110 bytes
/Primes.c                    310 bytes
/primeWithSubs.c            1297 bytes
/scrollText.c               2501 bytes
/tester.c                    139 bytes
/usefull.h                  1436 bytes
/weather.jpg               34601 bytes
/WeMoLogo.jpg              13587 bytes
/WIFIname                      9 bytes
/WIFIpass                      9 bytes
/wthr240.jpg               21393 bytes
/data/WIFIname.dat            13 bytes
/data/WIFIpass.dat            12 bytes
->
```

The cat(filename, show line numbers) function has its output in a sizeable text area while the ls() function shows its output in the normal output table without the leading '-> ' normally presented with print(f) outputs. tester.c has lines commented out that are used if you want to also direct program outputs to a TFT color graphic display.

The programs

- drawing.c
- example.c
- Hershey.c

- scrollText.c
- tester.c

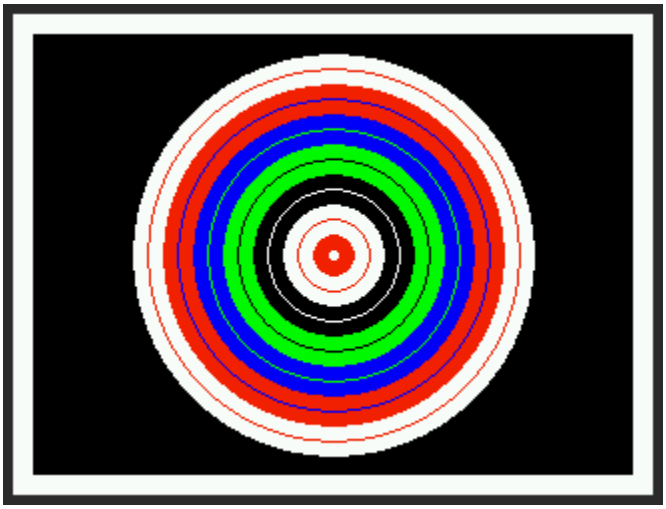
Are used with the TFT display and are summarized in the following paragraphs.

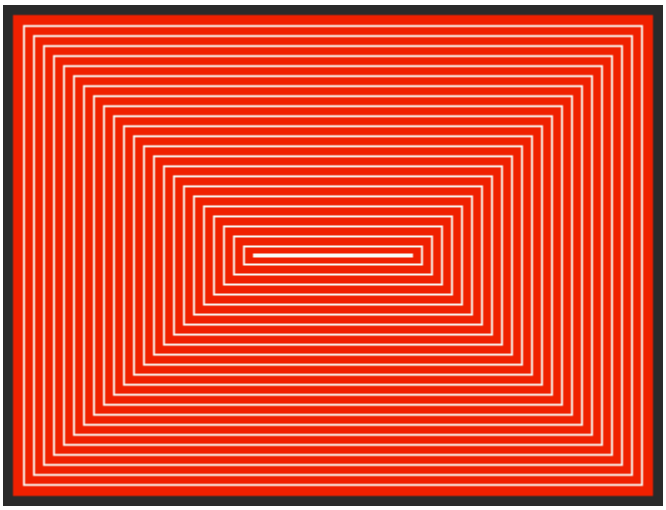
When using a 320x240 TFT it will display the following after reset.

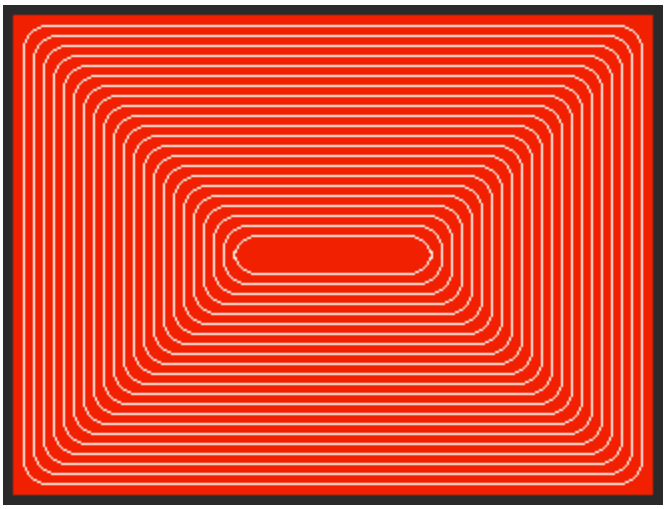

This display is a harmless advertisement for Micro Image Systems and the *book Building Blocks for IOT with your PC and WiFi Peripherals.* It also shows that the ESP32 is running in station mode with an IO address to 192.168.0.21 – your IP address will likely be different. If the ESP32 is running in AP mode it will show the display shown in the following figure.

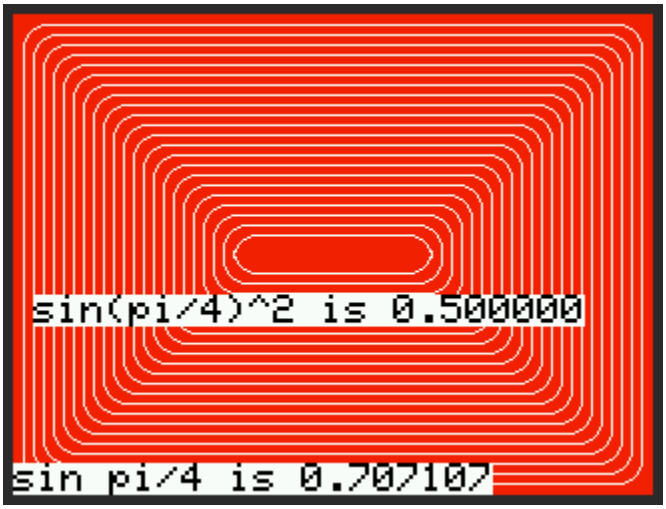

When in AP mode the Network Name will be ESP32PICOC but you can use the Settings page available from the Admin toolbar to change the name.

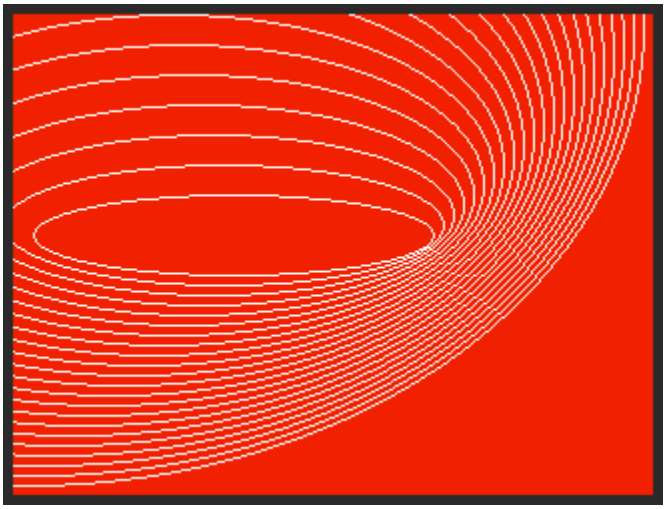
drawing.c – Produces some graphics drawings and then goes through a demo of the Hershey fonts built into the Interpreter by the author along with a slide show of images that are loaded on the ESP32. The drawing program calls screenCapture after each major display change and this will capture the TFT display images using the Processing platform and the Processing sketch ScreenCapture.pde available from the following URL https://github.com/rlunglh/IOT_with_your_PC/blob/master/ScreenCapture.pde You will need the Processing environment to run the ScreenCapture.pde file and this can be downloaded and installed using the instructions at https://processing.org/tutorials/gettingstarted/ The supplied ScreenCapture.pde sets the correct baud rate and is set to use your second enumerated COMM port. If you look at Device Manager and see only one device under Ports then you will need to change the Processing sketch line “int serial port = 1;” to “int serial port = 0;” The Processing ScreenCapture sketch will no run if the Arduino IDDE Serial Monitor is running for your ESP32. Also, when ScreenCapture initially starts, it resets the ESP32. Therefor just get the ScreenCapture.pde running and then proceed with getting you programs running in the Interpreter. The Screen Capture button on the ESP32Proram web pages will trigger a TFT Screen Capture and the function screenCapture() will trigger a capture if placed within your programs. The following figures are the screen captures obtained automatically when running the drawing.c program.

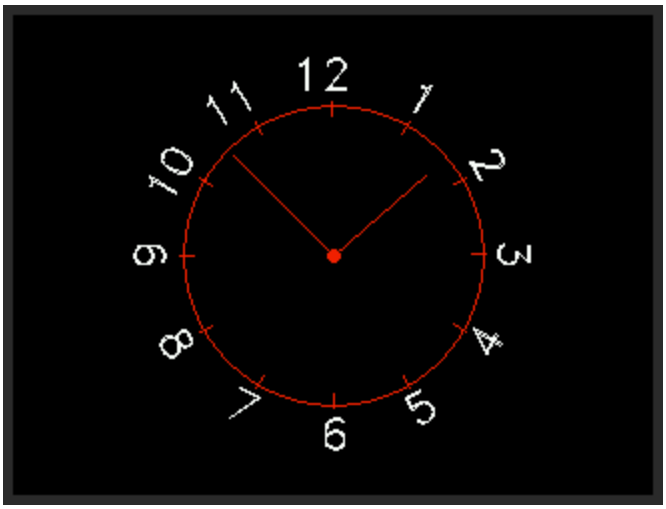







sin(pi/4)^2 is 0.500000
sin pi/4 is 0.707107




12
1
2
3
4
5
6
7
8
9
10
11


Hi
World
Hi
Hello World

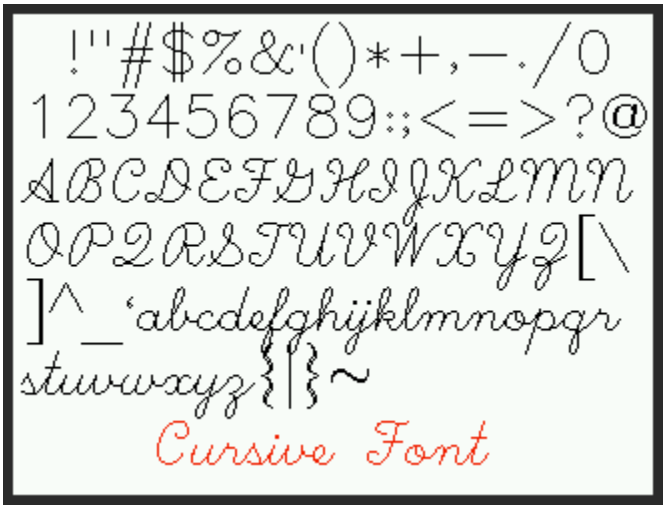

Black On White???
White On Black???
Christmas Anyone???
Hello World

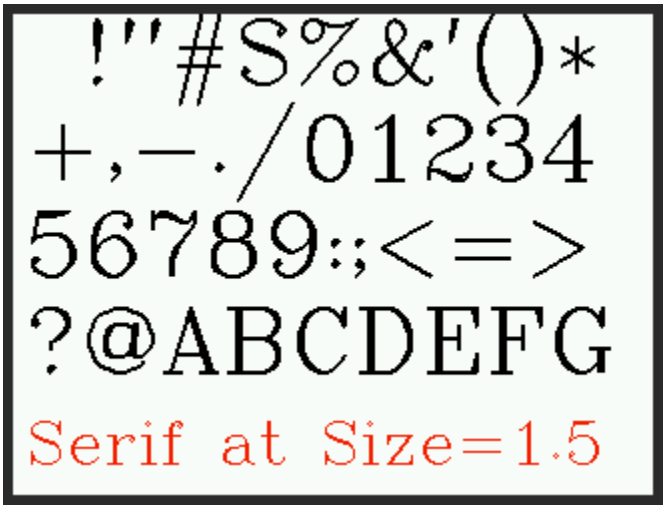

!"#$%&'()*+,-./0
123456789:;<=>?@
ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^
_`abcdefghijklmno
pqrstuvwxyz{|}~
Serif Font

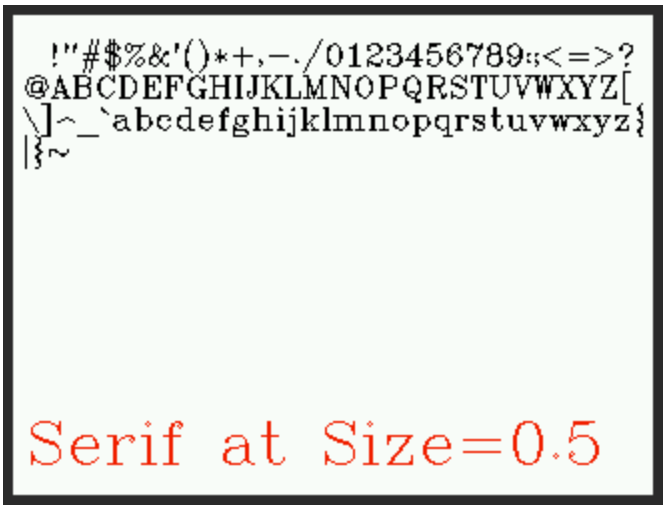

!"#$%&'()*+,-./0
123456789:;<=>?@
ABCDEFGHIJKLMNOP
QRSTUVWXYZ[\]^_`a
bcdefghijklmnopqrs
tuvwxyz{|}~
Sans Font

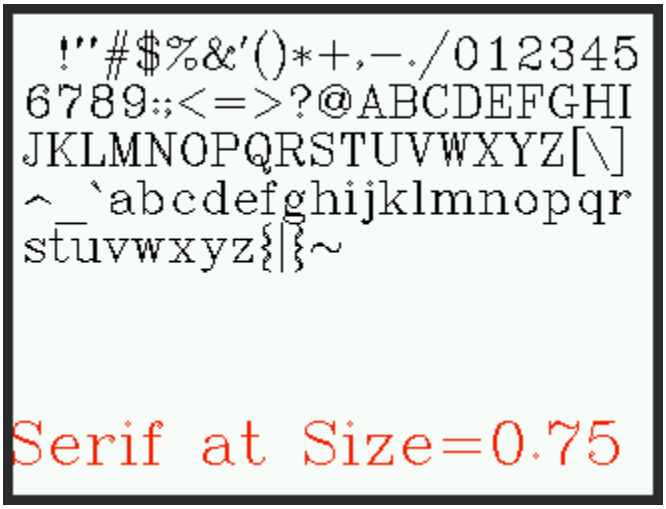

!"#$%&'()*+,-./0
123456789:;<=>?@
ABCDEFGHIJKLMNOP
QRSTUVWXYZ[\]^_`
abcdefghijklmnopq
rstuvwxyz{|}~
Sans Font Bold


!"#$%&'()*+,—./0
123456789:;<=>?@
ΑΒΧΔΕΦΓΗΙ·ΚΛΜΝΟΠ
ΘΡΣΤΥ°ΩΞΨΖ[\]^_‘αβ
χδεφγηι×κλμνοπϑρ
στυ÷ωξψζ{|}~
Greek Font

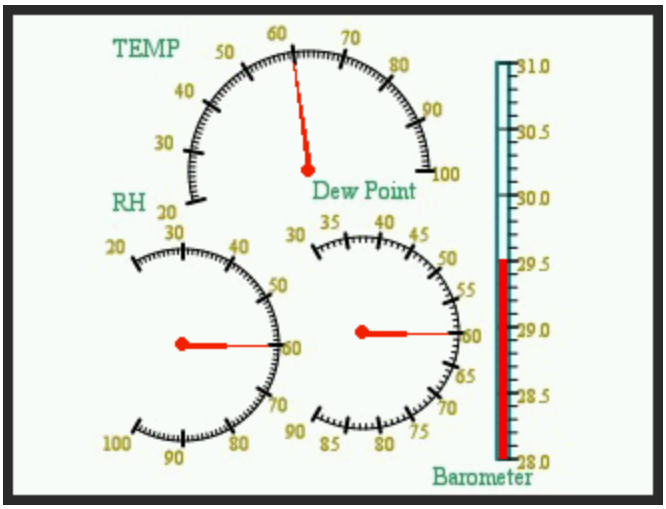

!"#$%&'()*+,—./0
123456789:;<=>?@
𝒜ℬ𝒞𝒟ℰℱ𝒢ℋℐ𝒥𝒦ℒℳ𝒩
𝒪𝒫𝒬ℛ𝒮𝒯𝒰𝒱𝒲𝒳𝒴𝒵[\
]^_‘abcdefghijklmnopqr
stuvwxyz{|}~
Cursive Font


!"#S%&'()*
+,—./01234
56789:;<=>
?@ABCDEFG
Serif at Size=1.5


!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[
\]^_`abcdefghijklmnopqrstuvwxyz{
|}~
Serif at Size=0.5

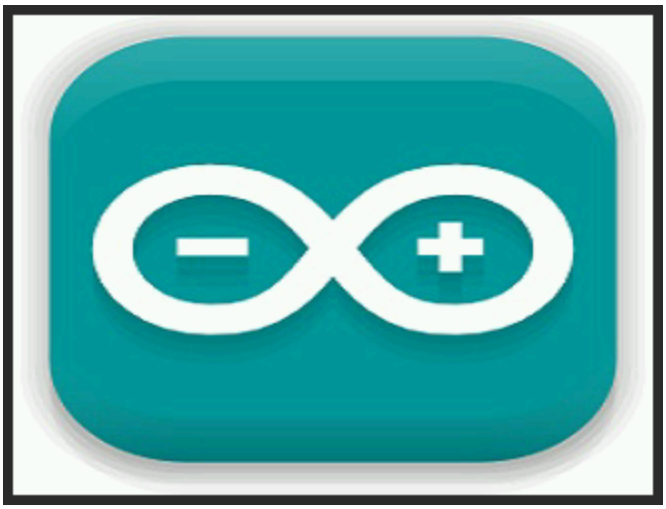

!"#$%&'()*+,-./012345
6789:;<=>?@ABCDEFGHI
JKLMNOPQRSTUVWXYZ[\]
^_`abcdefghijklmnopqr
stuvwxyz{|}~
Serif at Size=0.75


ABCDEWX


TEMP
Dew Point
RH
Barometer


μIS
Micro
Image
Systems
picoc Interpreter


WeMos




3V3 D8 D7 D6 D5 D0 A0 RST
RESET
D1 mini
5V G D4 D3 D2 D1 RX TX


ESP-WROOM-32
RST

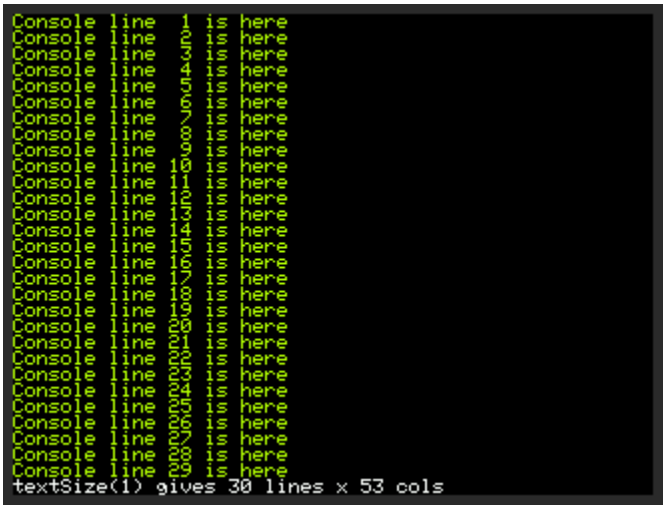

Even though all the displays presented drawing.c have been shown in the preceding figures, you should watch it run if you have a TFT display because there are slow stroke drawings of the Cursive font and That's All Folks that are demonstrative regarding the drawing of the Hershey fonts.

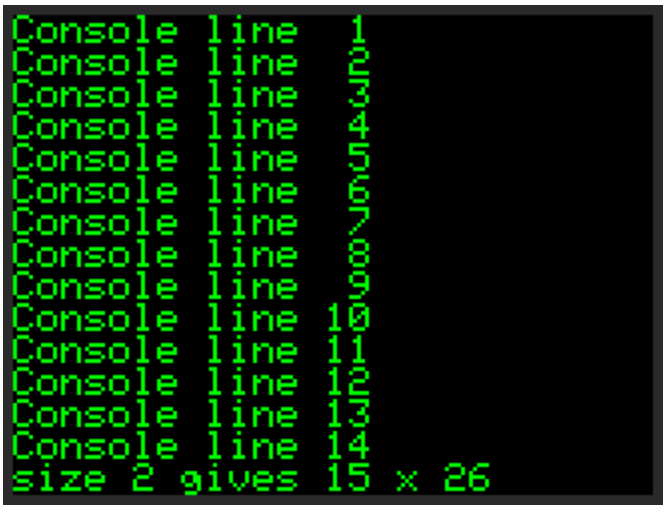
> Years ago (1977) I was introduced to a set of font definitions using vectors produced by Dr. Allen Vincent Hershey at the Naval Weapons Laboratory in 1967. Hence the name Hershey Fonts. These font definitions are quite well done and include serif and san serif fonts along with Greek and cursive fonts that are at least entertaining.

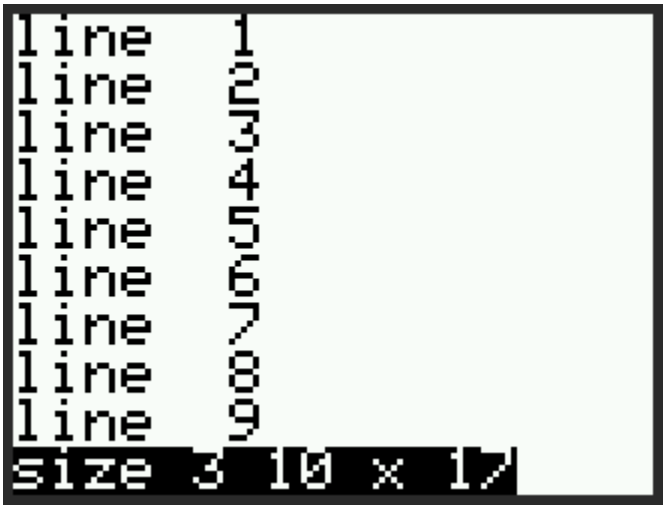
example.c – was shown earlier without using the TFT display. This time uncomment the TFTsetTextSize(1) , setConsoleOn(2) and Debug() lines in the program Save it and click Run giving the cat function and program outputs with console like output on the TFT display. A screen capture following program completion is shown in the following figure.

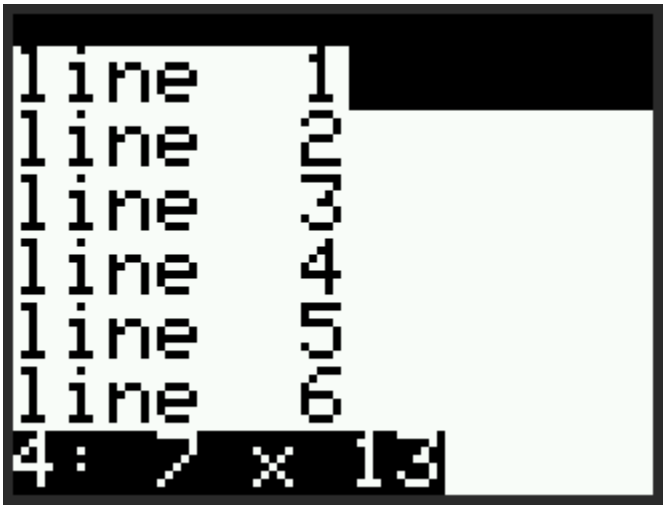

hershey.c – this program illustrates most of the Hershey built in functions and shows how to Invert the display for visual effects.

scrollText.c – this program does what the name implies and illustrates the scrolling of console output directed to the TFT display. A screenCapture of the TFT screens is shown in the following figures.

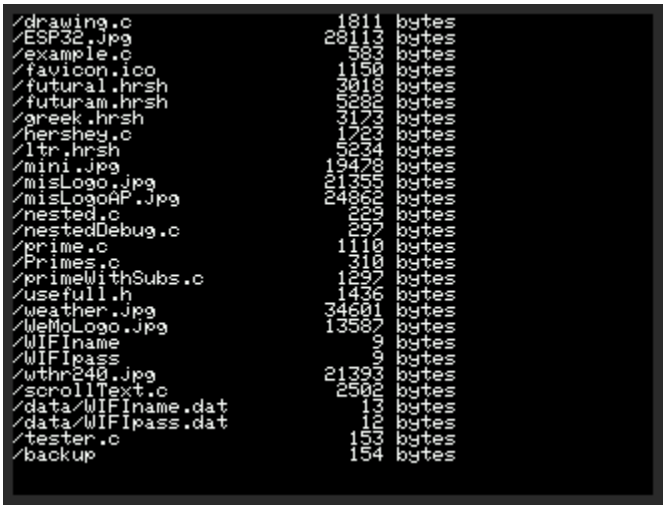


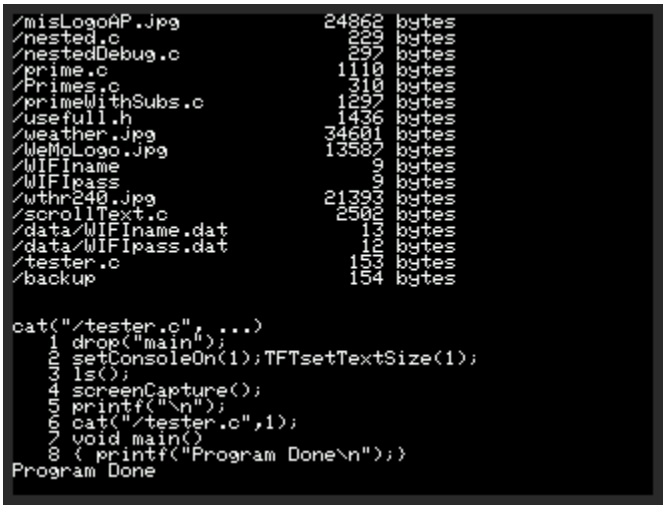

Console line 1
Console line 2
Console line 3
Console line 4
Console line 5
Console line 6
Console line 7
Console line 8
Console line 9
Console line 10
Console line 11
Console line 12
Console line 13
Console line 14
size 2 gives 15 x 26


line 1
line 2
line 3
line 4
line 5
line 6
line 7
line 8
line 9
size 3 10 x 17


line 1
line 2
line 3
line 4
line 5
line 6
4: 7 x 13


tester.c – this program simply runs the cat() function. If you uncomments the line "//setConsoleOn(1);TFTsetTextSize(1);" add screenCapture(); after the LS(); line, and then Save Run you get the following output.



If you are going to be using the ESP32 in the Arduino environment it makes sense to have the analog, digital, and PWM capabilities of the Arduino IDE also available in the Interpreter. The interpreter includes the following functions with arguments that are integers unless otherwise indicated.

- void pwmSetup(pin,frequency,range);  - set ESP32 pin to PWM output at frequency with values from 0 to range
- void servoAngle(pin,angle); set pin t0 PWM corresponding to float angle. Angle ranges from -90 - +90
- void pwmServo(pin,duty);  set pin to float duty dutycycle, duty ranges from 0.0 to 1.0
- void analogWrite(pin,value); set pin to value – user must account for the pins range set in pwmSetup to get the value they really want
- void delay(msec); delay msec milliseconds
- void digitalWrite(pin,value);
- int digitalRead(pin);
- void pinMode(pin,char * mode);   set pin to mode “INPUT” “OUTPUT” or “INPUTPULLUP”