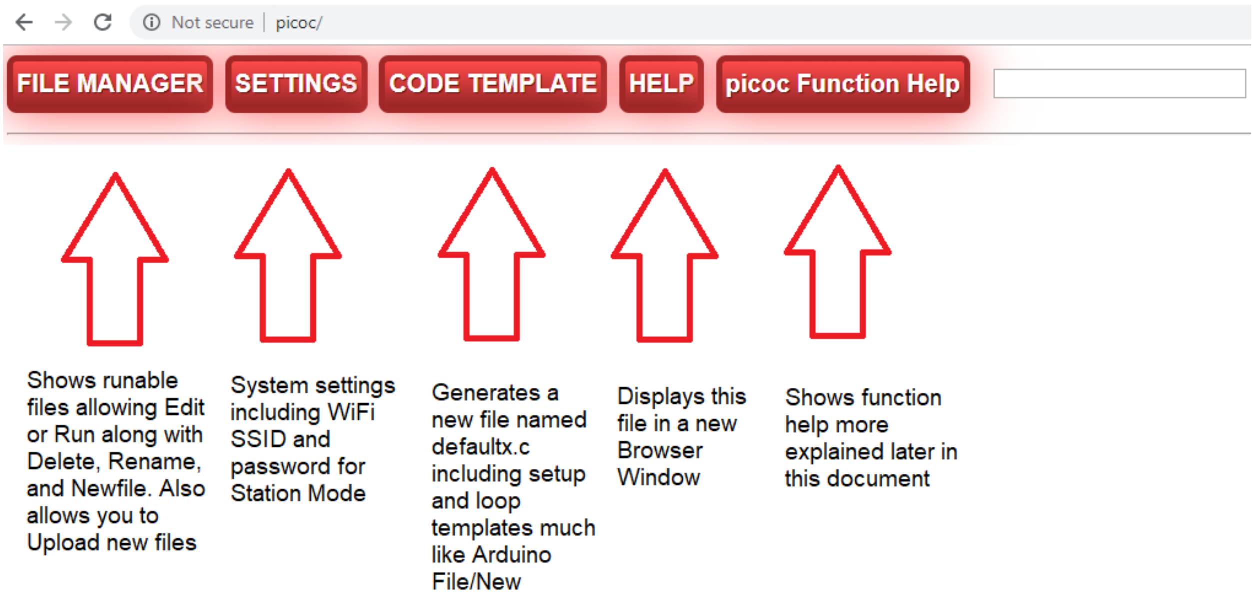
ESP32 picoc C Language Interpreter

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/

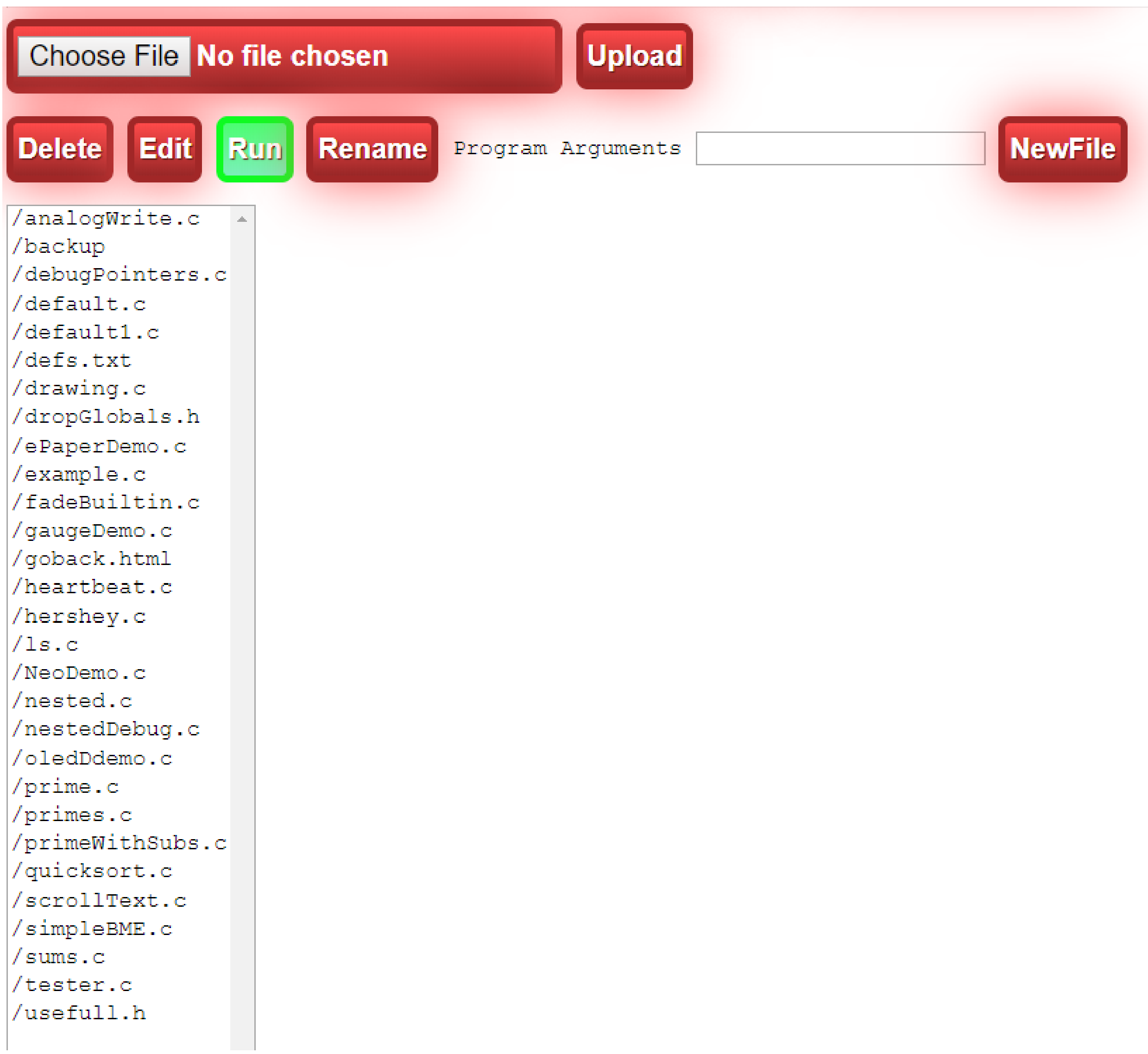
Once the Arduino IDE is installed just unzip the ESP32Program.zip file into your Arduino folder. You need to edit and upload some files to the ESP32 for the sketch to run. First look at the WIFIname.data file in the ESP32Program/data/data folder on your PC. This file supplies the SSID for the ESP32 to 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
```

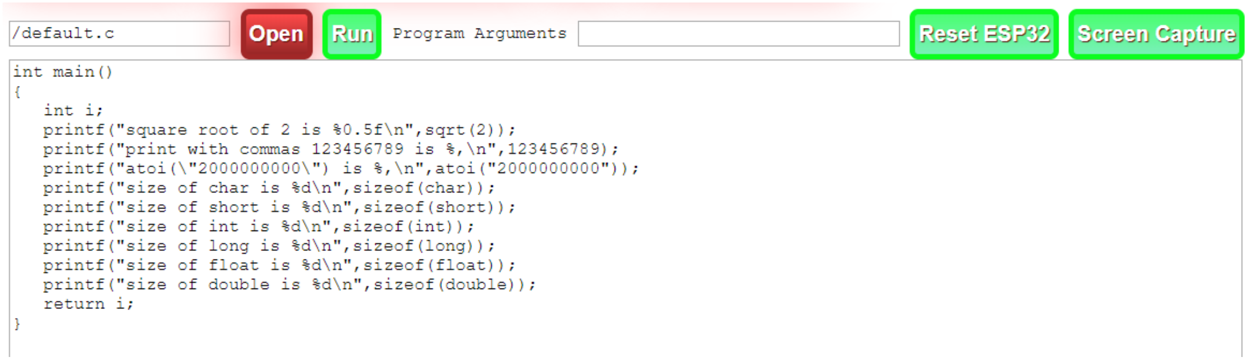
As shown in the code comments ALWAYS_STATION is defined to assure that the ESP32 will not enter Access Point mode. If a station connect fails the ESP32 will repeatedly restart and try to connect to the specified SSID until connection is successful. If you do have a problem connecting, make sure that 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:



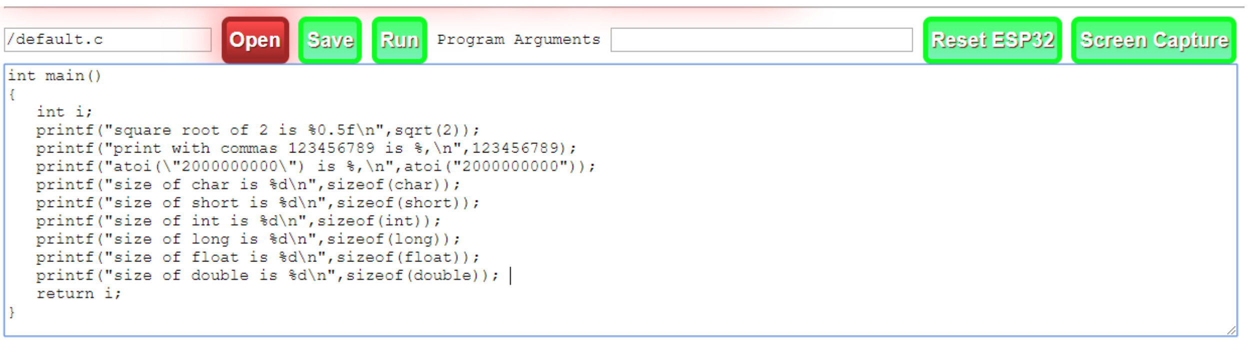
The following paragraphs give a quick tour of operation. Click the File Manager button giving the display shown below:



Double Click the /default.c line in the select 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. The file manager only shows program files and does not show .jpg .ico or .dat files. The ls() function in the Interpreter will show all files along with their file sizes.

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:

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.

Behind the screens, the interpreter collects all globally defines functions and variables and automatically deletes them after a program run so the program can be run repeatedly without having the interpreter complain that things such as main are already defined. This is accomplished by recording global scope object creation in the /DropGlobals.h file – when you ran the /default.c program, its contents were updated to what is shown below

```
drop("main");
drop("__exit_value");
```

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

- AnalogWrite.c
- debugPointers.c
- default.c
- example.c
- fadeBuiltin.c
- heartbeat.c
- ls.c
- nested.c
- nestedDebug.c
- prime.c
- Primes.c
- primeWithSubs.c
- quicksort.c
- sums.c
- tester.c
-

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

analogWrite.c – this program brightens and dims the built in LED on the ESP32, The program is set to use pin 2, if you use an ESP module other than the DH-ET line MiniKit, or the DOIT ESP32 DEVKIT that I have tested with, you may need to change the pin being used. For instance, on a HELTEC WiFi Kit 32 you need to use pin 25 which is accomplished by change the line `int LED=2;` to `int LED=25;`

debugPointers.c – an example program showing how the interpreter presents pointer reference data in the Debug mode.

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

example.c – shows some Debug output for nested loops.

fadeBuiltin.c – example program showing the loop() function in uninterrupted use. You can stop the loop by clicking the ResetESP32 button. This will restart the ESP32 and return you to the Edit view of the file being run

heartbeat.c – simply blinks the built_in LED in a heartbeat fashion

ls.c – lists the SPIFFS file system showing all file names and sizes.

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 **Reset ESP32** **Screen Capture**

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

```
Global Level Outputs
```

```
Output from main( )
```

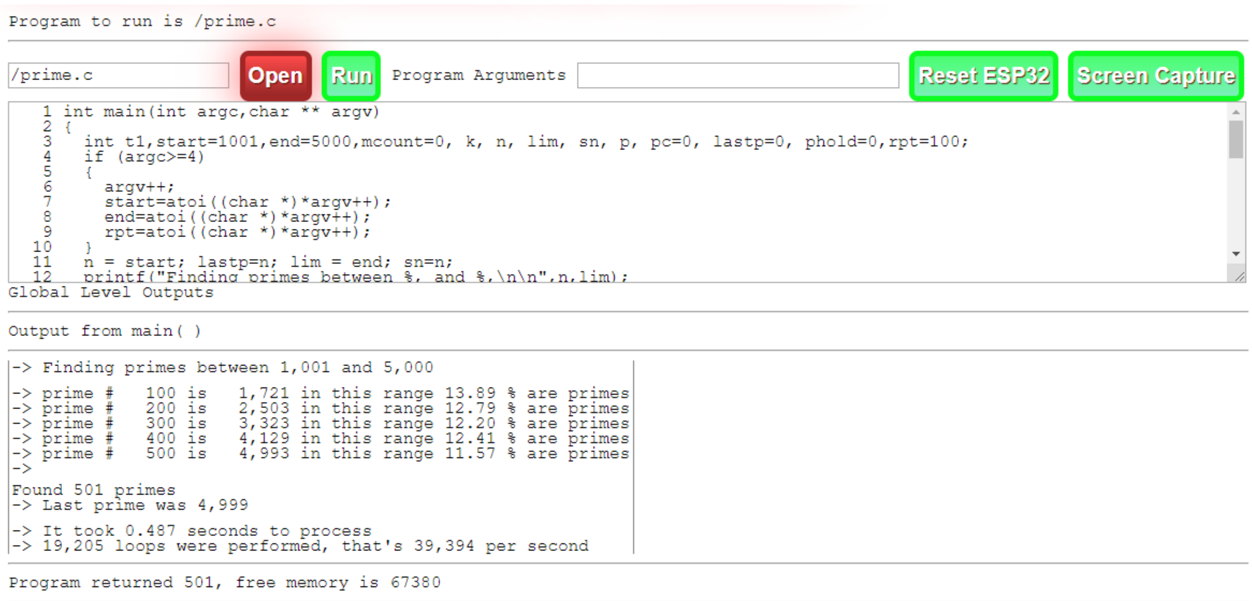
```
-> i*4+j = 0
-> i*4+j = 1
-> i*4+j = 2
-> i*4+j = 3
-> i*4+j = 4
-> i*4+j = 5
-> i*4+j = 6
-> i*4+j = 7
   8:  0  >    if (i==3) stopDebug();            |i : int 2 | j : int 4
   9:  0  >    for (int j=0;j<4;j++)             |i : int 2 | j : int 4
   9:  8  >    for (int j=0;j<4;j++)             |i : int 2 | j : int 4
  10:  0  >      printf("i*4+j = %d\n",i*4+j);   |i : int 2 | j : int 0
-> i*4+j = 8
   9: 21  >    for (int j=0;j<4;j++)             |i : int 2 | j : int 0
  10:  0  >      printf("i*4+j = %d\n",i*4+j);   |i : int 2 | j : int 1
-> i*4+j = 9
   9: 21  >    for (int j=0;j<4;j++)             |i : int 2 | j : int 1
  10:  0  >      printf("i*4+j = %d\n",i*4+j);   |i : int 2 | j : int 2
-> i*4+j = 10
   9: 21  >    for (int j=0;j<4;j++)             |i : int 2 | j : int 2
  10:  0  >      printf("i*4+j = %d\n",i*4+j);   |i : int 2 | j : int 3
-> i*4+j = 11
   9: 21  >    for (int j=0;j<4;j++)             |i : int 2 | j : int 3
  11:  0  >  }                                   |i : int 2 | j : int 4
   5: 19  >  for (int i=0;i<4;i++)               |i : int 2 | j : int 4
   6:  0  >  {                                   |i : int 3 | j : int 4
   7:  0  >    if (i==2) Debug();                |i : int 3 | j : int 4
   8:  0  >    if (i==3) stopDebug();            |i : int 3 | j : int 4
   8: 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 73320
```

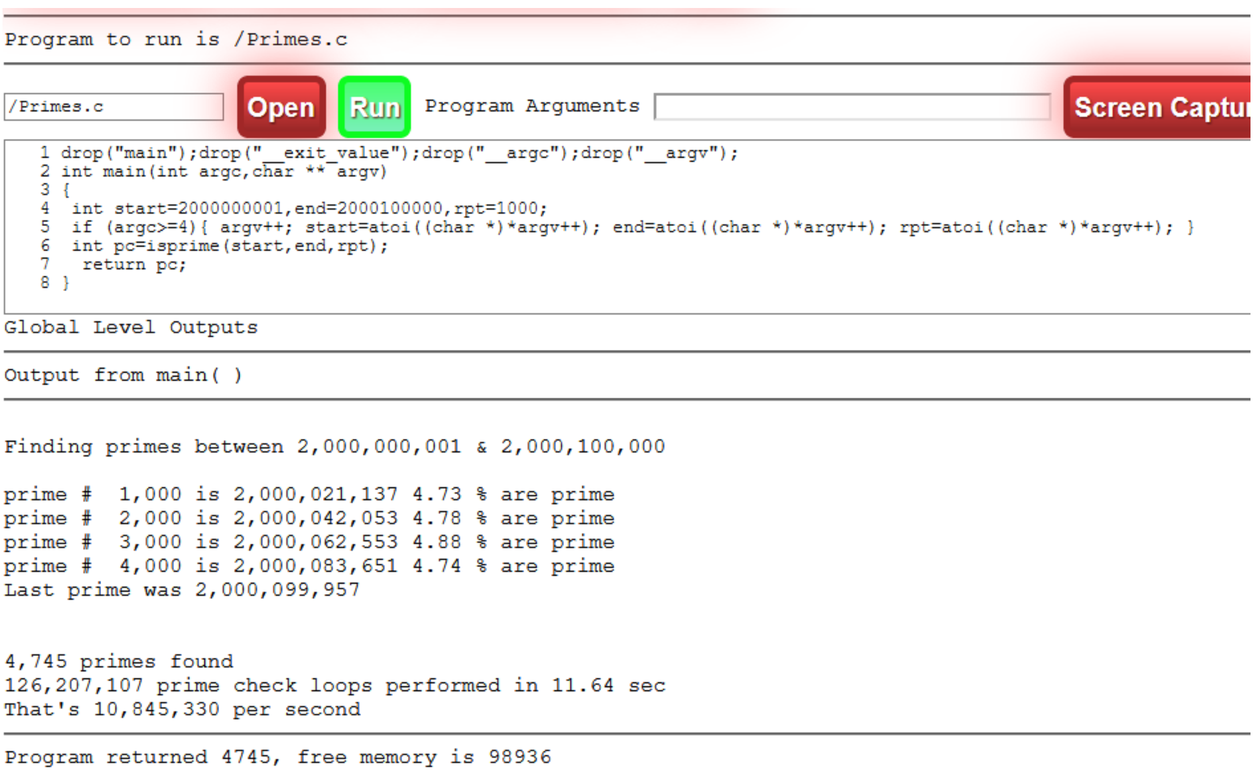
You see the program listing as in the program examples shown for default.c. Lines 3 and45 set watches for variables i and j. Line 7 initiates debug operation when i is equal to 2. Line 8 stops debug operation when i is equal to 3. The interpreter gives you a conditional debugging feature since the Debug() 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 7 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 8, 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 9:8 line shows that the for evaluation will next execute the int j=0 assignment. Line 10:0 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 8:13 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 range. Let's look at the output of prime.c as provided in the stock ESP32Progrma data directory.



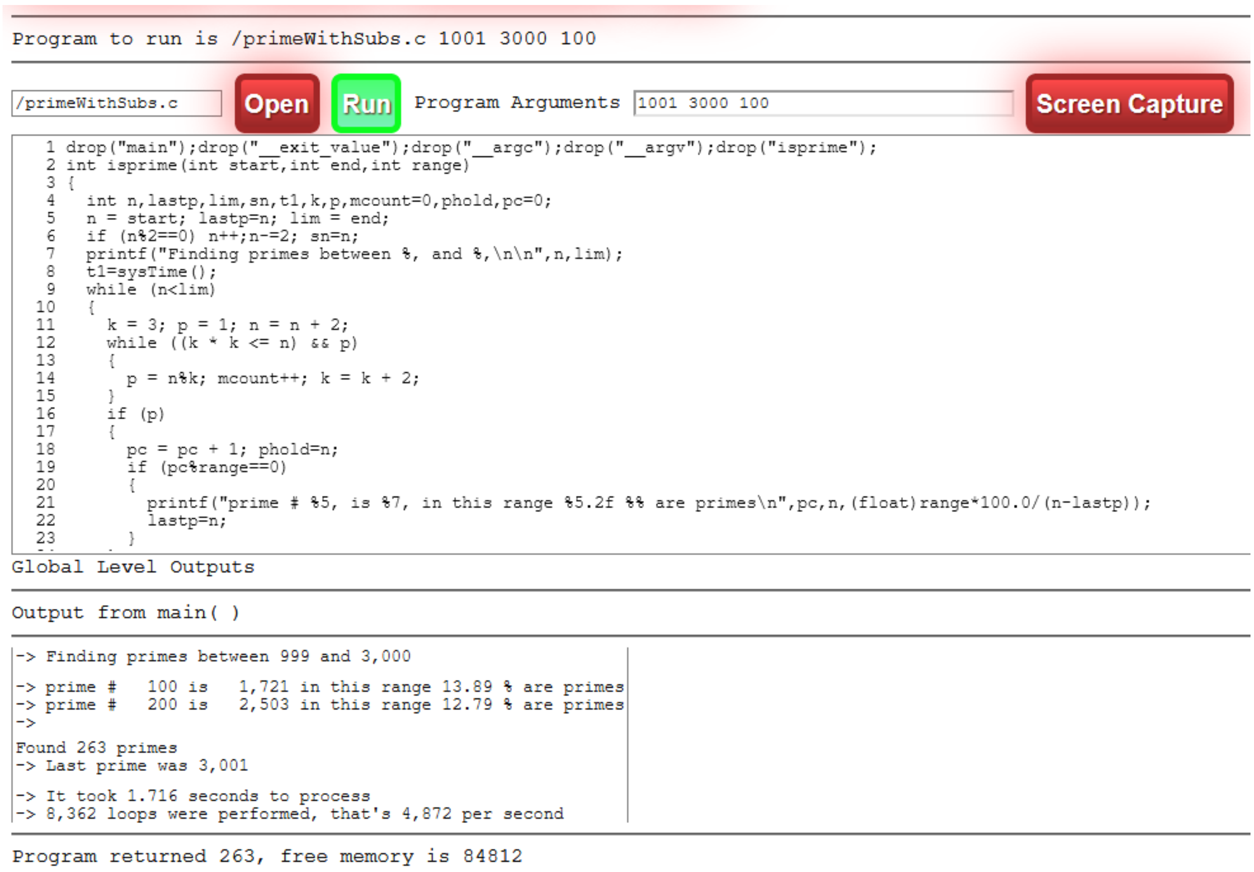
As you can see above, it lists the prime found after each 100 discoveries, and processes over a range of 1001 to 5000. The program also times its execution and counts how many while ((k*k<=n && p) loops are performed to allow reporting the loops complete per second, in this case, 5,011 per second. I acknowledge that the 5,001 loops per second including a multiplication, two compares, a modulo divide, an increment and an add but, to say the least, this 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.



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 thing 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 experiences 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 (not to mention recompiling and uploading for each Serial.print.. statement).

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.



Quicksort.c – quicksort is a classic sorting algorithms that includes recursion and is included in the samples to illustrate a more complex program logic task being put on the interpreter.

Sums.c – is another recursion algorithm that is easier to follow since it is a small piece of code. The output of a sums.c run with Debug and Watch is shown in the following figure.

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

```
/sums.c    Open  Run  Program Arguments
```

```
int sums(int top)
{
   if (top==1) return 1;
   return top+sums(top-1);
}
void main()
{
Debug();Watch("top");
   int start=10;
   int sum=sums(start);
   printf("Sums of 1 to %d are %d\n",start,sum);
stopDebug();
}
```

```
Global Level Outputs
```

```
Output from main( )
```

```
  8:  8  >Debug();Watch("top");
  9:  0  >   int start=10;                                     |top not defined|
 10:  0  >   int sum=sums(start);                              |top not defined|
  2:  0  >{                                                    |top : int 10
  3:  0  >   if (top==1) return 1;                             |top : int 10
  4:  0  >   return top+sums(top-1);                           |top : int 10
  2:  0  >{                                                    |top : int 9
  3:  0  >   if (top==1) return 1;                             |top : int 9
  4:  0  >   return top+sums(top-1);                           |top : int 9
  2:  0  >{                                                    |top : int 8
  3:  0  >   if (top==1) return 1;                             |top : int 8
  4:  0  >   return top+sums(top-1);                           |top : int 8
  2:  0  >{                                                    |top : int 7
  3:  0  >   if (top==1) return 1;                             |top : int 7
  4:  0  >   return top+sums(top-1);                           |top : int 7
  2:  0  >{                                                    |top : int 6
  3:  0  >   if (top==1) return 1;                             |top : int 6
  4:  0  >   return top+sums(top-1);                           |top : int 6
  2:  0  >{                                                    |top : int 5
  3:  0  >   if (top==1) return 1;                             |top : int 5
  4:  0  >   return top+sums(top-1);                           |top : int 5
  2:  0  >{                                                    |top : int 4
  3:  0  >   if (top==1) return 1;                             |top : int 4
  4:  0  >   return top+sums(top-1);                           |top : int 4
  2:  0  >{                                                    |top : int 3
  3:  0  >   if (top==1) return 1;                             |top : int 3
  4:  0  >   return top+sums(top-1);                           |top : int 3
  2:  0  >{                                                    |top : int 2
  3:  0  >   if (top==1) return 1;                             |top : int 2
  4:  0  >   return top+sums(top-1);                           |top : int 2
  2:  0  >{                                                    |top : int 1
  3:  0  >   if (top==1) return 1;                             |top : int 1
  3: 14  >   if (top==1) return 1;                             |top : int 1
 11:  0  >   printf("Sums of 1 to %d are %d\n",start,sum);     |top not defined
-> Sums of 1 to 10 are 55
 12:  0  >stopDebug();                                         |top not defined|
```

```
Program returned 4745, free memory is 69848
```

In the interpreter output, you can clearly see the value of top that is being passed the the sums function decreases from 10 to 1 and that at a value of 1 all the sums are returned and the total of Sums 1 to 10 is 55.

Tester.c – a simple program that illustrates the ls() and Cat() function outputs.

The programs

- drawing.c
- example.c
- hershey.c
- scrollText.c
- tester.c

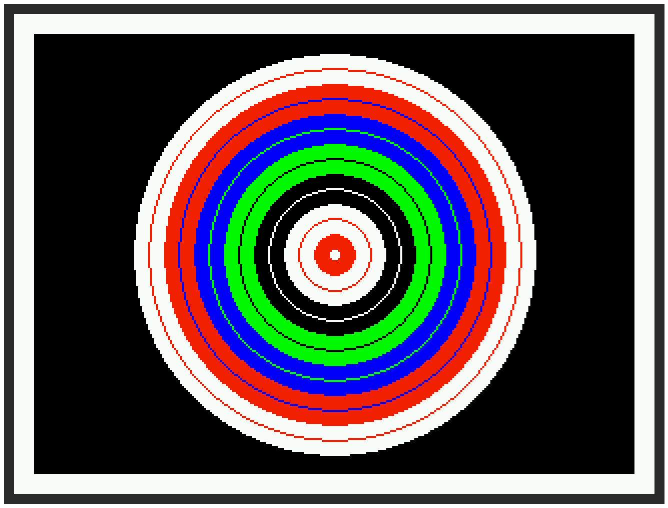
Are used with a 320x240 TFT display which is my favorite way of running the ESP32. These programs are summarized in the following paragraphs.

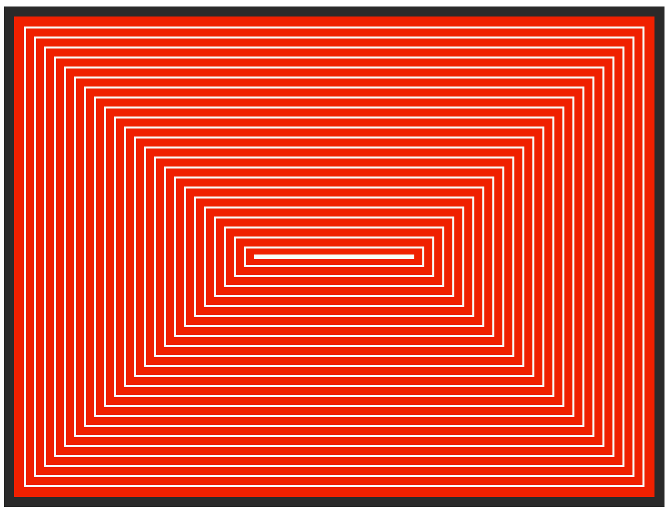
When using a 320x240 TFT, the ESP32 will display the following after reset.

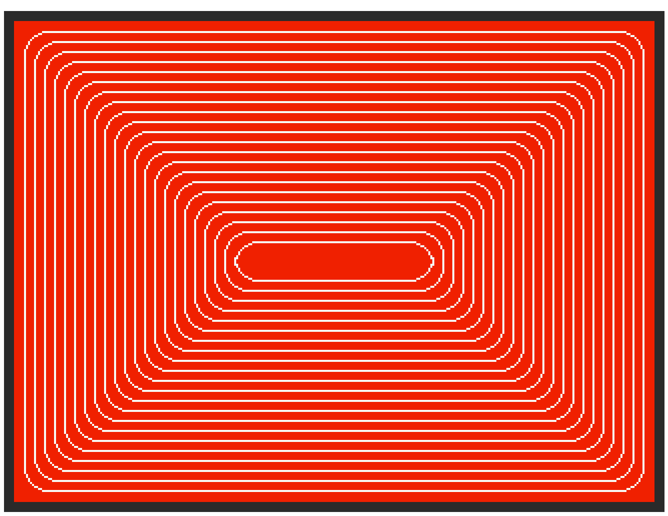
This display is a harmless advertisement for Micro Image Systems and the *book Building Blocks for IOT with your PC and WiFi Peripherals.* It also shows that the ESP32 is running in station mode with an IO address to 192.168.0.24 – your IP address will likely be different. If the ESP32 is running in AP mode it will show the display shown in the following figure.

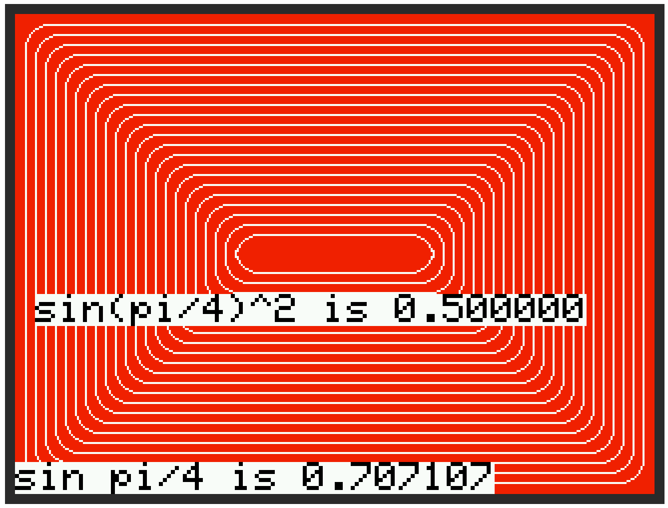
When in AP mode the Network Name will be ESP32PICOC but you can use the Settings page available from the Admin toolbar to change the name.

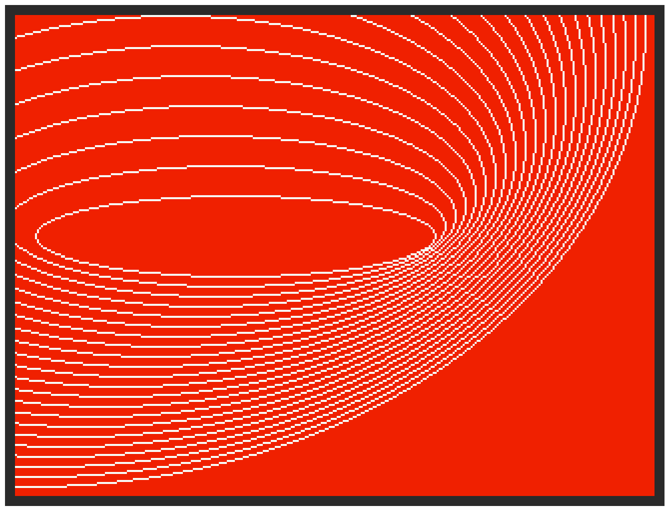
drawing.c – Produces some graphics drawings and then goes through a demo of the Hershey fonts built into the Interpreter by the author along with a slide show of images that are loaded on the ESP32. The drawing program calls screenCapture after each major display change and this will capture the TFT 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 not run if the Arduino IDE 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.

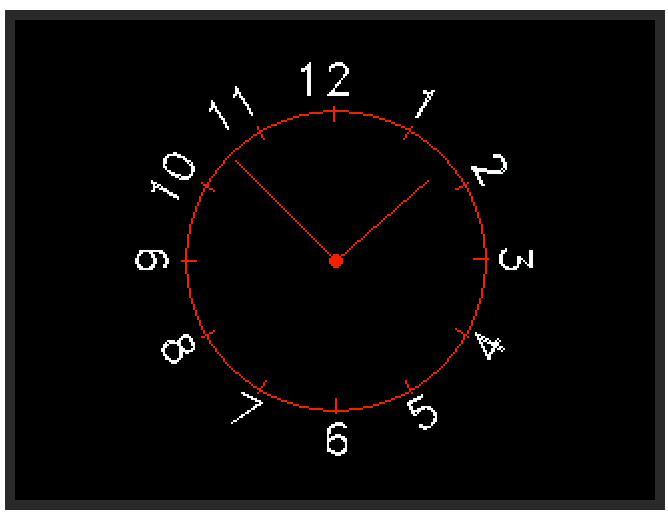




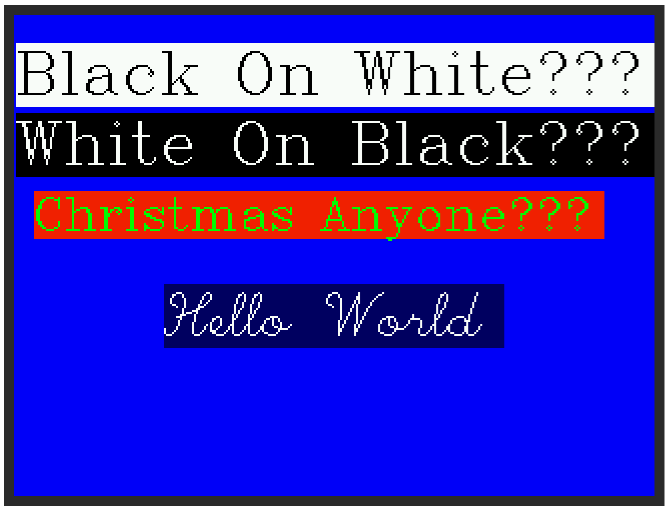



sin(pi/4)^2 is 0.500000
sin pi/4 is 0.707107




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{|}~

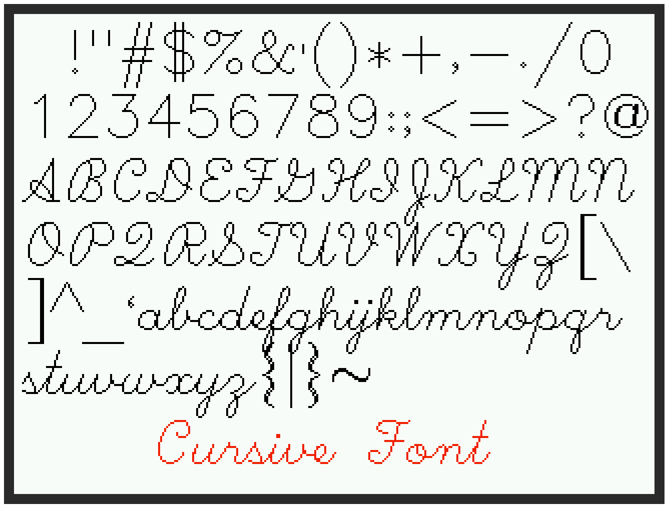
Sans Font

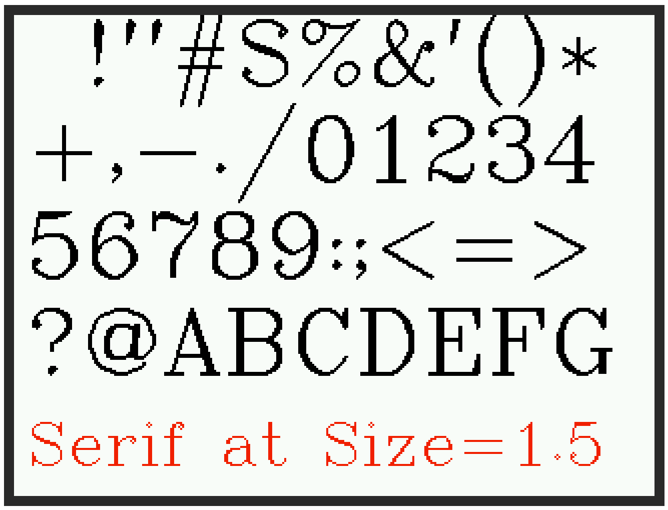
!"#$%&'()*+,−./0
123456789:;<=>?@
ABCDEFGHIJKLMNOP
QRSTUVWXYZ[\]^_`
abcdefghijklmnopq
rstuvwxyz⟨|⟩~

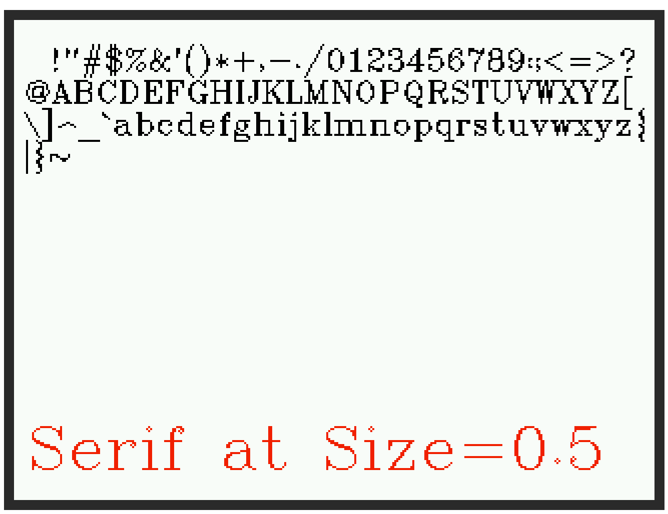
Sans Font Bold

!"#$%&'()*+,−./0
123456789:;<=>?@
ABXΔEΦΓHI·KΛMNOΠ
ΘPΣTϒ°ΩΞΨZ[\]^_‘αβ
χδεφγηι×κλμνοπϑρ
στυ÷ωξψζ{|}~

Greek Font

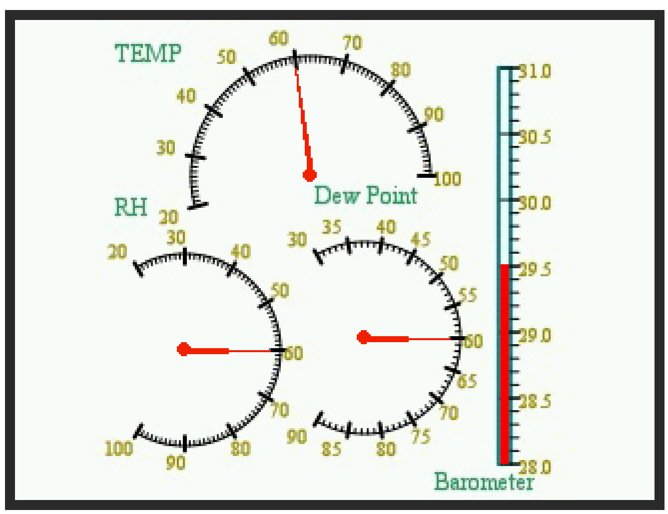

!"#$%&'()*+,-./0
123456789:;<=>?@
ABCDEFGHIJKLMN
OPQRSTUVWXYZ[\
]^_`abcdefghijklmnopqr
stuvwxyz{|}~
Cursive Font


!"#$%&'()*
+,-./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
20 30 40 50 60 70 80 90 100
RH
20 30 40 50 60 70 80 90 100
Dew Point
30 35 40 45 50 55 60 65 70 75 80 85 90
31.0 30.5 30.0 29.5 29.0 28.5 28.0
Barometer

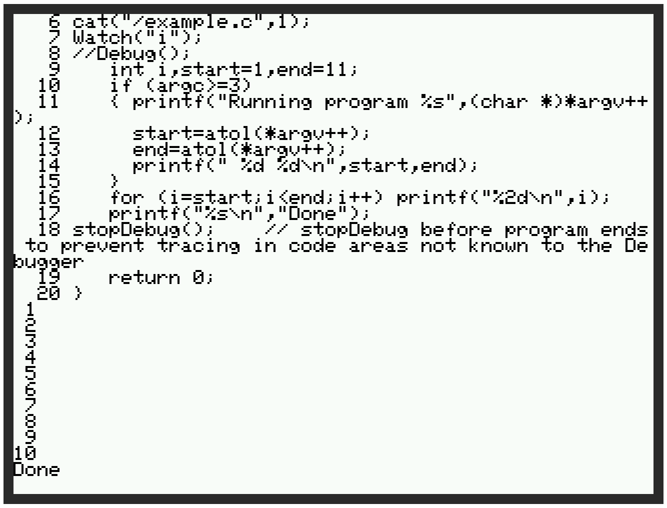
µIS Micro Image Systems

picoc Interpreter

WeMos

Even though all the displays presented by 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.

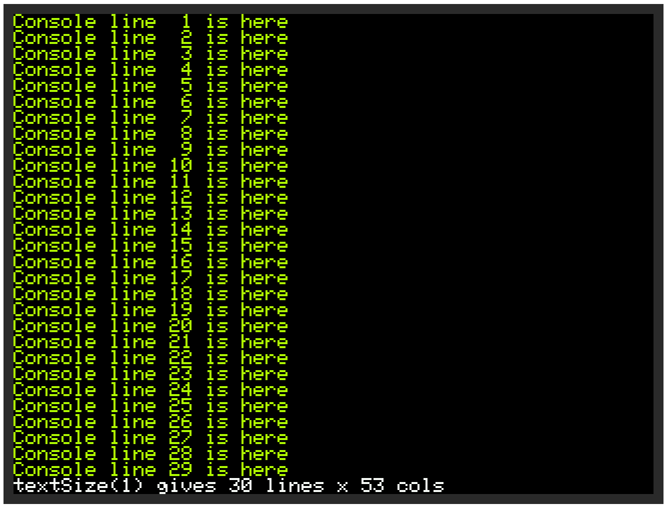
example.c – was shown earlier without using the TFT display. This time uncomment the TFTsetTextSize(1) , setConsoleOn(2) and Debug() lines in the program Save it and click Run giving the cat function and program outputs with console like output on the TFT display. A screen capture following program completion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

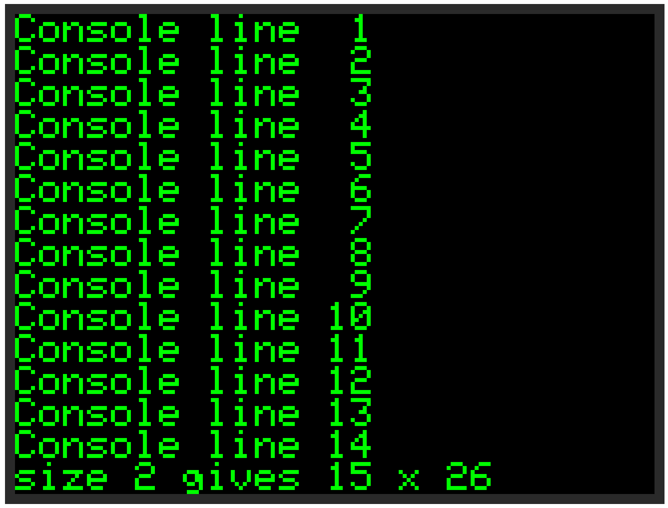


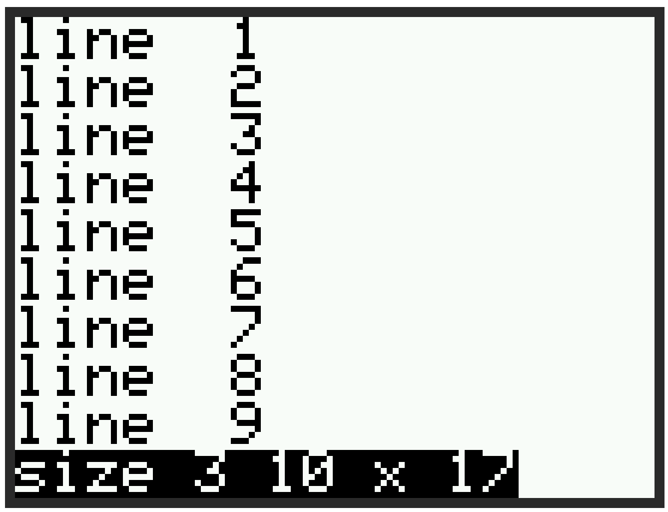
hershey.c – this program illustrates most of the Hershey built in functions and shows how to Invert the display for visual effects.

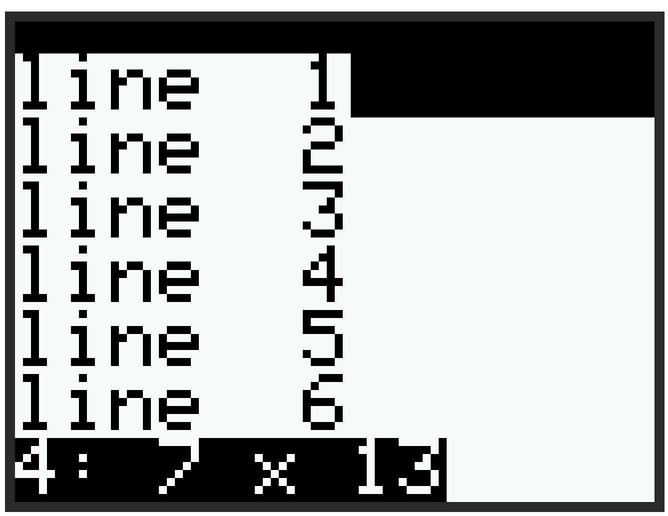
Years ago (1977) I was introduced to a set of font definitions using vectors produced by Dr. Allen Vincent Hershey at the Naval Weapons Laboratory in 1967. Hence the name Hershey Fonts. These font definitions are quite well done and include serif and san serif fonts along with Greek and cursive fonts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

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.

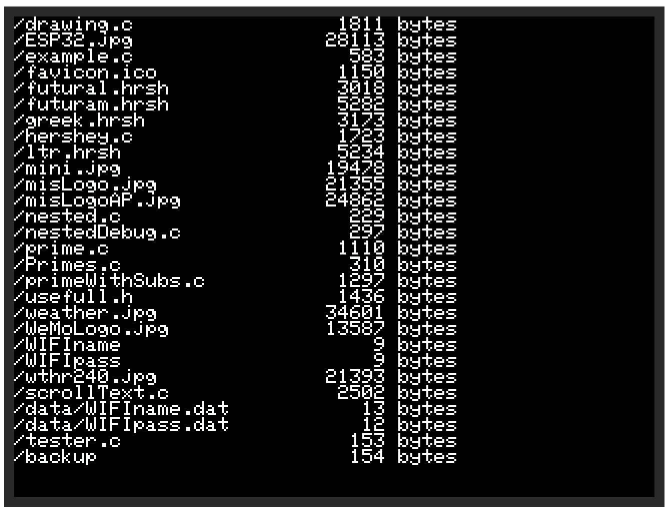


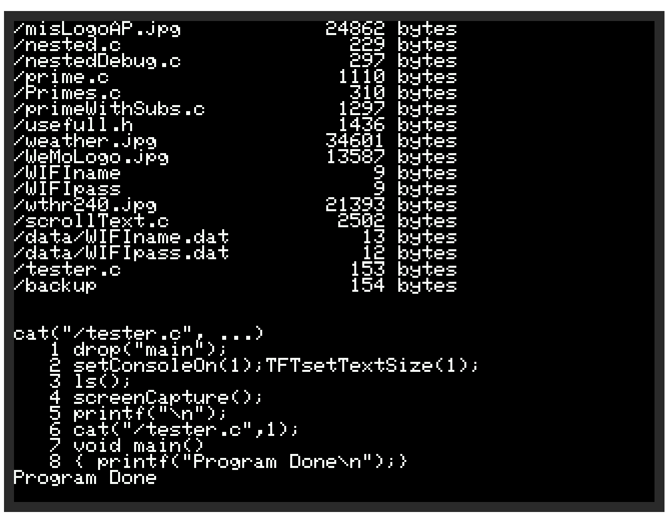

Console line 1
Console line 2
Console line 3
Console line 4
Console line 5
Console line 6
Console line 7
Console line 8
Console line 9
Console line 10
Console line 11
Console line 12
Console line 13
Console line 14
size 2 gives 15 x 26


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.





Pulse Width Modulation and AnalogWrite

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"

You can also get function references directly from the ESP32Program web pages. The “picoc Function Help” button along with the textbox to the right of the button are your gateway to function definitions. Say that you want to see all TFT_ functions, just put TFT-* in the text box and click the Picoc Function Help button. This will give you the following results.

Help for function TFT_*

---

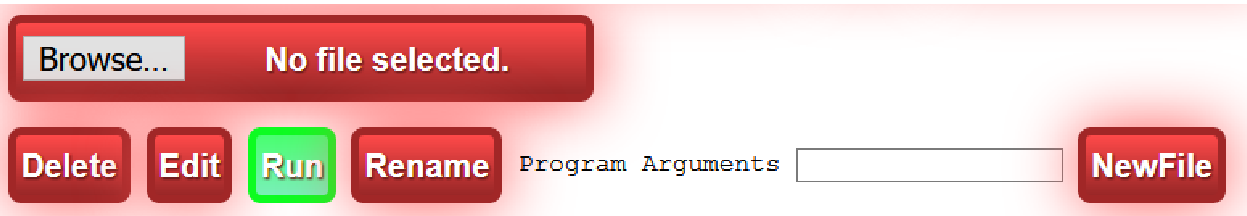
void TFT_draw(char *,int,int); filename,x,y -- must be a Jpeg file
void TFT_drawCircle(int,int,int,int); xc,yc,radius,color
void TFT_drawEllipse(short,short,short,short,short); xc,yc,radiusx,radiusy,color
void TFT_drawLine(int,int,int,int,int); x1,y1,x2,y2,color
void TFT_drawRect(int,int,int,int,int); x,y,width,height,color
void TFT_drawRoundRect(int,int,int,int,int,int); x,y,width,height,radius,color
void TFT_drawTriangle(int,int,int,int,int,int,int); x1,y1,x2,y2,x3,y3,color
void TFT_fillCircle(int,int,int,int); xc,yc,radius,color
void TFT_fillEllipse(short,short,short,short,short); xc,yc,radiusx,radiusy,color
void TFT_fillTriangle(int,int,int,int,int,int,int); x1,y1,x2,y2,x3,y3,color
void TFT_fillRect(int,int,int,int,int); x,y,width,height,color
void TFT_fillRoundRect(int,int,int,int,int,int); x,y,width,height,radius,color
void TFT_fillScreen(int); color
int TFT_Gauge(float xc,float yc,float sang,float eang,float radius,float sval,float eval,int divisions,float increments,int color,char * fmt,char * valueFmt);
void TFT_Gauge_draw(int); gauge#
void TFT_Gauge_drawDanger(int,float,float,int); gauge#,startValue,endValue, color
void TFT_Gauge_drawDangerByValue(int,float,float,int); gsuge#,startValue,endValue, color
void TFT_Gauge_dropGauges(); frees all TFT_Gauge storage
void TFT_Gauge_setPosition(int,float); gauge#,value
int TFT_HbarGraph(float x,float y,int width,int height,float sval,float eval,int divisions,float increments,int color,char * fmt,char * valueFmt
void TFT_HbarGraph_draw(int); gauge#
void TFT_HbarGraph_dropGauges(); frees all HbarGraph gauge storage
void TFT_HbarGraph_setPosition(int,float); gauge#,value
void TFT_invertDisplay(int);, pass 0 for normal 1 for inverse
void TFT_print(char *); char string to print
void TFT_pushRect(int,int,int,int,short *); startx,starty,width,height
void TFT_readRect(int,int,int,int,short *); startx,starty,width,height

void TFT_setCursor(int,int); x,y y increases from 0 at top of display
void TFT_setTextColor(int,int); foreColor,backColor
void TFT_setTextSize(int); size
int TFT_VbarGraph(float x,float y,int width,int height,float sval,float eval,int divisions,float increments,int color,char * fmt,char * valueFmt
void TFT_VbarGraph_draw(int); gauge#
void TFT_VbarGraph_dropGauges(); frees all VbarGraph gauge storage
void TFT_VbarGraph_setPosition(int,float); gauge#,value

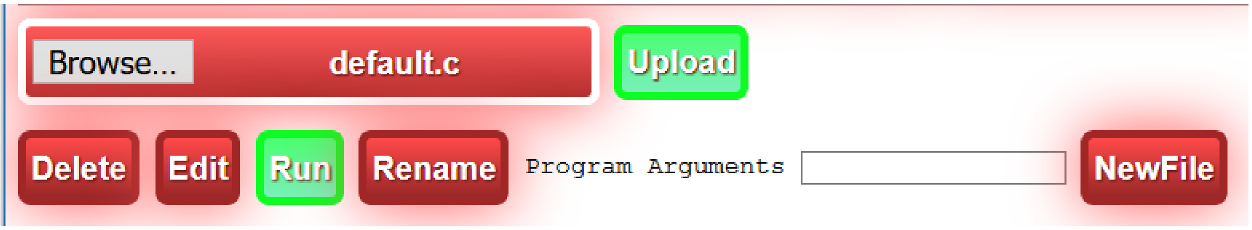
You can enter * to get all functions - displays the defs.txt file, a full name, an empty string – to get the defs.txt file shown in a window with navigation, or as done above a string followed by * to do a wildcard search. To return to your previous page just use the Go Back button.

Miscellaneous Buttons on the File Manager Page

The Buttons on The File Manager page appear as shown in the following figure.



The top button, Browse... allows you to select a file to be uploaded to the ESP32. Once you click the Browse... button and select a file to upload, the Upload button will be made visible. As shown below.



When you click Upload the selected file will be uploaded to the uploads folder on the ESP32 SPIFFS and you will be directed to an edit page for the uploaded file. **NOTE *Only text type files can be uploaded in this manner.*** If you want to add other file types to the ESP32 SPIFFS, put them in the ESP32Program's data folder and use the ESP32 Sketch Data Upload Tool. Be advised that any files in the SPIFFS that are not in the data directory, or that you have changed since the last upload, will be lost. I carefully copy and paste between the browser window and the PC's editor when I change a file on the ESP32 to prevent loosing changes if I use the Arduino ESP32 Sketch Data Upload function.

As mentioned above, another means of transferring content from your PC to the ESP3 2 is to use the Edit file and use copy and paste between a file viewer/editor on your PC (my favorite is Notepad++) and the browser Edit page.

The Delete button does exactly what you'd expect it deletes the selected file in the file list from the ESP32 SPIFFS.

The Edit and Run buttons have already been discussed and their operation is intuitive.

The rename button lets you rename/move files on the ESP32 SPIFFFS. Its dialog is shown in the following figure.

```
Old Name
/uploads/drawing - Copy.c
New Name
/uploads/drawing - Copy.c
```

Change Name

The Change Name button will rename the file from Old Name to New Name.

The New File button creates an empty file with the name newfilex.txt and opens it in the edit view. If you do not save the file, for instance you click the File Manager button in the edit view before a Save, the file will not appear in the file system. Each New File button click will create files with higher sequence numbers than what already exists in the file system.

The Program Arguments filed lets you pass values to a program if it is coded with a main function following the main(int argc,char ** argv) {…} conventions. For instance the prime.c program accepts program arguments and its operation with Program Arguments is shown in two following figures.

```
/prime.c                     Open  Run  Program Arguments  1001 3000 100
```

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

```
/prime.c                     Open  Run  Program Arguments  1001 3000 100
```

```c
int main(int argc,char ** argv)
{
  int t1,start=1001,end=5000,mcount=0, k, n, lim, sn, p, pc=0, lastp=0, phold=0,rpt=100;
  if (argc>=4)
  {
    argv++;
    start=atoi((char *)*argv++);
    end=atoi((char *)*argv++);
    rpt=atoi((char *)*argv++);
  }
  n = start; lastp=n; lim = end; sn=n;
  printf("Finding primes between %, and %,\n\n",n,lim);
  t1=sysTime();
  while (n<lim)
  {
    k = 3; p = 1; n = n + 2;
    while ((k * k <= n) && p)
    {
      p = n%k; mcount++; k = k + 2;
    }
    if (p)
    {
      pc = pc + 1; phold=n;
```

```
Global Level Outputs

Output from main( )

-> Finding primes between 1,001 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 0.214 seconds to process
-> 8,359 loops were performed, that's 39,106 per second

Program returned 263, free memory is 69692
```

The program by default runs with start=1001, end=5000, and rpt=100. When at least 3 arguments are passed (making argc=4 since argv[0] if the program name) the code uses the parameters to fill in start, end, and rpt sequentially. The lines

```
if (argc>=4)
{
  argv++;
  start=atoi((char *)*argv++);
  end=atoi((char *)*argv++);
  rpt=atoi((char *)*argv++);
}
```

Illustrate how to use the argv values to set integers, they can just as easily be assigned as character string pointers i.e. char * by not using the atoi() function.

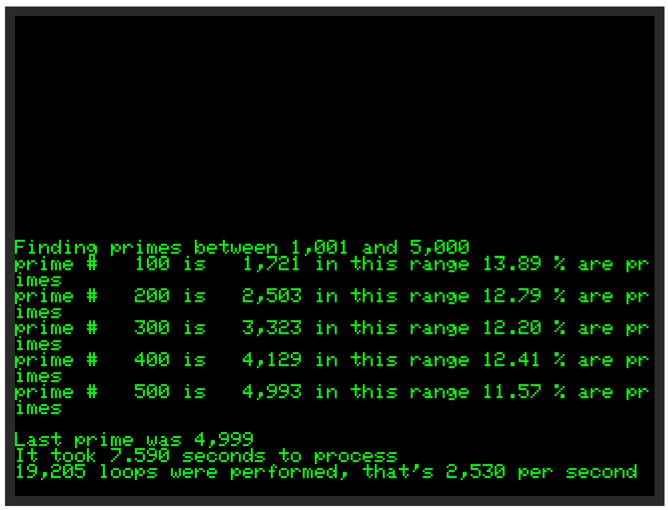
The Code Template and HELP buttons in the Admin toolbar are discussed in the following two paragraphs.

The Code Template button creates a simple program template much like the Arduino File/New button. The template includes main, setup, and loop functions as shown below:

```
void setup();
void loop();
int main(int argc,char ** argv)
{
  setup();
  //for (;;) loop();
  //  Uncomment line above to run your loop() function forever.
  //  When running loop forever,
  //  you'll get no response to web events
  //  unless you include doLoop();
  //  in your interpreted loop() function
 return 0;
}
void setup()
{
}
void loop()
{
  doLoop();
}
```

The program’s main function is set to return an int and uses argc and argv inputs which are gathered from the Program Arguments textbox when the program is run. Even though argc and argv are declared, you don’t need to make any use of them. The main function calls setup and will thereafter repeatedly call loop() if the `for (;;) loop();` line is uncommented. The template includes doLoop() in the loop function to allow the browser buttons to be active when the sketch is running. Note however, that if doLoop() is called, that program outputs to the console will stop since the browser session is closed when doLoop() is called. doLoop() exists to provide a means to stop programs that have continuous Loop operations since it enables the Screen Capture and ResetESP32 buttons to be used. There are two means of getting program outputs when running a loop() function continuously in a program. First, you can use the sprint(char *) function to send output to the Serial Monitor – I like to use Termite by CompuPhase since it supplies and RTS control that will also reset the ESP32.

Secondly, you can use a TFT, OLED, or ePaper display with the ESP32 and send output to those devices. Particularly useful is `void setConsoleOn(int);` 0 turns console output off for printf function, 1 turns console output on. So `setConsoleOn(1);` will send your program's printf output to a console on the TFT. As an example, if you add setConsoleOn(1); near the top of the prime.c program, Save the file, and click Run, the TFT output will appear as shown below.



A better example of using a continuousl running Loop() function including doLoop() is contained in the simpleBME.c program. This program uses a BME280 sensor that measures pressure, temperature, and relative humidity and communicates over a I2C serial interface. The interpreter includes the following functions for the BME280 sensor.

```
void BME_init(); must be called before using the BME280 sensor
float BME_readPressure(); returns barametric pressure from BME280 sensor
float BME_readRH(); returns relative humidity from BME280 sensor
float BME_readTemp(); returns temp from BME280 sensor
```

The simpleBME.c program uses all four functions. Its primary BME 280 calls and output functions are

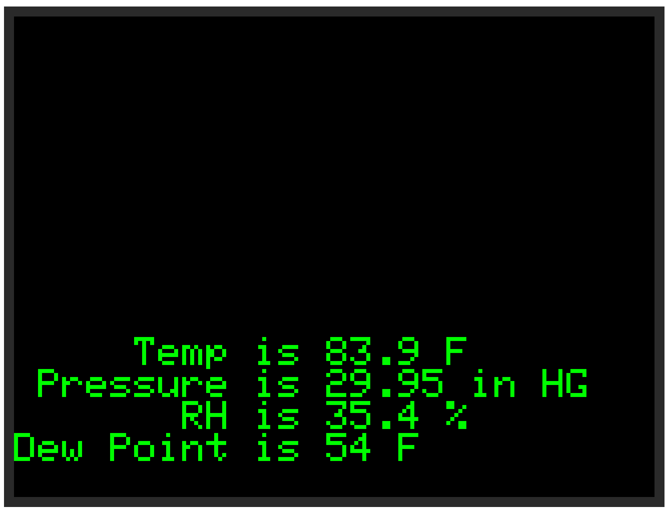
```
float temp=BME_readTemp();
float pressure=BME_readPressure();
float rh=BME_readRH();
float tc=(temp-32)*5/9;
```

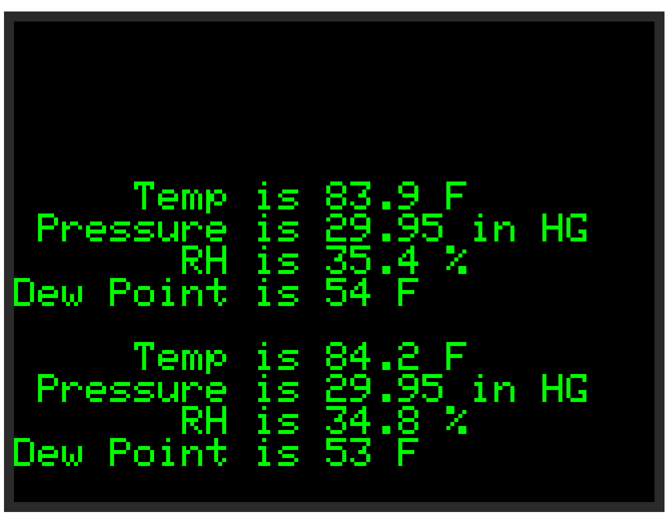
```
  //DP =  243.04*( LN(RH/100)+( 17.625*T  )/ (243.04 + T ))  / (17.625 -  LN(RH / 100) -  (17.625*T ) / (243.04 + T ))
 float dp = (243.04*(log(rh/100)+((17.625*tc) / (243.04 + tc))) / (17.625 - log(rh / 100) - ((17.625*tc) / (243.04 + tc))))*9.0 / 5 + 32;
 printf("\n");
 printf("     Temp is %0.1f F\n",temp);
 printf(" Pressure is %0.2f in HG\n",pressure);
 printf("       RH is %0.1f %%\n",rh);
 printf("Dew Point is %0.0f F\n",dp);
```

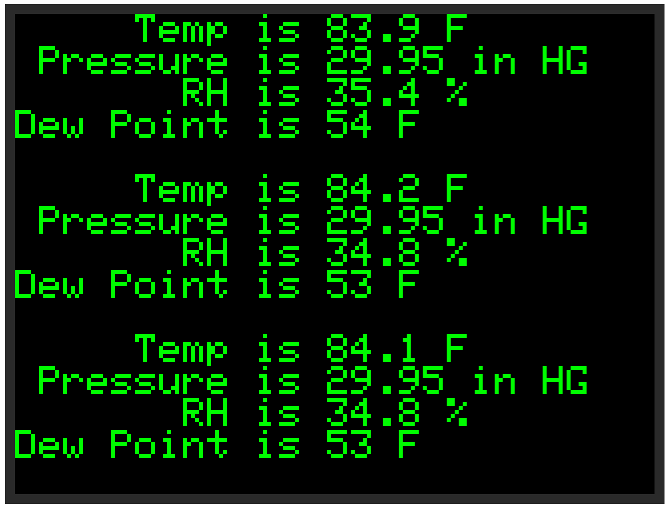
These lines read the temp, pressure, and relative humidity into float values temp, pressure, and rh and calculate the Dew Point using natural log functions as shown in the comment //DP=… Then the program format prints these values with appropriate labels and units. The program's last lines in the loop() function are shown below.

```
delay(5);  // this delays 5 msec and doLoop() calls the ESP32Program loop() which adds another 5 msec of delay
             // for a total of 10 msec delay per loop cycle thus lCount%3000==0 is true every 30 seconds
```

This approach to running loop with a fairly short delay, assures that the ESP32Program gets to process web events frequently (here, typically 100 times per second) and results in a good user response. The screen captures shown below were taken using the Screen Capture button while the program was continuously running on the ESP32.







The /oledDemo.c program on the ESP32 provides a simple example of using a 128x64 pixel OLED I2C attached display with the interpreter. TO use the o* functions, OLED must be defined when compiling the ESP32Program sketch. The program's setup function runs the following lines

```
int i;char buf[33];
oclear();
odrawString(0,0,"Display is 6x21");
for (i=1;i<6;i++) {
  sprintf((char *)&buf,"Display line %d is drawn here",i);
  odrawString(0,10*i,(char *)&buf);
}
odisplay();
delay(5000);
oclear();
odrawString(0,25,"Hello World");
odisplay();delay(3000);oconsoleInit();
for (i=0;i<21;i++)
{
  sprintf((char *)&buf,"Line %d is put on the display here",i);
  oconsolePrintln((char *)&buf);delay(250);
}
```

The help lines for these functions are

```
void oclear(void); clears OLED display, you still have to call odisplay to show
changes
void oconsoleInit(); must be called to start OLED console display
void oconsolePrintln(char *); draw char string at current cursor x,y
void odisplay(void); update display
void odrawLine(int,int,int,int); x1,y1,x2,y2
void odrawRect(int,int,int,int); x,y,width,height
void odrawString(int,int,char *); x,y in pixels,char string
void ofillRect(int,int,int,int); x,y,width,height
void oprint(char *); display in OLED at current cursor position
void oprintln(char *); display in OLED at current cursor position with [CR]
void osetCursor(int,int); x,y
void osetPixel(int,int); x,y
```

When using the OLED, be sure to call odisplay(); each time drawing operations are completed or your display changes will not be shown.

Similar to the OLED display, there is support for a 1.54 inch 200x200 ePaper display that is particularly useful on sensor projects since it provides a low power solution that runs well on batteries and can take advantage of the EPS32's deep sleep modes and still display its information during deep sleep. A fairly complete and useful example ePaper program is /ePaperDemo.c. This program provides an example temperature, relative humidity, barometric pressure, and dew point bar graph display. The ePAPER global must be defined when compiling the ESP32Program to use an EPaper display. The available functions are shown below.

void ePaper_drawLine(int,int,int,int,int) x1,y1,x2,y2,color
void ePaper_drawRect(int,int,int,int,int) x,y,width,height,color -- 0 is black 0xffff is white
void ePaper_fillRect(int,int,int,int,int) x,y,width,height,color
void ePaper_init() initialize ePaper display
void ePaper_powerDown(); put the ePaper display in powerDown mode
void ePaper_println(char *);" print char string
void ePaper_setCursor(int,int); x,y
void ePaper_setTextColor(int); color

Color values are 0 for black and 0xffff for white.