4/19/2021

In-Depth Lab 4

Asynchronous USB Communication

ABSTRACT

The purpose of this lab is to utilize asynchronous communication via the BulldogPIC++ on chip USART to an external serial monitor running on a host computer. The C18 C-language program accepts primitive commands from the host to execute individual modules on startup. The modules include displaying the current Lux value, setting Blink-M smart LED color and intensity, and displaying a hex byte on the PIC18 LEDs. The ability to set both rows on the LCD screen, reading the current temperature in degrees Centigrade, turning on and off a static note frequency or playing part of the Super Mario theme song are also included.

Brandon Empie

ECNS-424

Table of Contents

Problem Discussion

The primary problem in this lab is focused on accounting for the many possible user inputs. Given a primitive command by the host using the Universal Synchronous/Asynchronous Receiver/Transmitter (USART) the programmer needs to direct the command entered to the appropriate outcome desired. After the task has been performed the program must flow back to the start without triggering any other tasks. This issue also presents the situation where an unknown command not accounted for by a task/module must also be dealt with. Below is a diagram showing the primitive commands accepted with their respective desired outcomes.

Command	Outcome
L	The BullDogPIC++ should return the value of Lux
B	The BullDogPIC++ should then accept the color (R,G,B) and intensity (0 -255) for the Blink-M
D	Prompt user for a byte value to be sent to PIC18 LEDs
N	Note on
O	Note off
S	Prompt user for a string to be sent to LCD
T	The BullDogPIC++ returns the current temperature in degrees Centigrade.
default	Display Menu of commands to the user

Due to Asynchronous communication requiring a specific baud rate based on the “Slave” devices specifications the proper baud rate must be selected carefully. For the purpose of this lab the program requires a baud rate of 9600 to function. Failure to complete this step accurately will prevent the program from operating completely but once configured will not require further attention.

A different problem present is each module must be capable of completing from start to finish without interruption. To function, the note frequency generation task relies on an interrupt and must not have a higher priority interrupt occur when running. Likewise, the Blink-M Smart LED task requires Inter-Integrated Circuit (I^2C) serial communication and must be able to complete without disruption. These two tasks in particular must be isolated against disruption to prevent total program failure.

Due to the USART reading in only char characters anytime an integer is required for a module a conversion from ASCII to the integer value must take place. An obstacle here is that because the char data type is a type of integer a simple data conversion or typecast cannot resolve the problem. This issue is essential to the Blink-M intensity portion of its module and the PIC18 LED module.

One of the more complicated issue in this lab relates to allowing the user to send a string to each line of the BulldogPIC++ LCD screen. The user needs to be able to enter each character one at a time in sequence requiring multiple reads from the USART. The LCD has a limit of how many characters it can display which may not seem like a challenge on its own, but the module must at the same time allow the user to end the string early. This means the programmer must somehow incorporate a variable length array or one that scales. This also means the method used to send the array to the LCD screen must also be variable depending on the array size.

A final problem to incorporate a small song time needs to be addressed dynamically. When a musician plays a note there are three factors present. First, the note/frequency being played. Second, the length of time the note is played for. Third, the length of time the musician is not playing any note before he plays his next. These same three factors must be addressed in C programming the same.

The method used to turn the frequency/note on and off is another critical element to this problem. Due to the frequency being generated via capture compare interrupt turning Port C to an input for example will prevent data from outputting to the piezoelectric speaker but will not prevent the interrupt from occurring or Timer 1 from continuing to count thereby wasting resources.

Algorithm Strategy

The primary strategy used in this lab revolved around using switch and case statements any time there was a one-to-many ratio of possible outcomes. This strategy made a difficult task easy as it not only separated the primitive commands into modules natively it also created a default option in the event the user failed to follow instructions. A while loop combined with a switch statement made the perfect backbone of the program giving the user the option to run each module one after the other if they so choose.

To set the baud rate at 9600 the Open1USART() function was used to streamline the process. The function allowed passing multiple parameters into it to easily configure USART1 for asynchronous mode, 8 data bits, no parity, and 1 stop bit. The baud rate was calculated based on the algorithm 9600 baud = 8MHz/(16/(SPBRG+1))[1] where the SPBRG was calculated and set to 51 decimal in this case with the desired outcome being 9600 baud. Another factor considered here was error rate as the rate represents how much data loss will occur. To calculate error rate the formula [(calculated baud rate – desired baud rate)/Desired baud rate] was used. This brings the total error rate to (9615-9600)/9600 = 0.16%[1] which has been deemed acceptable to the programmers' standards.

To accomplish allowing each task to run without interruption the programmer separated them into modules using subroutines beneath each case statement. The note frequency was controlled so that its high priority interrupt does not occur except within its own module to prevent a conflict. This method additionally prevented any overlap between both USART and I^2C communication from occurring.

The Blink-M smart LED module utilized a switch statement with many case statements like the main loop of the program. This method made it easy to set various colors and brightness levels on the Blink-M. The getc1USART() function[2] was used to read characters entered by the user to be matched with the appropriate case statement. The intensity level however needed to be converted from characters to a decimal number between 0-255. This was addressed by subtracting 48 from each character determined by the ASCII table[3].This total was then checked one final time to ensure the number was less than 255 as a method of flow control. If the number is above the maximum the program forces the default case statement and shuts the LED off and lets the user know they failed to follow instructions.

For general communication between the program and the USART the putrs1USART() function was very effective. This function provided some control as to what line the cursor could be placed prior to sending the next prompt. The function fprintf()[4] was used to send data to the user as it allowed passing data stored in a variable to the USART effectively.

To send a string of characters to the BuldogPIC++ LCD screen the USART requests one character at a time using the getc1USART() function and for each to be stored into an array followed by checking if the USART was ready for the next character with the DataRdy1USART() function. This process was placed in a loop to handle its repetitiveness. The main issue with this method was that in order to exit the loop two conditions needed to be possible for the array to be considered variable as desired. The loop runs until a carriage return/enter key has been pressed by the user or if the string entered by the user reaches the LCD maximum length of 16 characters. A check variable “done” was used within the loop to assess each character the user enters to “check” for the carriage return. To check if the LCD string length had been reached the array pointer “S” was used as it indicated the size of the array after one complete loop.

Finally, to play part of the Super Mario theme song the note frequency, note length, and delay period before the next note is played were all stored into arrays. This method allowed the use of a single array index variable to be used to set each segment of the song. The use of arrays was critical for this to function as they allowed the frequency, note length, and delay period to constantly change as they do in music as mentioned in the problem discussion section.

Explanation of Flowchart

Initialization and Main:

- Set internal clock to 8MHz
- Set light sensor channel to analog configuration
- Setup A/D
- Initialize I^2C
    - I^2C Clock output use MSSP module
    - I^2C data input/output use MSSP module
    - Clock = Fosc/4
    - Input data sampled at end of data output time
    - Transmit occurs on transition from active to Idle clock state
    - I^2C baud rate set to 50KHz
- Initialize USART
    - Asynchronous mode
    - Eight bit
    - 9600 baud rate

- Shut off Blink-M
- Initialize capture compare 2 & Timer 1
- Timer 1 off
- Start Main loop
- Request primitive command from user on USART
- User entered ‘L’
    - Read A/D
    - Convert A/D result into Lux
    - Display Lux value to USART for the user
    - Restart main loop
- User entered ‘B’
    - Call “Color” subroutine
    - Restart main loop
- User entered ‘D’
    - Call “LEDs” subroutine
    - Restart Main loop
- User entered ‘N’
    - Give user option to play static note or part of the Super Mario theme song
    - If user entered ‘N’ again set the frequency to the note E Natural and turn on Timer 1 (enables Compare ISR) to generate frequency on buzzer, let user know note is on via USART then restart main loop
    - If user entered “S” Turn on Timer 1 (enables compare ISR) to generate frequency on buzzer and play part of Super Mario theme song then restart main loop

- User entered ‘O’
    - Turn off timer 1 (disabling the ISR)
    - Let user know via USART that the note is now off
    - Restart Main loop
- User entered ‘S’
    - Tell user to enter a string for line 1 of LCD via USART
    - Clear LCD from before
    - Set cursor to beginning of Line 1
    - Record and send users string to Line 1 of the LCD by calling SendString()
    - Ask user if they want to enter a string for line 2
    - If they enter a lower or uppercase y, set cursor to beginning of line 2, record and send users string to line 2 of the LCD by calling SendString()
    - If they entered anything else besides an upper or lower case y, restart main loop
- User entered ‘T’
    - Read the current temperature in degrees centigrade by calling tempread()
    - Print temperature to the USART for the user to see it
    - Restart Main loop
- User failed to enter a valid command
    - Send command menu list to USART for user to see
    - Restart Main loop
- Display waiting for command prompt on USART to let user know the program is ready for another command
- **Interrupt Service Routine Compare:**

- Compare ISR used to generate note frequency output to piezoelectric speaker
- Check to make sure CCP2 interrupt flag caused the interrupt
- Clear timer 1 to prevent "lau" frequency from occurring
- Clear CCP2 interrupt flag for the next time

Subroutine-ADConveter:

- Reads 8 bit A/D result from Everlight light sensor
- Stores result into variable X
- Return to Main

Subroutine-BlinkM:

- Each step requires the master send data and receive acknowledgements from the slave to indicate successful communication and therefore takes time
- Start I^2C
- Once started, send Blink-M address 0x00
- Send Blink-M command to stop script
- Send Blink-M command to get ready for RGB values
- Send Red, Green, then Blue values to Blink-M in sequence
- Stop I^2C
- Return to Main

Subroutine-TempRead:

- Purpose of this subroutine is to store current temperature read by TC-74 temperature sensor into the variable called "temperature"

- Start I^2C
- Send TC-74 address 0x9A to write
- Read Configuration from TC-74
- Send TC-74 address 0x9B to read
- Read TC-74
- If not ready to read temperature update ‘cmd’ and restart subroutine
- Otherwise read temperature into “temperature” variable
- Return to Main

Subroutine-Color:

- Let user pick a color for Blink-M via USART (red, green, blue, yellow, cyan, purple)
- Let user select Intensity level from 0 to 255 via USART
- Convert 3-digit characters entered by user into one integer
- Check to make sure user followed directions, shut off Blink-M if they did not
- Setup R,G,B variables based on desired color
- Call BlinkM() subroutine to activate Blink-M with above parameters using I^2C
- Return to Main

Subroutine-LEDs:

- Request two capitalized characters from user via USART
- Convert characters into integer value
- Left shift first character 4 bits to the left to get hexadecimal upper byte
- Add upper byte and lower byte together to form complete byte
- Send byte to PIC18 LEDs

- Print value of HexUpper variable to the USART
- Restart Main loop

Subroutine-SendString:

- Read in one character at a time from USART into array LCDString[S]
- Copy read character into “done” variable
- Increment S for next loop
- Repeat above steps until user has pressed enter key (carriage return) or they have entered 16 characters
- Store the size of the array minus one into “Arraysize” if carriage return was used
- Clear array index variable for next time
- Write string to the LCD display
- Return to Main

Subroutine-SuperMario:

- Set note frequency via CCPR2 with NoteArray[j]
- Clear Timer 1
- Turn the frequency/sound on via Timer 1
- Leave the note on by delaying based on NoteLength[j]
- Turn the frequency/sound off via Timer 1
- Delay based on delay[j] before next note should be played
- Increment array index for next sequence
- The above process is then repeated until the array index reaches 24
- The same sequence of notes is then played again starting at index = 8

- A new sequence of notes is then played with the same method used above, just new arrays
- Return to main

Unique Problem Resolution

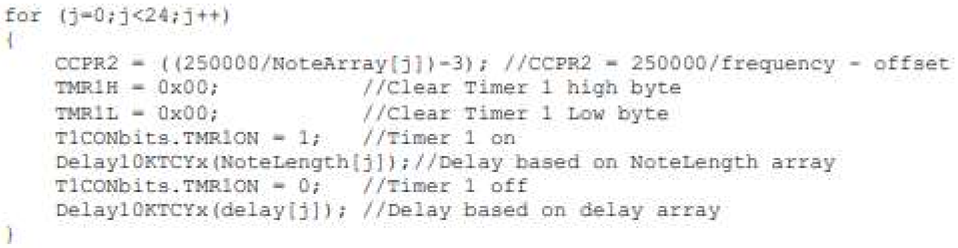
The first issue encountered was figuring out how to send data back to the USART. This step was required for reading the lux and temperature values and was critical to the main part of the program.The putrs1USART() function was effective at sending preset strings but did not have the capability to send data stored in variables to the USART. After experimenting by setting break points in MPLAB X IDE the solution was found to be the fprintf() function. Fprintf() allows sending both strings and variables using the variable data type format specifier.

The next unique issue encountered revolved around converting multiple characters input on the USART into an integer value for the Blink-M Intensity. The C18 USART function libraries were explored for a method to read in a char as an integer. The programmer realized one did not exist because a char is in fact an integer data type. The issue present was deduced from this point that the char was being read in as its ASCII value. To circumvent this each character was converted from ASCII to its integer digit equivalent. After all digits had been read in and converted, they needed to be reassembled into one combined integer number. This was resolved using the equation [Intensity = (H*100 + T*10 + O)] where the variables represent the digit order hundreds, tens, and ones place as they were entered initially. The result was then sent to the Blink-M according to the color selected prior by the user in accordance with the Blink-M data sheet[5] to set the "Intensity" of the color.

The third unique issue revolved around allowing the user to send a string to the LCD display. It was clear that the user needed to enter one character at a time meanwhile the program can store it into an array followed by the array index being incremented for the next character until finished. The need for a loop was evident but how to terminate this loop took some thought. After investigation it was resolved that the two conditions needing to be incorporated was first in the event the array reaches its maximum or the maximum character length of the LCD.

The second condition consisted of how the loop is terminated in the event the user is done typing but has not maxed out the array length. The first condition was simpler as the index variable for the array can be compared at the end of the loop against the maximum 16. The second condition was desired to include a carriage return or in the event the user pressed the enter key the loop exit. The first thought what happens when the enter key is pressed? This question needed to be analyzed by single stepping the program at this point to see what value is stored into the array when the key is pressed. The result was 0xd in hex and at this point both conditions were known but now a solution on how to incorporate both together required investigation. The programmer took a fundamental approach and set the two conditions in a do while statement using an AND logic gate[6] like so [do{}while(done != 0xd && S != 16);]. The done variable was used to check for the carriage return allowing the index variable to increment at the end of the loop. The loop would then proceed until one of the conditions became false (0) at which point it will exit.

Lastly, being able to play a song required a process to address the frequency of the note, the length the note is played, and the delay period while no note is played prior to the next frequency to be played. This concept is a repeating occurrence in any song but the values change very often for each creating this complex problem. The programmer realized the easiest way to resolve this issue was to create 3 large arrays to handle each situation individually. These three arrays preconfigured with the appropriate values could then be incremented using an index variable at the end of a for loop as seen below.



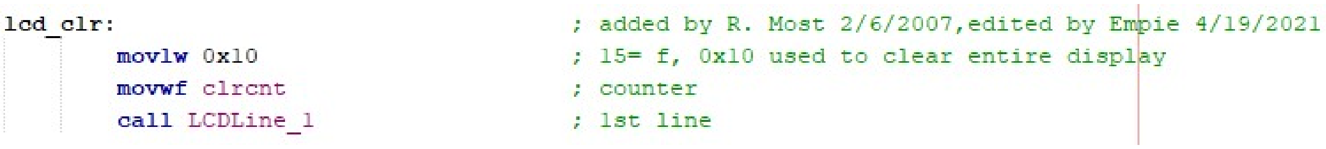
```
for (j=0;j<24;j++)
{
    CCPR2 = ((250000/NoteArray[j])-3); //CCPR2 = 250000/frequency - offset
    TMR1H = 0x00;           //Clear Timer 1 high byte
    TMR1L = 0x00;           //Clear Timer 1 Low byte
    T1CONbits.TMR1ON = 1;   //Timer 1 on
    Delay10KTCYx(NoteLength[j]);//Delay based on NoteLength array
    T1CONbits.TMR1ON = 0;   //Timer 1 off
    Delay10KTCYx(delay[j]); //Delay based on delay array
}
```

Difficulties and Workarounds

One difficulty present for the LEDs subroutine involved figuring out a way to convert two characters into their hexadecimal equivalent. To handle numbers between from zero through 9 the subroutine handles them in the same fashion as does other areas of the program. Because the range 00-FF incorporates letters it was a little tricky because the letters had to be handled separately from the ASCII numbers. This was handled by verifying the ASCII values against the entered character. If a letter was entered the program subtracts 55 decimal to produce a result between A-F (10-15) at which point if the value is the first character entered it is left shifted 4 bits to become the upper nibble of the result. The process is repeated for the second character except for the bit shift as it will already be in lower byte format.

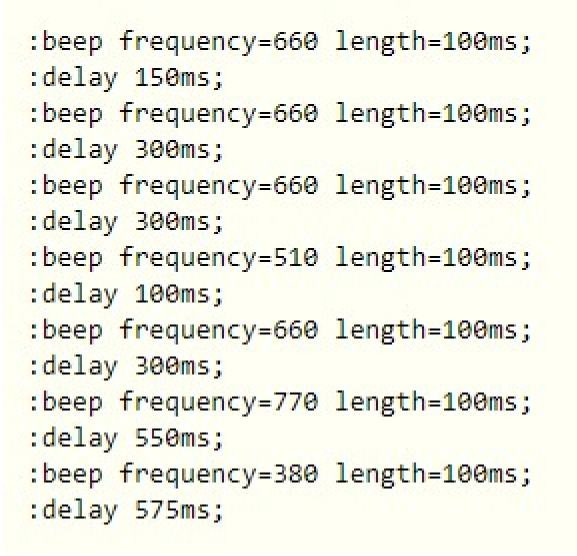
The next difficulty revolved around the LCD screen particularly to setting the second line of the screen. Initially, the SendString() subroutine would only write to the first line and trying to determine what line it should write to depending on user input was becoming cumbersome as many new variables were having to be introduced. The programmer came up with a genius workaround to the issue by using the LCD function calls LCDLine_1() and LCDLine_2() outside of the subroutine to make it more modular. This worked well considering that the starting position for writing to each line was at the first position of each row concurrent with the LCD functions. The SendString() subroutine was then setup to write to the LCD screen to the first line of the LCD and second if they so choose.

Another small workaround with the LCD screen was that the programmer desired an easy way to clear the screen each time the user choose to write a new string to it. The lcd_clr() being the most effective method was discovered to have one caveat. It was discovered through testing that the function could only clear 15 characters on the LCD screen instead of the required 16. This was resolved by modifying the assembly language program under the function and changing the value stored into the clrcnt counter from 0xf to 0x10 as shown below.



```
lcd_clr:                                ; added by R. Most 2/6/2007,edited by Empie 4/19/2021
        movlw 0x10                      ; 15= f, 0x10 used to clear entire display
        movwf clrcnt                    ; counter
        call LCDLine_1                  ; 1st line
```

The last difficulty pertained to the Super Mario theme song. Initially the subroutine would not run because of a random error. After the error code was investigated it was discovered that the memory space reserved for variables was full. This was a new and unexpected issue for the programmer to encounter. The programmer handled this problem by storing three of the arrays into ROM to circumvent the issue. From this point forward the programmer was more aware and started reusing variables as much as possible. The array values used were found on a wiki page[7] and converted appropriately with a calculator, an example of the page information is show in the screen capture below.



```
:beep frequency=660 length=100ms;
:delay 150ms;
:beep frequency=660 length=100ms;
:delay 300ms;
:beep frequency=660 length=100ms;
:delay 300ms;
:beep frequency=510 length=100ms;
:delay 100ms;
:beep frequency=660 length=100ms;
:delay 300ms;
:beep frequency=770 length=100ms;
:delay 550ms;
:beep frequency=380 length=100ms;
:delay 575ms;
```

[7]

After getting the subroutine to run it was noticed that random notes in the sequence would sometimes not play. The programmer realized some part of the interrupt was causing the note frequency to only last a moment causing no frequency to output to the piezoelectric speaker. Even though it was logically still being played as all the other notes were in the loop. After analyzing the problem it became clear that Timer 1 was still counting regardless if the frequency was output to the speaker or not. This meant that certain frequencies would by default be cut off in the event Timer 1 was near the CCP2 value giving the random note loss effect.

The programmer was forced to find an alternative method to turn off the speaker during the "off" period prior to the note being played. The first consideration made to flip the speaker to an input was ineffective in that it did not fix the problem and that the interrupt would still occur during the off period. The result was discovered by setting Timer 1 to 0x0000 just before the next note was to be played. This corrected the issue while also preventing the processor from wasting resources as the interrupt no longer would occur if Timer 1 never reached the compare value.

Results

The program performed all primitive commands exceptionally well. Through the USART the user can reasonably communicate to the program to request and provide two-way communication. It accurately displays the lux, temperature in degrees centigrade, and user entered hex byte on the PIC18 LEDs when given the primitive command input ‘L’,’T’, and ‘D’. When the user selects primitive command ‘R’ followed by ‘128’ the Blink-M displays the color red at approximately 50% intensity as expected. Given the commands ‘N’ ‘N’ the note E natural is played on the piezoelectric speaker until deactivated by entering the ‘O’ command as designed. Through the ‘S’ command the string “Its Super Mario” was sent to line 1 of the LCD through the USART verifying its functionality. These modules were all verified by entering the commands and monitoring output of the PUTTY terminal emulator application, single stepping the MPLAB X IDE, and by observation of the naked eye.

Conclusion

The program complexity was intense requiring the programmer fully understand all previous labs in the course quite well. A background situation present throughout the lab was the general length of code as the programmer did not have a history of efficiency. He was forced to learn some new methods to make a lengthy section of code into a much more condensed version in effort to prevent a 20 hour flow chart drawing at the end. The switch statement proved invaluable in this lab and a critical aspect regarding how each module would run. Without which it would have been much more difficult and a lengthy process to perform the same functionality with if statements. Flow control was put into effect slightly in this lab as the programmer did not favor the idea that the user could get away with not following directions without it which was a new concept for consideration. Modifications to the original program were made for general functionality as it was nice to be able to use both LCD lines as well as send some additional colors to the Blink-M. The Super Mario subroutine while most frustrating incorporated the most fun. The power of C programming was really put on display in that section as the entire song plays in less than 50 lines of code.

Some final considerations for the program revolve around flow control, verification for the user, and doing fun things. The program has room for more flow control mechanisms such as allowing the user to enter both upper-case and lower-case commands throughout the entire program would be a nice feature. Verification for the user throughout the program would be beneficial. For example, if the user entered the string “Super Mario” it would be nice for the string to be fed back to the USART so the user could see what is being sent. Another alternative here would be that ability for the string to appear on the screen prior to the carriage return being entered as the user enters it like how a text message application functions on a cell phone. Some fun things the user thought about given more time would be to integrate the Blink-M with the piezoelectric speaker to create a “light show” whereas the notes play, and the color/intensity change simultaneously. For this to work the programmer would start by exploring the Blink-M script function as I^2C cannot be interrupted by the high priority compare interrupt used for the frequency and vice versa with the note frequency.

References

[1]PIC18F46K80 Microchip product data sheet (DS30009977G), "PIC18F46K80 Family 28/40/44/64-Pin, Enhanced Flash Microcontrollers with ECAN™ XLP Technology"https://ww1.microchip.com/downloads/en/DeviceDoc/PIC18F66K80%20FAMILY%20Enhanced%20Flash%20MCU%20with%20ECAN%20XLP%20Technology%2030009977G.pdf., 2017.

[2]MPLAB ® C18 C COMPILER LIBRARIES (DS51297F) http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB_C18_Libraries_51297f.pdf., 2005.

[3]SPLC782A Sunplus product data sheet (Version 1.0), "SPLC782A 16COM/80SEG Controller/Driver" https://www.pacificdisplay.com/ics_app%20notes/sunplus/SPLC782av10.pdf,2001.

[4]"Format Specification Syntax: printf` and wprintf` Functions," https://docs.microsoft.com/en-us/cpp/c-runtime-library/format-specification-syntax-printf-and-wprintf-functions?view=msvc-160 Oct. 26, 2020.

[5]"Blinkm data sheet (v20080130a), "BLINKM v1 DATASHEET", https://oldsite.thingm.com/fileadmin/thingm/downloads/old/BlinkM_datasheet-20080603.pdf., 2007

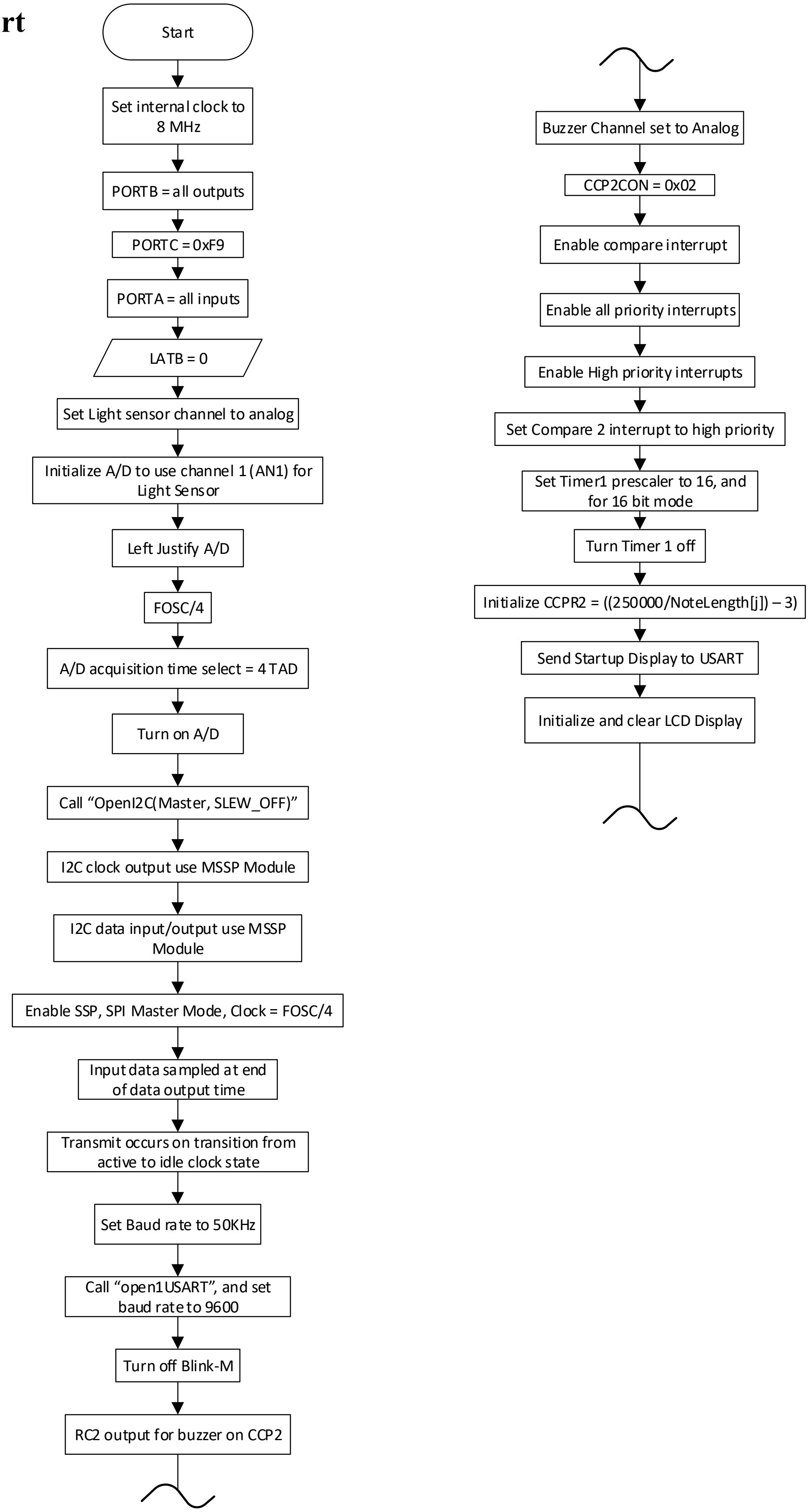
[6]"While loop with multiple conditions in C++," *Stack Overflow*, https://stackoverflow.com/questions/16568149/while-loop-with-multiple-conditions-in-c May 15, 2013.

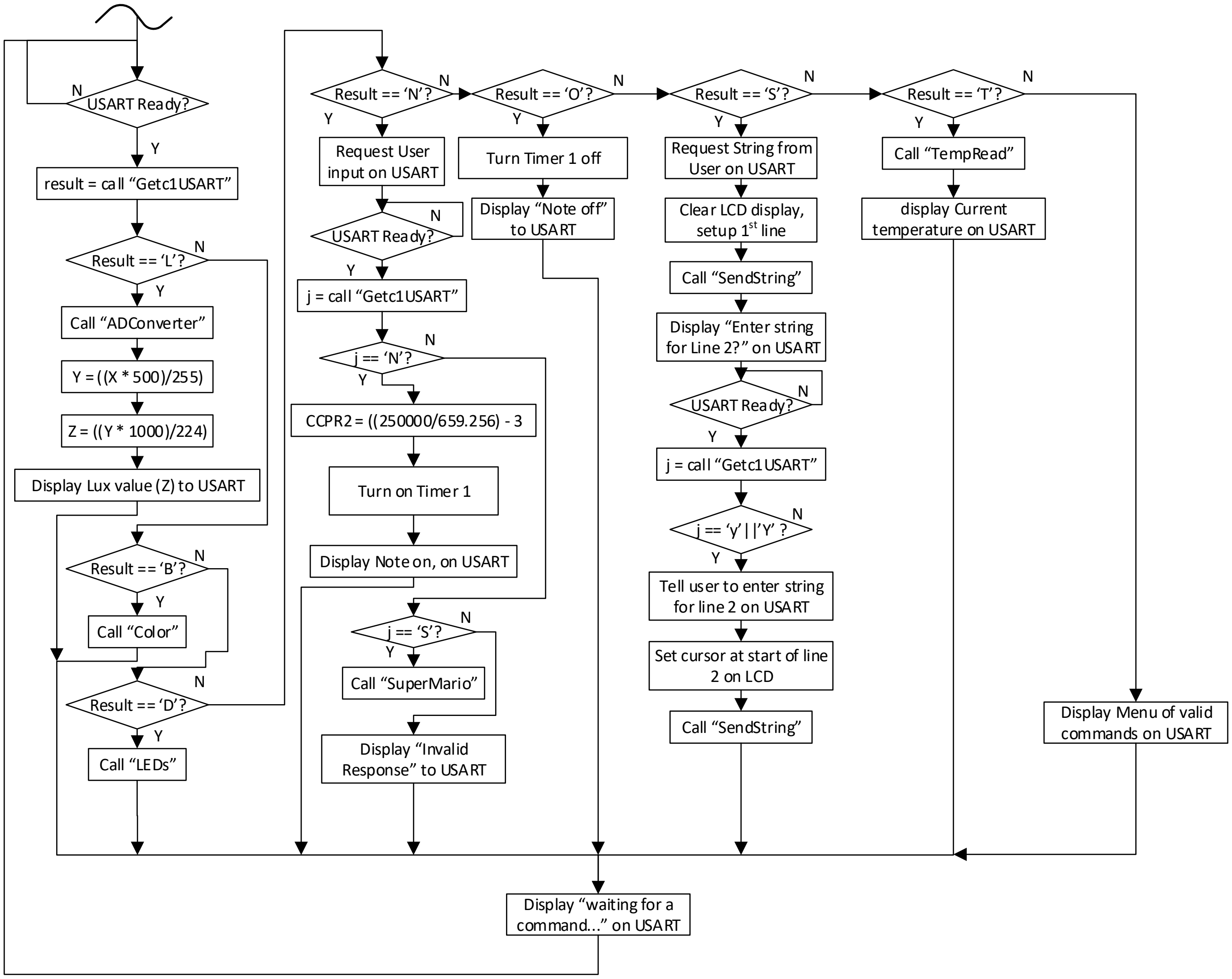
[7]"Super Mario Theme - MikroTik Wiki," https://wiki.mikrotik.com/wiki/Super_Mario_Theme, 2021

Appendix

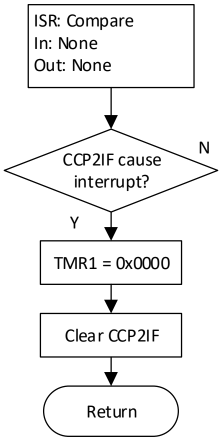
Empie INDL4 Flowchart

Flowchart

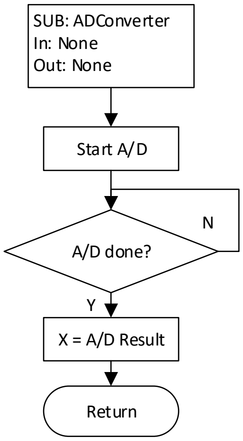

Start
Set internal clock to 8 MHz
PORTB = all outputs
PORTC = 0xF9
PORTA = all inputs
LATB = 0
Set Light sensor channel to analog
Initialize A/D to use channel 1 (AN1) for Light Sensor
Left Justify A/D
FOSC/4
A/D acquisition time select = 4 TAD
Turn on A/D
Call "OpenI2C(Master, SLEW_OFF)"
I2C clock output use MSSP Module
I2C data input/output use MSSP Module
Enable SSP, SPI Master Mode, Clock = FOSC/4
Input data sampled at end of data output time
Transmit occurs on transition from active to idle clock state
Set Baud rate to 50KHz
Call "open1USART", and set baud rate to 9600
Turn off Blink-M
RC2 output for buzzer on CCP2
Buzzer Channel set to Analog
CCP2CON = 0x02
Enable compare interrupt
Enable all priority interrupts
Enable High priority interrupts
Set Compare 2 interrupt to high priority
Set Timer1 prescaler to 16, and for 16 bit mode
Turn Timer 1 off
Initialize CCPR2 = ((250000/NoteLength[j]) – 3)
Send Startup Display to USART
Initialize and clear LCD Display



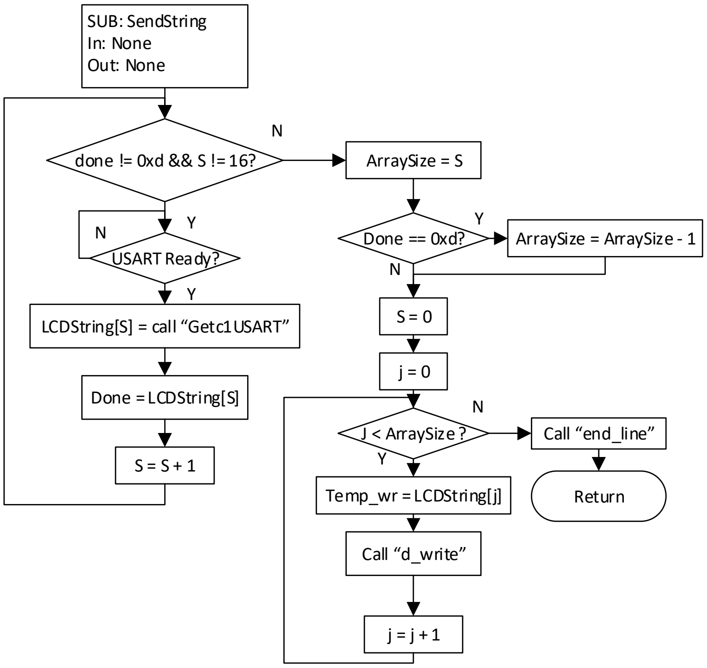
USART Ready?
N
Y
result = call "Getc1USART"
Result == 'L'?
N
Y
Call "ADConverter"
Y = ((X * 500)/255)
Z = ((Y * 1000)/224)
Display Lux value (Z) to USART
Result == 'B'?
N
Y
Call "Color"
Result == 'D'?
N
Y
Call "LEDs"
Result == 'N'?
N
Y
Request User input on USART
USART Ready?
N
Y
j = call "Getc1USART"
j == 'N'?
N
Y
CCPR2 = ((250000/659.256) - 3
Turn on Timer 1
Display Note on, on USART
j == 'S'?
N
Y
Call "SuperMario"
Display "Invalid Response" to USART
Result == 'O'?
N
Y
Turn Timer 1 off
Display "Note off" to USART
Result == 'S'?
N
Y
Request String from User on USART
Clear LCD display, setup 1st line
Call "SendString"
Display "Enter string for Line 2?" on USART
USART Ready?
N
Y
j = call "Getc1USART"
j == 'y' | |'Y' ?
N
Y
Tell user to enter string for line 2 on USART
Set cursor at start of line 2 on LCD
Call "SendString"
Result == 'T'?
N
Y
Call "TempRead"
display Current temperature on USART
Display Menu of valid commands on USART
Display "waiting for a command…" on USART



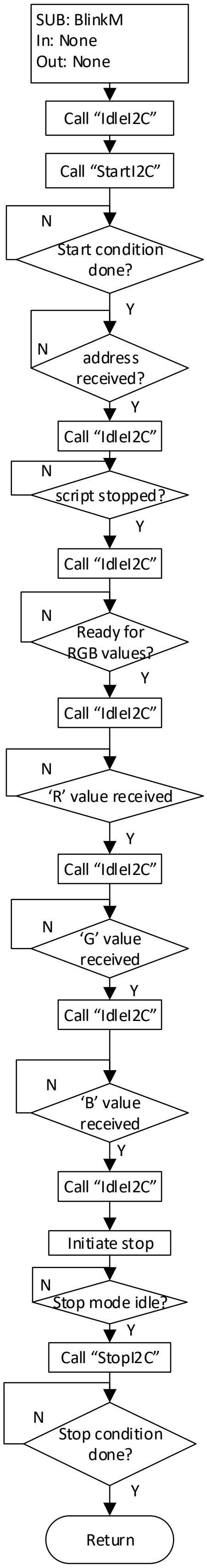
ISR: Compare
In: None
Out: None
CCP2IF cause interrupt?
N
Y
TMR1 = 0x0000
Clear CCP2IF
Return

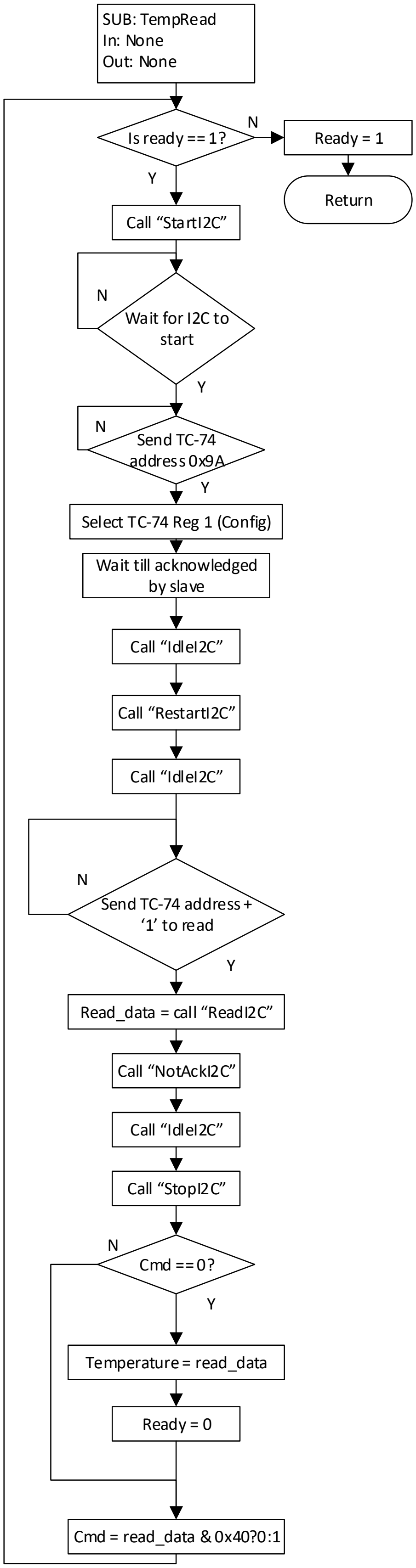


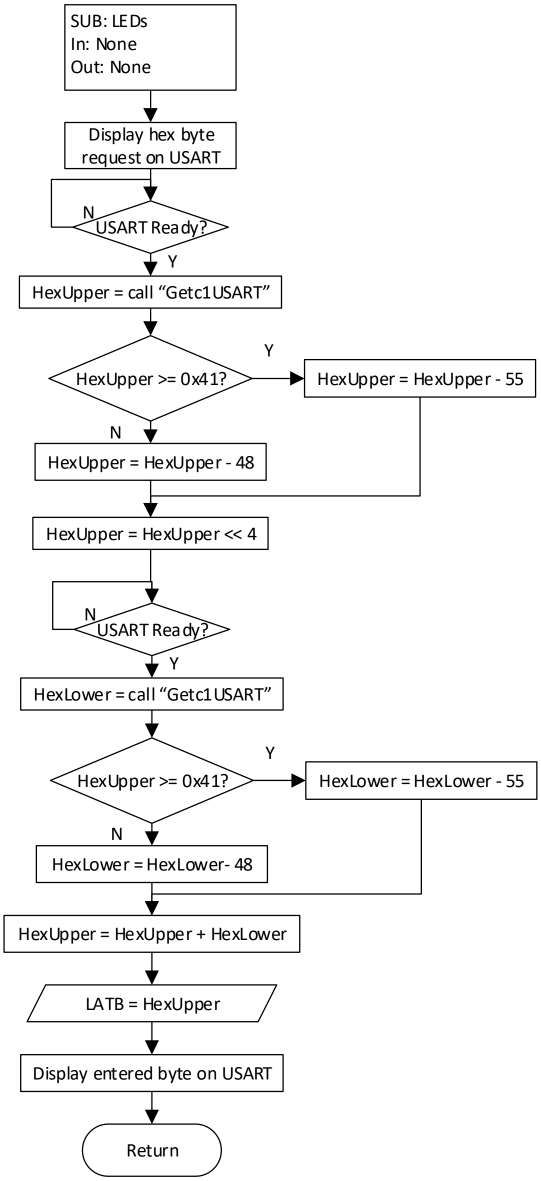
SUB: ADConverter
In: None
Out: None
Start A/D
A/D done?
N
Y
X = A/D Result
Return

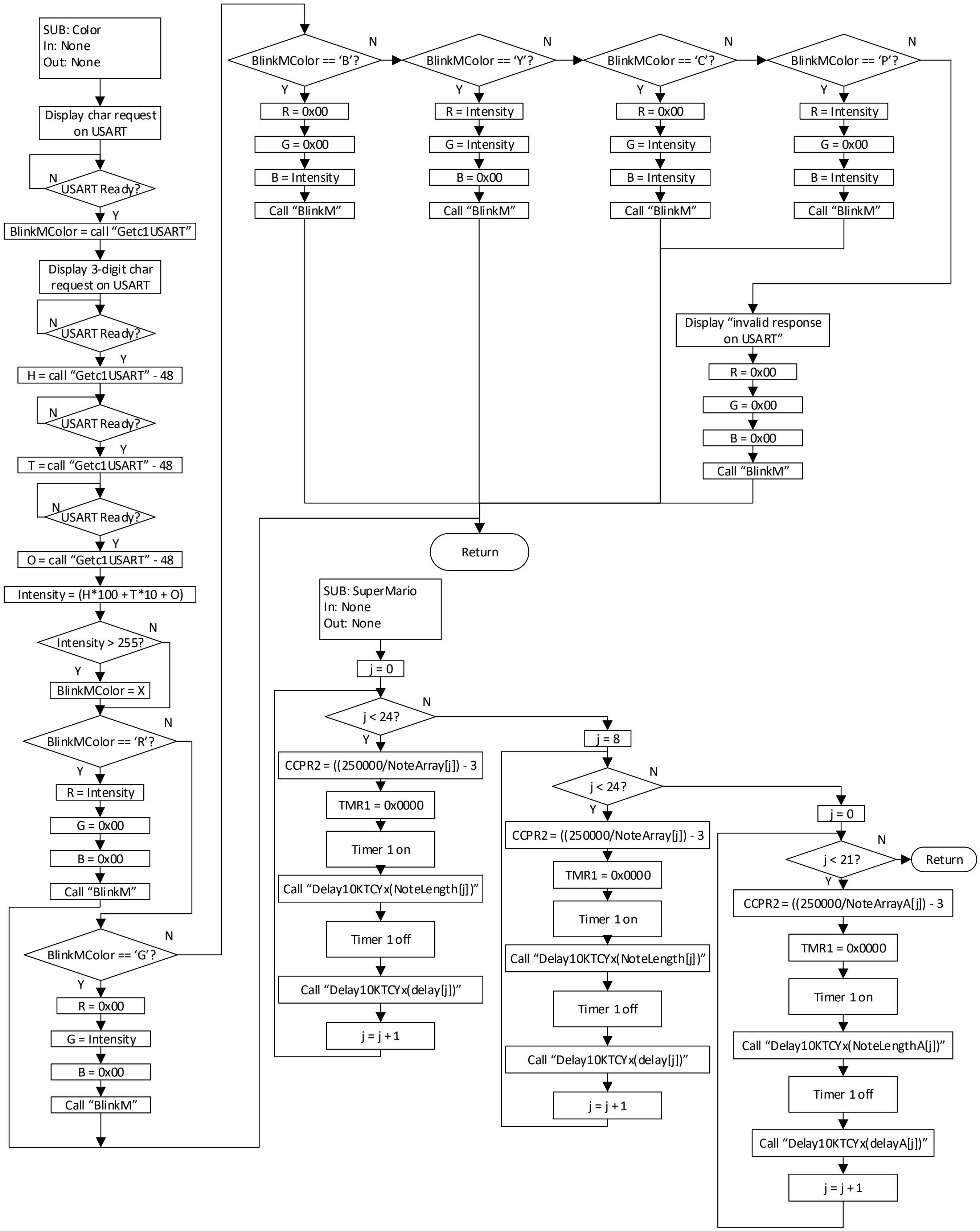


SUB: SendString
In: None
Out: None
done != 0xd && S != 16?
N
Y
USART Ready?
N
Y
LCDString[S] = call "Getc1USART"
Done = LCDString[S]
S = S + 1
ArraySize = S
Done == 0xd?
Y
N
ArraySize = ArraySize - 1
S = 0
j = 0
J < ArraySize ?
N
Y
Temp_wr = LCDString[j]
Call "d_write"
j = j + 1
Call "end_line"
Return

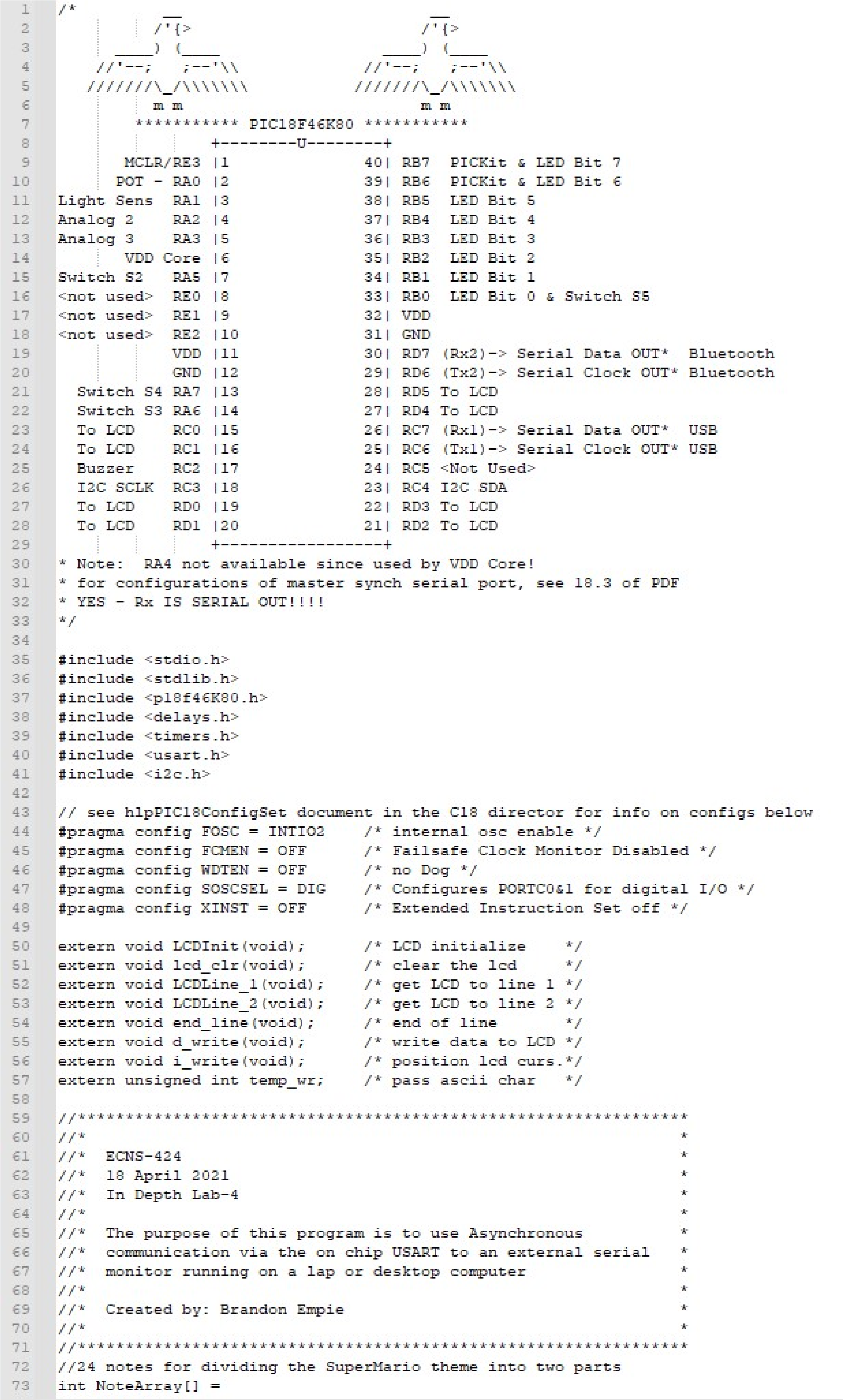

SUB: BlinkM
In: None
Out: None
Call "IdleI2C"
Call "StartI2C"
N
Start condition done?
Y
N
address received?
Y
Call "IdleI2C"
N
script stopped?
Y
Call "IdleI2C"
N
Ready for RGB values?
Y
Call "IdleI2C"
N
'R' value received
Y
Call "IdleI2C"
N
'G' value received
Y
Call "IdleI2C"
N
'B' value received
Y
Call "IdleI2C"
Initiate stop
N
Stop mode idle?
Y
Call "StopI2C"
N
Stop condition done?
Y
Return


SUB: TempRead
In: None
Out: None
Is ready == 1?
N
Ready = 1
Return
Y
Call "StartI2C"
N
Wait for I2C to start
Y
N
Send TC-74 address 0x9A
Y
Select TC-74 Reg 1 (Config)
Wait till acknowledged by slave
Call "IdleI2C"
Call "RestartI2C"
Call "IdleI2C"
N
Send TC-74 address + '1' to read
Y
Read_data = call "ReadI2C"
Call "NotAckI2C"
Call "IdleI2C"
Call "StopI2C"
N
Cmd == 0?
Y
Temperature = read_data
Ready = 0
Cmd = read_data & 0x40?0:1

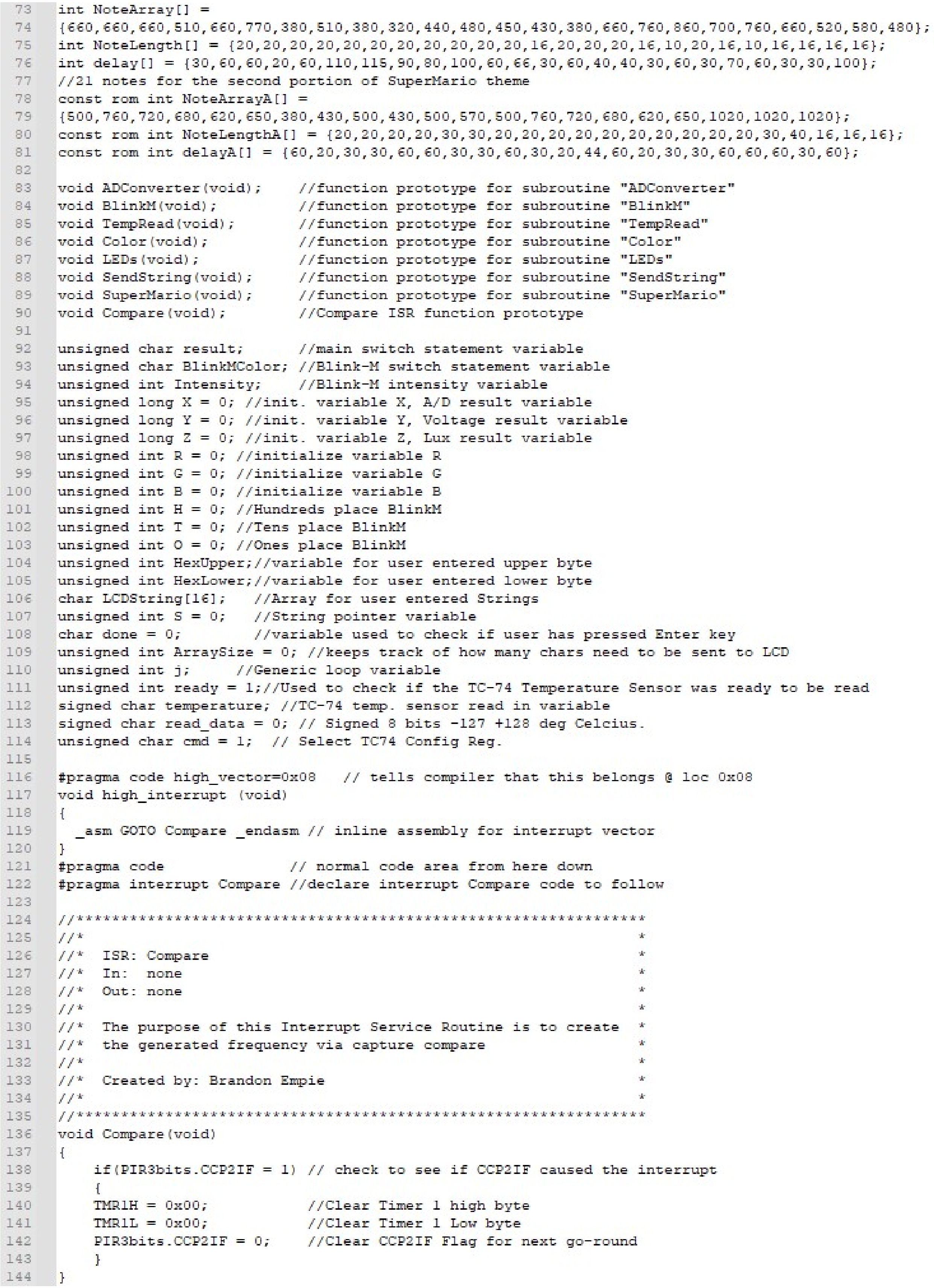

SUB: LEDs
In: None
Out: None
Display hex byte request on USART
N
USART Ready?
Y
HexUpper = call "Getc1USART"
HexUpper >= 0x41?
Y
HexUpper = HexUpper - 55
N
HexUpper = HexUpper - 48
HexUpper = HexUpper << 4
N
USART Ready?
Y
HexLower = call "Getc1USART"
HexUpper >= 0x41?
Y
HexLower = HexLower - 55
N
HexLower = HexLower- 48
HexUpper = HexUpper + HexLower
LATB = HexUpper
Display entered byte on USART
Return


SUB: Color
In: None
Out: None
Display char request on USART
USART Ready?
N
Y
BlinkMColor = call “Getc1USART”
Display 3-digit char request on USART
USART Ready?
N
Y
H = call “Getc1USART” - 48
USART Ready?
N
Y
T = call “Getc1USART” - 48
USART Ready?
N
Y
O = call “Getc1USART” - 48
Intensity = (H*100 + T*10 + O)
Intensity > 255?
N
Y
BlinkMColor = X
BlinkMColor == ‘R’?
N
Y
R = Intensity
G = 0x00
B = 0x00
Call “BlinkM”
BlinkMColor == ‘G’?
N
Y
R = 0x00
G = Intensity
B = 0x00
Call “BlinkM”
BlinkMColor == ‘B’?
N
Y
R = 0x00
G = 0x00
B = Intensity
Call “BlinkM”
BlinkMColor == ‘Y’?
N
Y
R = Intensity
G = Intensity
B = 0x00
Call “BlinkM”
BlinkMColor == ‘C’?
N
Y
R = 0x00
G = Intensity
B = Intensity
Call “BlinkM”
BlinkMColor == ‘P’?
N
Y
R = Intensity
G = 0x00
B = Intensity
Call “BlinkM”
Display “invalid response on USART”
R = 0x00
G = 0x00
B = 0x00
Call “BlinkM”
Return
SUB: SuperMario
In: None
Out: None
j = 0
j < 24?
N
Y
CCPR2 = ((250000/NoteArray[j]) - 3
TMR1 = 0x0000
Timer 1 on
Call “Delay10KTCYx(NoteLength[j])”
Timer 1 off
Call “Delay10KTCYx(delay[j])”
j = j + 1
j = 8
j < 24?
N
Y
CCPR2 = ((250000/NoteArray[j]) - 3
TMR1 = 0x0000
Timer 1 on
Call “Delay10KTCYx(NoteLength[j])”
Timer 1 off
Call “Delay10KTCYx(delay[j])”
j = j + 1
j = 0
j < 21?
N
Return
Y
CCPR2 = ((250000/NoteArrayA[j]) - 3
TMR1 = 0x0000
Timer 1 on
Call “Delay10KTCYx(NoteLengthA[j])”
Timer 1 off
Call “Delay10KTCYx(delayA[j])”
j = j + 1

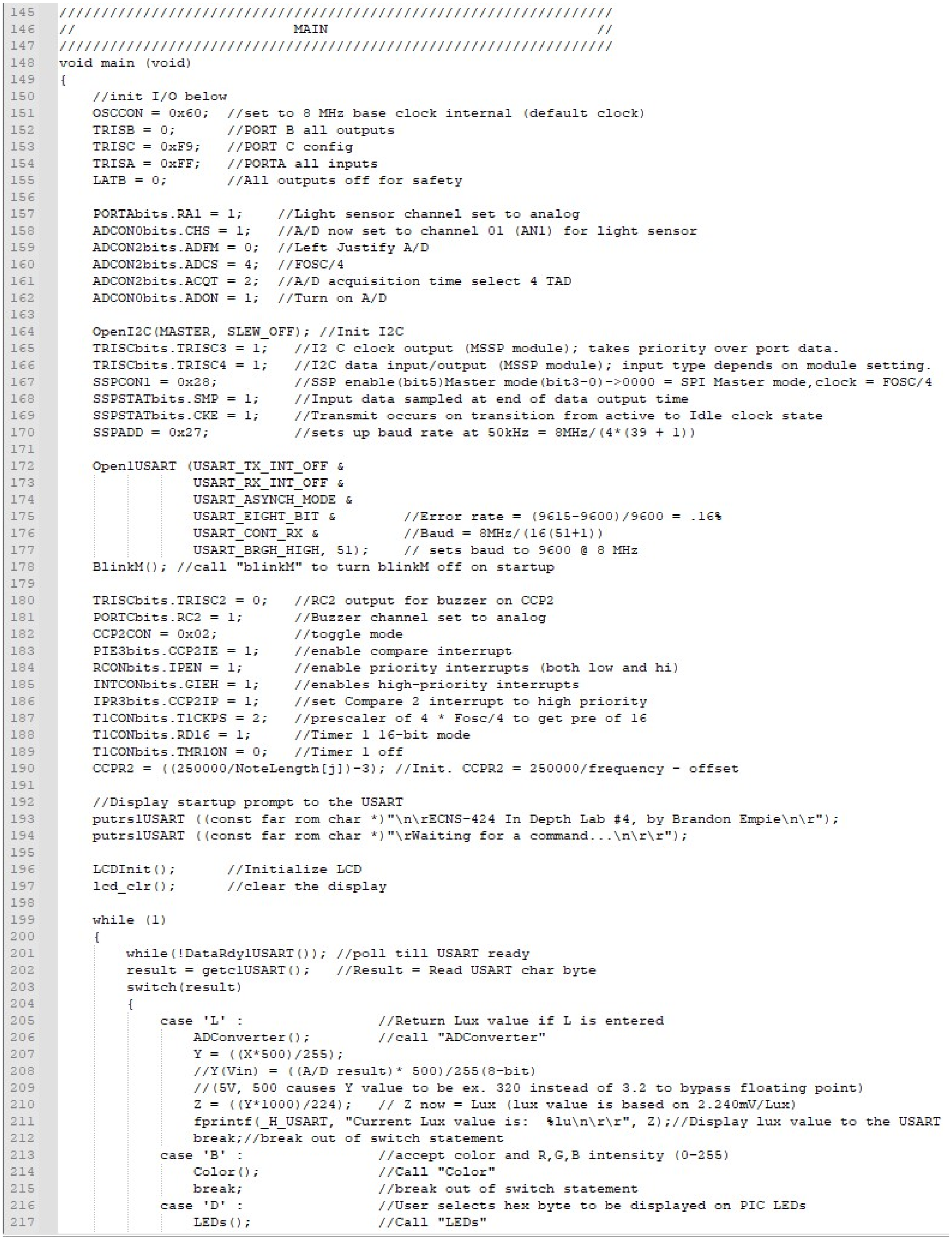
C-Code



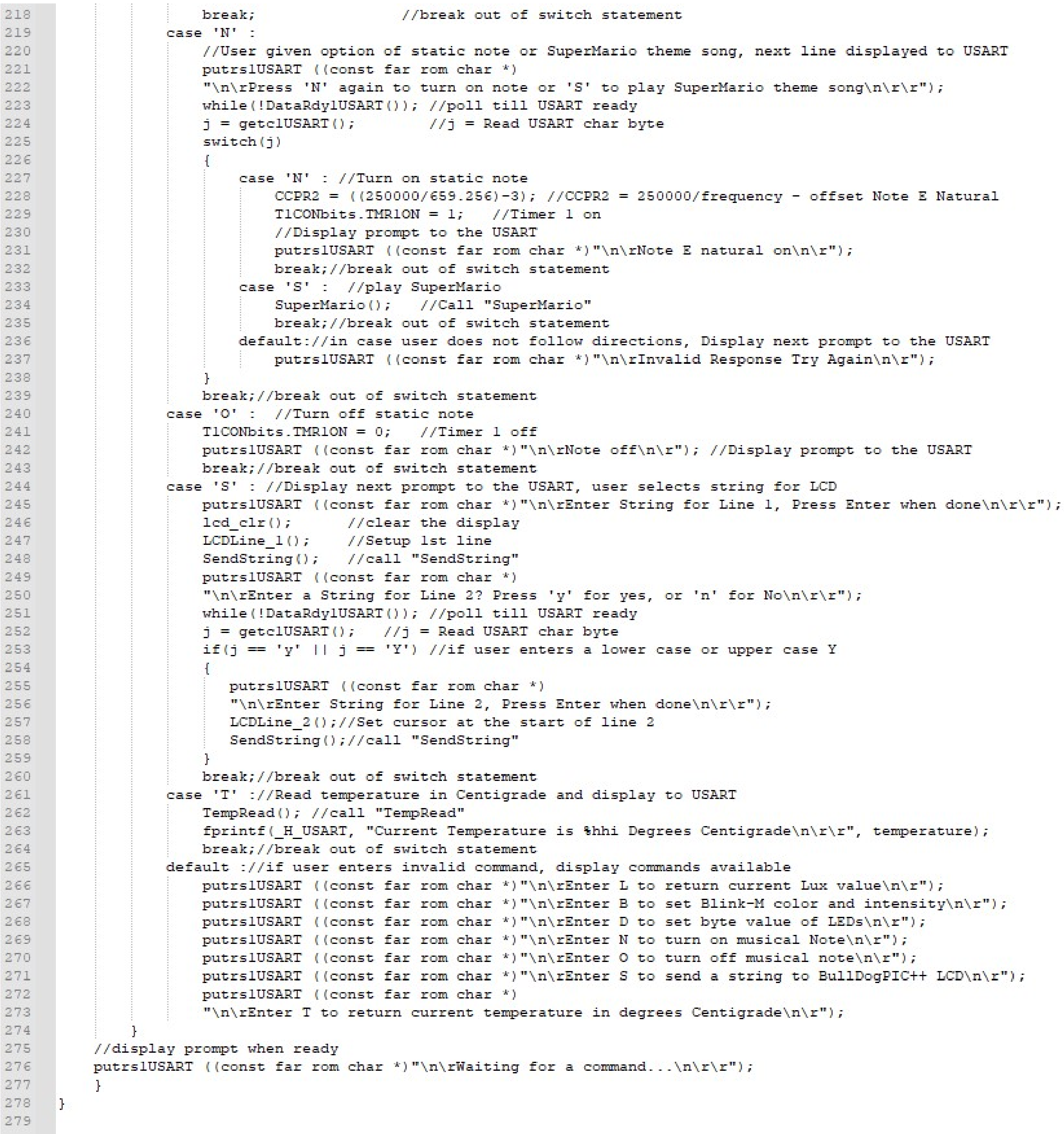
```c
/*         __                          __
          /'{>                        /'{>
      ____) (____                 ____) (____
    //'--;   ;--'\\             //'--;   ;--'\\
   ///////\_/\\\\\\\           ///////\_/\\\\\\\
          m m                         m m
        *********** PIC18F46K80 ***********
                +--------U--------+
       MCLR/RE3 |1              40| RB7  PICKit & LED Bit 7
      POT - RA0 |2              39| RB6  PICKit & LED Bit 6
Light Sens  RA1 |3              38| RB5  LED Bit 5
Analog 2    RA2 |4              37| RB4  LED Bit 4
Analog 3    RA3 |5              36| RB3  LED Bit 3
       VDD Core |6              35| RB2  LED Bit 2
Switch S2   RA5 |7              34| RB1  LED Bit 1
<not used>  RE0 |8              33| RB0  LED Bit 0 & Switch S5
<not used>  RE1 |9              32| VDD
<not used>  RE2 |10             31| GND
            VDD |11             30| RD7 (Rx2)-> Serial Data OUT*  Bluetooth
            GND |12             29| RD6 (Tx2)-> Serial Clock OUT* Bluetooth
  Switch S4 RA7 |13             28| RD5 To LCD
  Switch S3 RA6 |14             27| RD4 To LCD
  To LCD    RC0 |15             26| RC7 (Rx1)-> Serial Data OUT*  USB
  To LCD    RC1 |16             25| RC6 (Tx1)-> Serial Clock OUT* USB
  Buzzer    RC2 |17             24| RC5 <Not Used>
  I2C SCLK  RC3 |18             23| RC4 I2C SDA
  To LCD    RD0 |19             22| RD3 To LCD
  To LCD    RD1 |20             21| RD2 To LCD
                +-----------------+
* Note:  RA4 not available since used by VDD Core!
* for configurations of master synch serial port, see 18.3 of PDF
* YES - Rx IS SERIAL OUT!!!!
*/

#include <stdio.h>
#include <stdlib.h>
#include <p18f46K80.h>
#include <delays.h>
#include <timers.h>
#include <usart.h>
#include <i2c.h>

// see hlpPIC18ConfigSet document in the C18 director for info on configs below
#pragma config FOSC = INTIO2    /* internal osc enable */
#pragma config FCMEN = OFF      /* Failsafe Clock Monitor Disabled */
#pragma config WDTEN = OFF      /* no Dog */
#pragma config SOSCSEL = DIG    /* Configures PORTC0&1 for digital I/O */
#pragma config XINST = OFF      /* Extended Instruction Set off */

extern void LCDInit(void);      /* LCD initialize    */
extern void lcd_clr(void);      /* clear the lcd     */
extern void LCDLine_1(void);    /* get LCD to line 1 */
extern void LCDLine_2(void);    /* get LCD to line 2 */
extern void end_line(void);     /* end of line       */
extern void d_write(void);      /* write data to LCD */
extern void i_write(void);      /* position lcd curs.*/
extern unsigned int temp_wr;    /* pass ascii char   */

//****************************************************************
//*                                                              *
//*  ECNS-424                                                    *
//*  18 April 2021                                               *
//*  In Depth Lab-4                                              *
//*                                                              *
//*  The purpose of this program is to use Asynchronous          *
//*  communication via the on chip USART to an external serial   *
//*  monitor running on a lap or desktop computer                *
//*                                                              *
//*  Created by: Brandon Empie                                   *
//*                                                              *
//****************************************************************
//24 notes for dividing the SuperMario theme into two parts
int NoteArray[] =
```



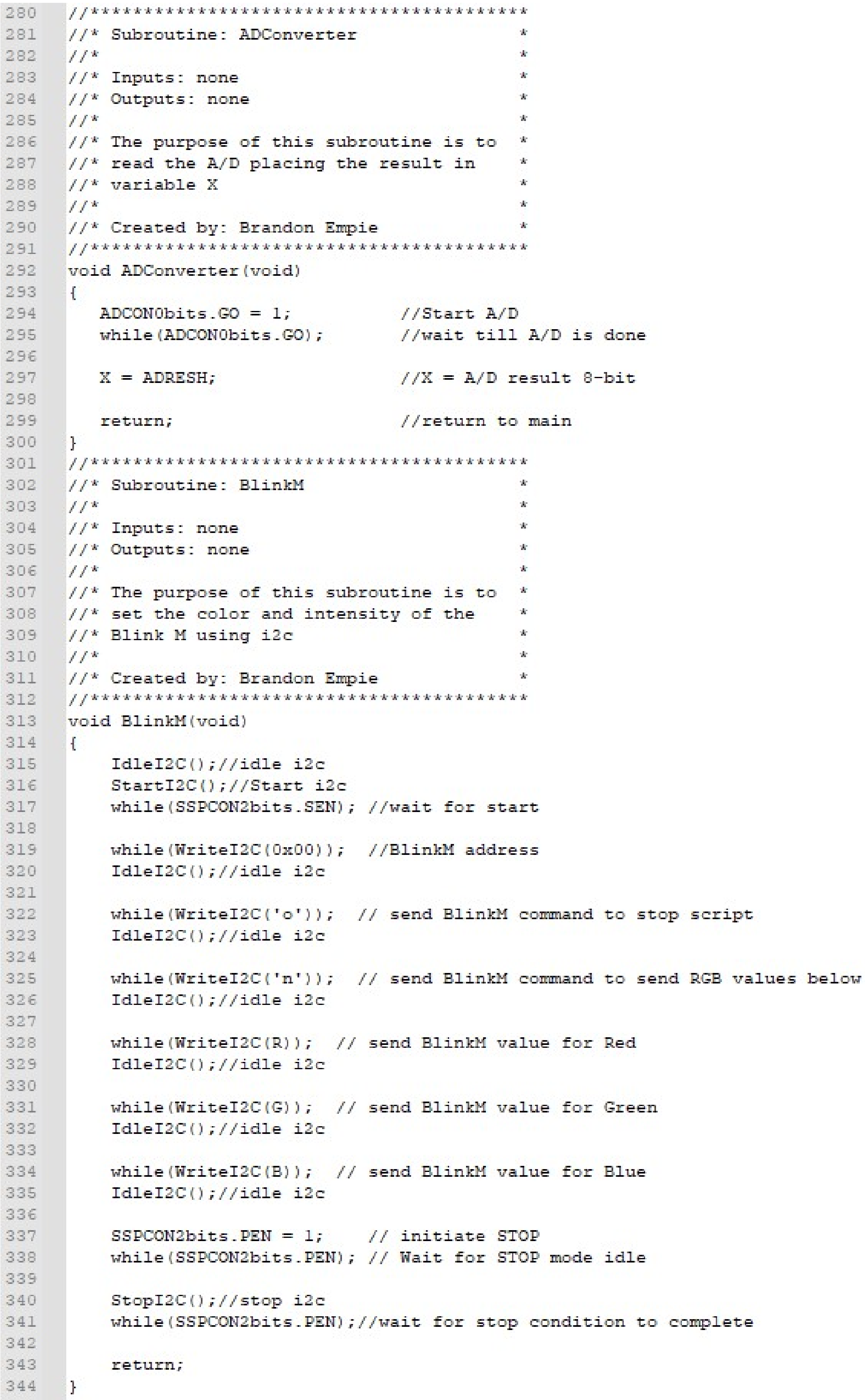
```
int NoteArray[] =
{660,660,660,510,660,770,380,510,380,320,440,480,450,430,380,660,760,860,700,760,660,520,580,480};
int NoteLength[] = {20,20,20,20,20,20,20,20,20,20,20,16,20,20,20,16,10,20,16,10,16,16,16,16};
int delay[] = {30,60,60,20,60,110,115,90,80,100,60,66,30,60,40,40,30,60,30,70,60,30,30,100};
//21 notes for the second portion of SuperMario theme
const rom int NoteArrayA[] =
{500,760,720,680,620,650,380,430,500,430,500,570,500,760,720,680,620,650,1020,1020,1020};
const rom int NoteLengthA[] = {20,20,20,20,30,30,20,20,20,20,20,20,20,20,20,20,30,40,16,16,16};
const rom int delayA[] = {60,20,30,30,60,60,30,30,60,30,20,44,60,20,30,30,60,60,60,30,60};

void ADConverter(void);    //function prototype for subroutine "ADConverter"
void BlinkM(void);         //function prototype for subroutine "BlinkM"
void TempRead(void);       //function prototype for subroutine "TempRead"
void Color(void);          //function prototype for subroutine "Color"
void LEDs(void);           //function prototype for subroutine "LEDs"
void SendString(void);     //function prototype for subroutine "SendString"
void SuperMario(void);     //function prototype for subroutine "SuperMario"
void Compare(void);        //Compare ISR function prototype

unsigned char result;      //main switch statement variable
unsigned char BlinkMColor; //Blink-M switch statement variable
unsigned int Intensity;    //Blink-M intensity variable
unsigned long X = 0; //init. variable X, A/D result variable
unsigned long Y = 0; //init. variable Y, Voltage result variable
unsigned long Z = 0; //init. variable Z, Lux result variable
unsigned int R = 0; //initialize variable R
unsigned int G = 0; //initialize variable G
unsigned int B = 0; //initialize variable B
unsigned int H = 0; //Hundreds place BlinkM
unsigned int T = 0; //Tens place BlinkM
unsigned int O = 0; //Ones place BlinkM
unsigned int HexUpper;//variable for user entered upper byte
unsigned int HexLower;//variable for user entered lower byte
char LCDString[16];   //Array for user entered Strings
unsigned int S = 0;   //String pointer variable
char done = 0;        //variable used to check if user has pressed Enter key
unsigned int ArraySize = 0; //keeps track of how many chars need to be sent to LCD
unsigned int j;     //Generic loop variable
unsigned int ready = 1;//Used to check if the TC-74 Temperature Sensor was ready to be read
signed char temperature; //TC-74 temp. sensor read in variable
signed char read_data = 0; // Signed 8 bits -127 +128 deg Celcius.
unsigned char cmd = 1;  // Select TC74 Config Reg.

#pragma code high_vector=0x08   // tells compiler that this belongs @ loc 0x08
void high_interrupt (void)
{
  _asm GOTO Compare _endasm // inline assembly for interrupt vector
}
#pragma code              // normal code area from here down
#pragma interrupt Compare //declare interrupt Compare code to follow

//****************************************************************
//*                                                              *
//*  ISR: Compare                                                *
//*  In:  none                                                   *
//*  Out: none                                                   *
//*                                                              *
//*  The purpose of this Interrupt Service Routine is to create  *
//*  the generated frequency via capture compare                 *
//*                                                              *
//*  Created by: Brandon Empie                                   *
//*                                                              *
//****************************************************************
void Compare(void)
{
    if(PIR3bits.CCP2IF = 1) // check to see if CCP2IF caused the interrupt
    {
    TMR1H = 0x00;           //Clear Timer 1 high byte
    TMR1L = 0x00;           //Clear Timer 1 Low byte
    PIR3bits.CCP2IF = 0;    //Clear CCP2IF Flag for next go-round
    }
}
```



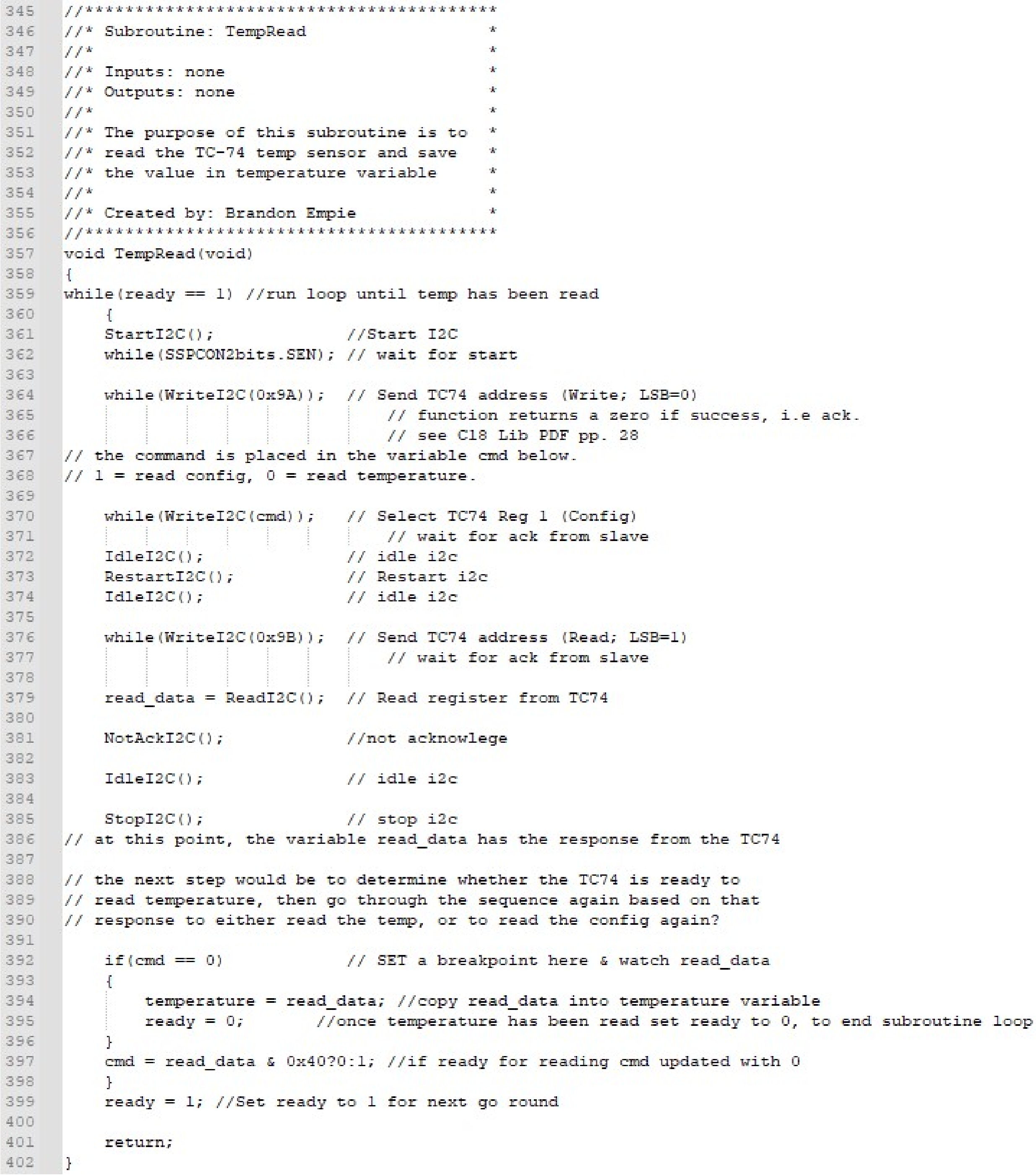
```
//////////////////////////////////////////////////////////////////
//                          MAIN                                //
//////////////////////////////////////////////////////////////////
void main (void)
{
    //init I/O below
    OSCCON = 0x60;  //set to 8 MHz base clock internal (default clock)
    TRISB = 0;      //PORT B all outputs
    TRISC = 0xF9;   //PORT C config
    TRISA = 0xFF;   //PORTA all inputs
    LATB = 0;       //All outputs off for safety

    PORTAbits.RA1 = 1;    //Light sensor channel set to analog
    ADCON0bits.CHS = 1;   //A/D now set to channel 01 (AN1) for light sensor
    ADCON2bits.ADFM = 0;  //Left Justify A/D
    ADCON2bits.ADCS = 4;  //FOSC/4
    ADCON2bits.ACQT = 2;  //A/D acquisition time select 4 TAD
    ADCON0bits.ADON = 1;  //Turn on A/D

    OpenI2C(MASTER, SLEW_OFF); //Init I2C
    TRISCbits.TRISC3 = 1;   //I2 C clock output (MSSP module); takes priority over port data.
    TRISCbits.TRISC4 = 1;   //I2C data input/output (MSSP module); input type depends on module setting.
    SSPCON1 = 0x28;         //SSP enable(bit5)Master mode(bit3-0)->0000 = SPI Master mode,clock = FOSC/4
    SSPSTATbits.SMP = 1;    //Input data sampled at end of data output time
    SSPSTATbits.CKE = 1;    //Transmit occurs on transition from active to Idle clock state
    SSPADD = 0x27;          //sets up baud rate at 50kHz = 8MHz/(4*(39 + 1))

    Open1USART (USART_TX_INT_OFF &
                USART_RX_INT_OFF &
                USART_ASYNCH_MODE &
                USART_EIGHT_BIT &        //Error rate = (9615-9600)/9600 = .16%
                USART_CONT_RX &          //Baud = 8MHz/(16(51+1))
                USART_BRGH_HIGH, 51);    // sets baud to 9600 @ 8 MHz
    BlinkM(); //call "blinkM" to turn blinkM off on startup

    TRISCbits.TRISC2 = 0;   //RC2 output for buzzer on CCP2
    PORTCbits.RC2 = 1;      //Buzzer channel set to analog
    CCP2CON = 0x02;         //toggle mode
    PIE3bits.CCP2IE = 1;    //enable compare interrupt
    RCONbits.IPEN = 1;      //enable priority interrupts (both low and hi)
    INTCONbits.GIEH = 1;    //enables high-priority interrupts
    IPR3bits.CCP2IP = 1;    //set Compare 2 interrupt to high priority
    T1CONbits.T1CKPS = 2;   //prescaler of 4 * Fosc/4 to get pre of 16
    T1CONbits.RD16 = 1;     //Timer 1 16-bit mode
    T1CONbits.TMR1ON = 0;   //Timer 1 off
    CCPR2 = ((250000/NoteLength[j])-3); //Init. CCPR2 = 250000/frequency - offset

    //Display startup prompt to the USART
    putrs1USART ((const far rom char *)"\n\rECNS-424 In Depth Lab #4, by Brandon Empie\n\r");
    putrs1USART ((const far rom char *)"\rWaiting for a command...\n\r\r");

    LCDInit();      //Initialize LCD
    lcd_clr();      //clear the display

    while (1)
    {
        while(!DataRdy1USART()); //poll till USART ready
        result = getc1USART();   //Result = Read USART char byte
        switch(result)
        {
            case 'L' :                //Return Lux value if L is entered
                ADConverter();        //call "ADConverter"
                Y = ((X*500)/255);
                //Y(Vin) = ((A/D result)* 500)/255(8-bit)
                //(5V, 500 causes Y value to be ex. 320 instead of 3.2 to bypass floating point)
                Z = ((Y*1000)/224);   // Z now = Lux (lux value is based on 2.240mV/Lux)
                fprintf(_H_USART, "Current Lux value is:  %lu\n\r\r", Z);//Display lux value to the USART
                break;//break out of switch statement
            case 'B' :                //accept color and R,G,B intensity (0-255)
                Color();              //Call "Color"
                break;                //break out of switch statement
            case 'D' :                //User selects hex byte to be displayed on PIC LEDs
                LEDs();               //Call "LEDs"
```



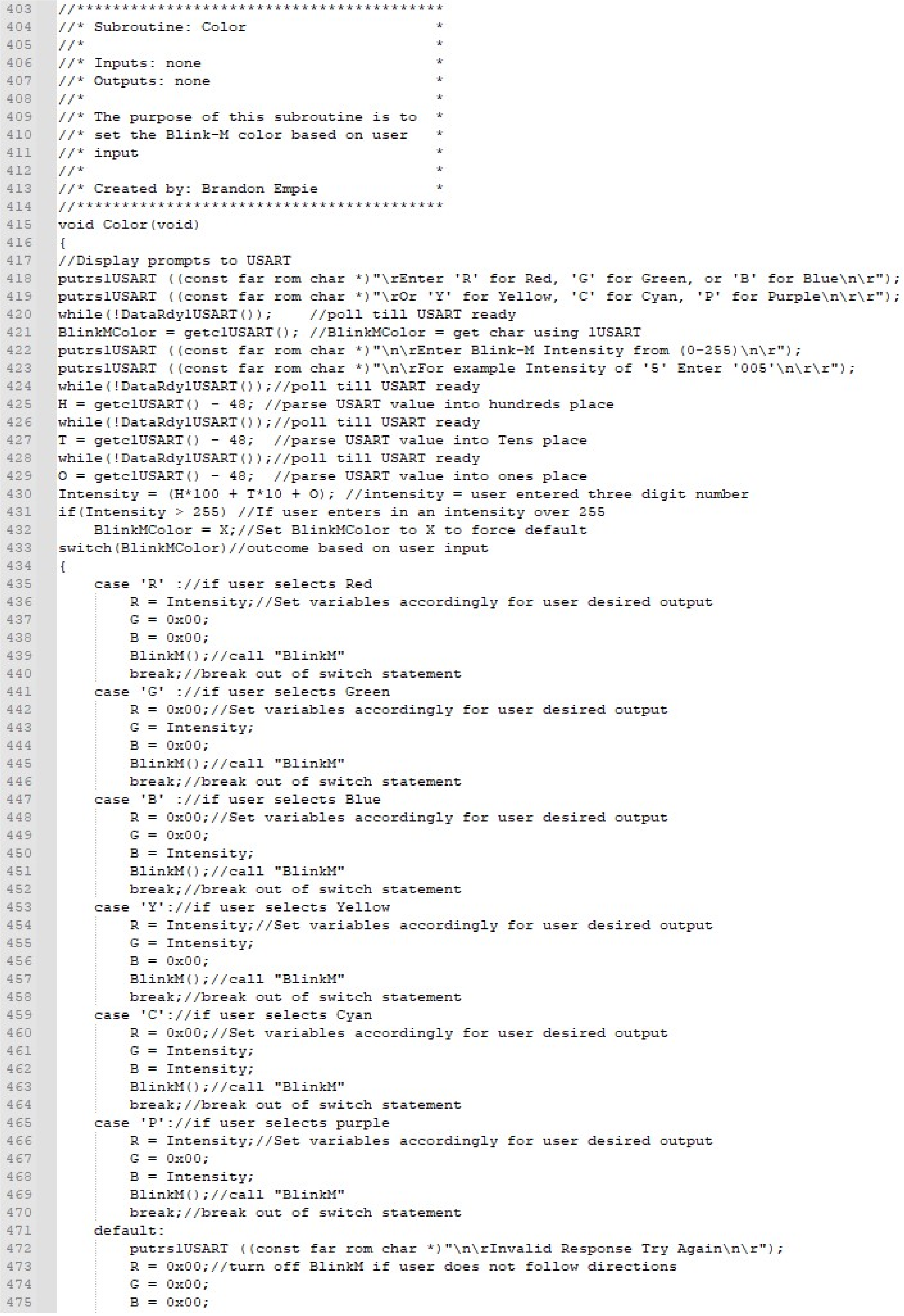
```
                break;                //break out of switch statement
            case 'N' :
                //User given option of static note or SuperMario theme song, next line displayed to USART
                putrs1USART ((const far rom char *)
                "\n\rPress 'N' again to turn on note or 'S' to play SuperMario theme song\n\r\r");
                while(!DataRdy1USART()); //poll till USART ready
                j = getc1USART();        //j = Read USART char byte
                switch(j)
                {
                    case 'N' : //Turn on static note
                        CCPR2 = ((250000/659.256)-3); //CCPR2 = 250000/frequency - offset Note E Natural
                        T1CONbits.TMR1ON = 1;   //Timer 1 on
                        //Display prompt to the USART
                        putrs1USART ((const far rom char *)"\n\rNote E natural on\n\r");
                        break;//break out of switch statement
                    case 'S' :  //play SuperMario
                        SuperMario();   //Call "SuperMario"
                        break;//break out of switch statement
                    default://in case user does not follow directions, Display next prompt to the USART
                        putrs1USART ((const far rom char *)"\n\rInvalid Response Try Again\n\r");
                }
                break;//break out of switch statement
            case 'O' :  //Turn off static note
                T1CONbits.TMR1ON = 0;   //Timer 1 off
                putrs1USART ((const far rom char *)"\n\rNote off\n\r"); //Display prompt to the USART
                break;//break out of switch statement
            case 'S' : //Display next prompt to the USART, user selects string for LCD
                putrs1USART ((const far rom char *)"\n\rEnter String for Line 1, Press Enter when done\n\r\r");
                lcd_clr();      //clear the display
                LCDLine_1();    //Setup 1st line
                SendString();   //call "SendString"
                putrs1USART ((const far rom char *)
                "\n\rEnter a String for Line 2? Press 'y' for yes, or 'n' for No\n\r\r");
                while(!DataRdy1USART()); //poll till USART ready
                j = getc1USART();   //j = Read USART char byte
                if(j == 'y' || j == 'Y') //if user enters a lower case or upper case Y
                {
                   putrs1USART ((const far rom char *)
                   "\n\rEnter String for Line 2, Press Enter when done\n\r\r");
                   LCDLine_2();//Set cursor at the start of line 2
                   SendString();//call "SendString"
                }
                break;//break out of switch statement
            case 'T' ://Read temperature in Centigrade and display to USART
                TempRead(); //call "TempRead"
                fprintf(_H_USART, "Current Temperature is %hhi Degrees Centigrade\n\r\r", temperature);
                break;//break out of switch statement
            default ://if user enters invalid command, display commands available
                putrs1USART ((const far rom char *)"\n\rEnter L to return current Lux value\n\r");
                putrs1USART ((const far rom char *)"\n\rEnter B to set Blink-M color and intensity\n\r");
                putrs1USART ((const far rom char *)"\n\rEnter D to set byte value of LEDs\n\r");
                putrs1USART ((const far rom char *)"\n\rEnter N to turn on musical Note\n\r");
                putrs1USART ((const far rom char *)"\n\rEnter O to turn off musical note\n\r");
                putrs1USART ((const far rom char *)"\n\rEnter S to send a string to BullDogPIC++ LCD\n\r");
                putrs1USART ((const far rom char *)
                "\n\rEnter T to return current temperature in degrees Centigrade\n\r");
        }
    //display prompt when ready
    putrs1USART ((const far rom char *)"\n\rWaiting for a command...\n\r\r");
    }
}
```



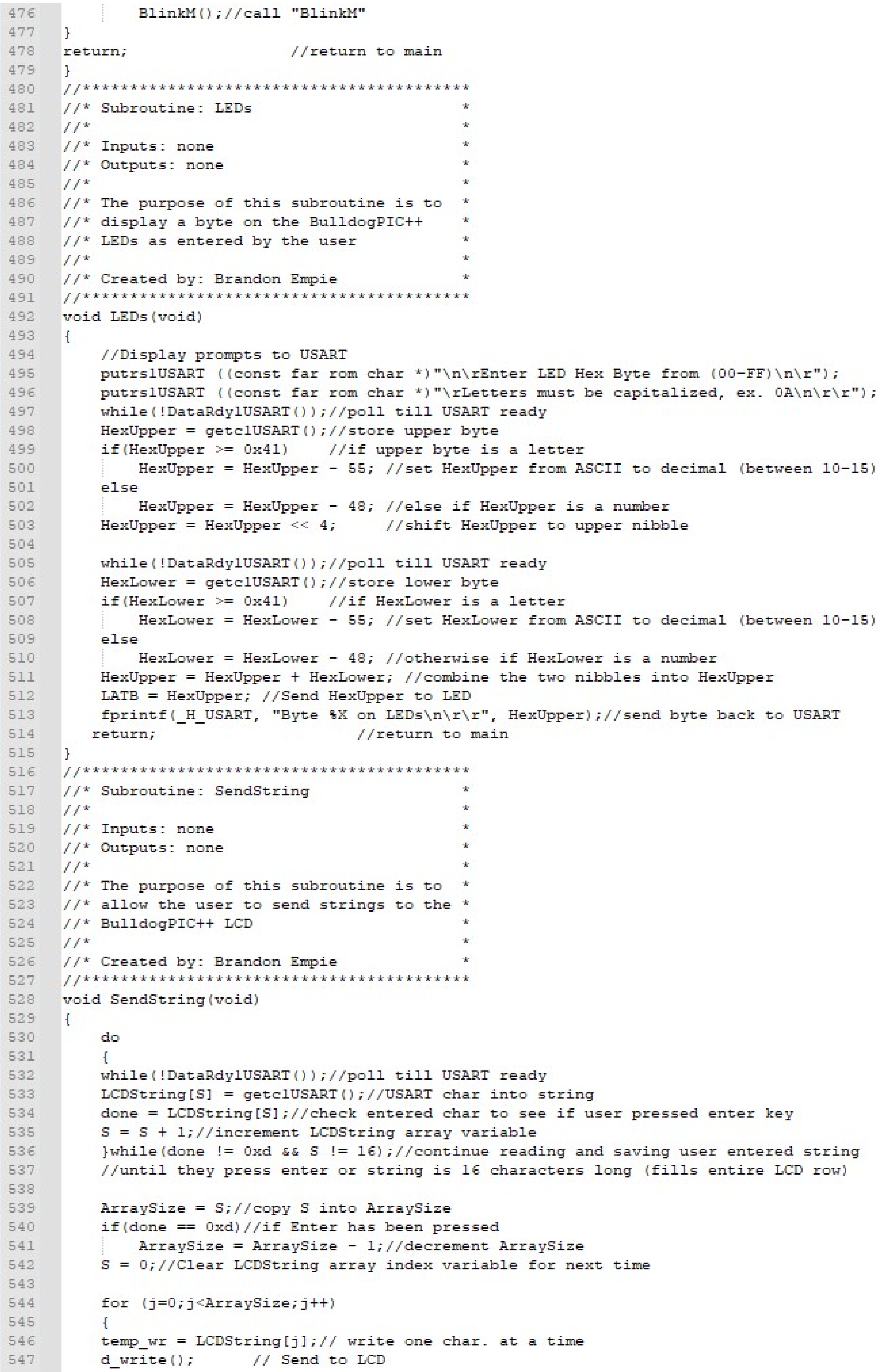
```
//*****************************************
//* Subroutine: ADConverter               *
//*                                       *
//* Inputs: none                          *
//* Outputs: none                         *
//*                                       *
//* The purpose of this subroutine is to  *
//* read the A/D placing the result in    *
//* variable X                            *
//*                                       *
//* Created by: Brandon Empie             *
//*****************************************
void ADConverter(void)
{
   ADCON0bits.GO = 1;          //Start A/D
   while(ADCON0bits.GO);       //wait till A/D is done

   X = ADRESH;                 //X = A/D result 8-bit

   return;                     //return to main
}
//*****************************************
//* Subroutine: BlinkM                    *
//*                                       *
//* Inputs: none                          *
//* Outputs: none                         *
//*                                       *
//* The purpose of this subroutine is to  *
//* set the color and intensity of the    *
//* Blink M using i2c                     *
//*                                       *
//* Created by: Brandon Empie             *
//*****************************************
void BlinkM(void)
{
    IdleI2C();//idle i2c
    StartI2C();//Start i2c
    while(SSPCON2bits.SEN); //wait for start

    while(WriteI2C(0x00));  //BlinkM address
    IdleI2C();//idle i2c

    while(WriteI2C('o'));  // send BlinkM command to stop script
    IdleI2C();//idle i2c

    while(WriteI2C('n'));  // send BlinkM command to send RGB values below
    IdleI2C();//idle i2c

    while(WriteI2C(R));  // send BlinkM value for Red
    IdleI2C();//idle i2c

    while(WriteI2C(G));  // send BlinkM value for Green
    IdleI2C();//idle i2c

    while(WriteI2C(B));  // send BlinkM value for Blue
    IdleI2C();//idle i2c

    SSPCON2bits.PEN = 1;    // initiate STOP
    while(SSPCON2bits.PEN); // Wait for STOP mode idle

    StopI2C();//stop i2c
    while(SSPCON2bits.PEN);//wait for stop condition to complete

    return;
}
```



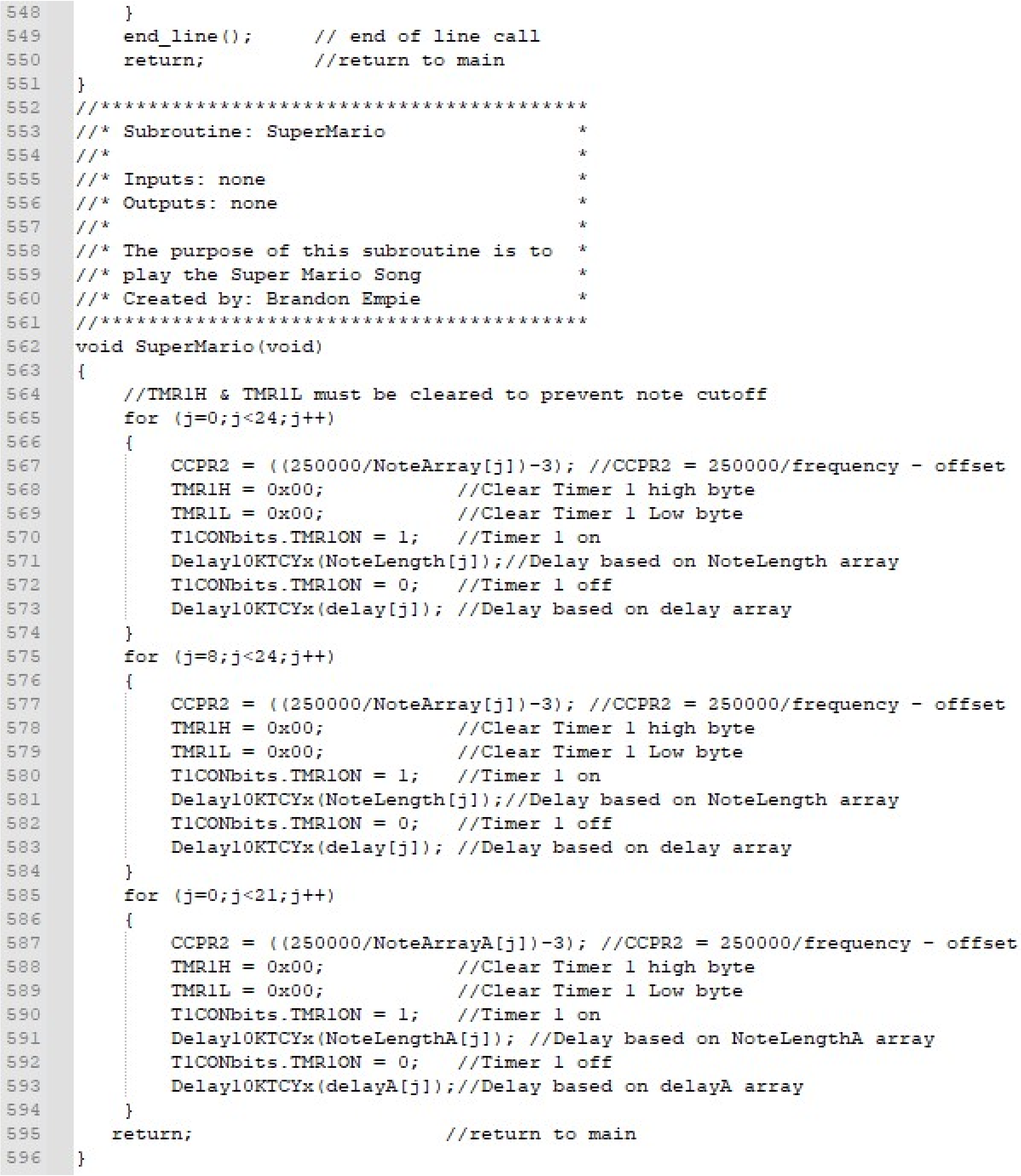
```
//*****************************************
//* Subroutine: TempRead                  *
//*                                       *
//* Inputs: none                          *
//* Outputs: none                         *
//*                                       *
//* The purpose of this subroutine is to  *
//* read the TC-74 temp sensor and save   *
//* the value in temperature variable     *
//*                                       *
//* Created by: Brandon Empie             *
//*****************************************
void TempRead(void)
{
while(ready == 1) //run loop until temp has been read
    {
    StartI2C();             //Start I2C
    while(SSPCON2bits.SEN); // wait for start

    while(WriteI2C(0x9A));  // Send TC74 address (Write; LSB=0)
                                // function returns a zero if success, i.e ack.
                                // see C18 Lib PDF pp. 28
// the command is placed in the variable cmd below.
// 1 = read config, 0 = read temperature.

    while(WriteI2C(cmd));   // Select TC74 Reg 1 (Config)
                                // wait for ack from slave
    IdleI2C();              // idle i2c
    RestartI2C();           // Restart i2c
    IdleI2C();              // idle i2c

    while(WriteI2C(0x9B));  // Send TC74 address (Read; LSB=1)
                                // wait for ack from slave

    read_data = ReadI2C();  // Read register from TC74

    NotAckI2C();            //not acknowlege

    IdleI2C();              // idle i2c

    StopI2C();              // stop i2c
// at this point, the variable read_data has the response from the TC74

// the next step would be to determine whether the TC74 is ready to
// read temperature, then go through the sequence again based on that
// response to either read the temp, or to read the config again?

    if(cmd == 0)            // SET a breakpoint here & watch read_data
    {
        temperature = read_data; //copy read_data into temperature variable
        ready = 0;       //once temperature has been read set ready to 0, to end subroutine loop
    }
    cmd = read_data & 0x40?0:1; //if ready for reading cmd updated with 0
    }
    ready = 1; //Set ready to 1 for next go round

    return;
}
```



```
//****************************************
//* Subroutine: Color                    *
//*                                      *
//* Inputs: none                         *
//* Outputs: none                        *
//*                                      *
//* The purpose of this subroutine is to  *
//* set the Blink-M color based on user   *
//* input                                *
//*                                      *
//* Created by: Brandon Empie            *
//****************************************
void Color(void)
{
//Display prompts to USART
putrs1USART ((const far rom char *)"\rEnter 'R' for Red, 'G' for Green, or 'B' for Blue\n\r");
putrs1USART ((const far rom char *)"\rOr 'Y' for Yellow, 'C' for Cyan, 'P' for Purple\n\r\r");
while(!DataRdy1USART());    //poll till USART ready
BlinkMColor = getc1USART(); //BlinkMColor = get char using 1USART
putrs1USART ((const far rom char *)"\n\rEnter Blink-M Intensity from (0-255)\n\r");
putrs1USART ((const far rom char *)"\n\rFor example Intensity of '5' Enter '005'\n\r\r");
while(!DataRdy1USART());//poll till USART ready
H = getc1USART() - 48; //parse USART value into hundreds place
while(!DataRdy1USART());//poll till USART ready
T = getc1USART() - 48;  //parse USART value into Tens place
while(!DataRdy1USART());//poll till USART ready
O = getc1USART() - 48;  //parse USART value into ones place
Intensity = (H*100 + T*10 + O); //intensity = user entered three digit number
if(Intensity > 255) //If user enters in an intensity over 255
    BlinkMColor = X;//Set BlinkMColor to X to force default
switch(BlinkMColor)//outcome based on user input
{
    case 'R' ://if user selects Red
        R = Intensity;//Set variables accordingly for user desired output
        G = 0x00;
        B = 0x00;
        BlinkM();//call "BlinkM"
        break;//break out of switch statement
    case 'G' ://if user selects Green
        R = 0x00;//Set variables accordingly for user desired output
        G = Intensity;
        B = 0x00;
        BlinkM();//call "BlinkM"
        break;//break out of switch statement
    case 'B' ://if user selects Blue
        R = 0x00;//Set variables accordingly for user desired output
        G = 0x00;
        B = Intensity;
        BlinkM();//call "BlinkM"
        break;//break out of switch statement
    case 'Y'://if user selects Yellow
        R = Intensity;//Set variables accordingly for user desired output
        G = Intensity;
        B = 0x00;
        BlinkM();//call "BlinkM"
        break;//break out of switch statement
    case 'C'://if user selects Cyan
        R = 0x00;//Set variables accordingly for user desired output
        G = Intensity;
        B = Intensity;
        BlinkM();//call "BlinkM"
        break;//break out of switch statement
    case 'P'://if user selects purple
        R = Intensity;//Set variables accordingly for user desired output
        G = 0x00;
        B = Intensity;
        BlinkM();//call "BlinkM"
        break;//break out of switch statement
    default:
        putrs1USART ((const far rom char *)"\n\rInvalid Response Try Again\n\r");
        R = 0x00;//turn off BlinkM if user does not follow directions
        G = 0x00;
        B = 0x00;
```



```
        BlinkM();//call "BlinkM"
}
return;                 //return to main
}
//*****************************************
//* Subroutine: LEDs                      *
//*                                       *
//* Inputs: none                          *
//* Outputs: none                         *
//*                                       *
//* The purpose of this subroutine is to  *
//* display a byte on the BulldogPIC++    *
//* LEDs as entered by the user           *
//*                                       *
//* Created by: Brandon Empie             *
//*****************************************
void LEDs(void)
{
    //Display prompts to USART
    putrs1USART ((const far rom char *)"\n\rEnter LED Hex Byte from (00-FF)\n\r");
    putrs1USART ((const far rom char *)"\rLetters must be capitalized, ex. 0A\n\r\r");
    while(!DataRdy1USART());//poll till USART ready
    HexUpper = getc1USART();//store upper byte
    if(HexUpper >= 0x41)    //if upper byte is a letter
        HexUpper = HexUpper - 55; //set HexUpper from ASCII to decimal (between 10-15)
    else
        HexUpper = HexUpper - 48; //else if HexUpper is a number
    HexUpper = HexUpper << 4;     //shift HexUpper to upper nibble

    while(!DataRdy1USART());//poll till USART ready
    HexLower = getc1USART();//store lower byte
    if(HexLower >= 0x41)    //if HexLower is a letter
        HexLower = HexLower - 55; //set HexLower from ASCII to decimal (between 10-15)
    else
        HexLower = HexLower - 48; //otherwise if HexLower is a number
    HexUpper = HexUpper + HexLower; //combine the two nibbles into HexUpper
    LATB = HexUpper; //Send HexUpper to LED
    fprintf(_H_USART, "Byte %X on LEDs\n\r\r", HexUpper);//send byte back to USART
   return;                      //return to main
}
//*****************************************
//* Subroutine: SendString                *
//*                                       *
//* Inputs: none                          *
//* Outputs: none                         *
//*                                       *
//* The purpose of this subroutine is to  *
//* allow the user to send strings to the *
//* BulldogPIC++ LCD                      *
//*                                       *
//* Created by: Brandon Empie             *
//*****************************************
void SendString(void)
{
    do
    {
    while(!DataRdy1USART());//poll till USART ready
    LCDString[S] = getc1USART();//USART char into string
    done = LCDString[S];//check entered char to see if user pressed enter key
    S = S + 1;//increment LCDString array variable
    }while(done != 0xd && S != 16);//continue reading and saving user entered string
    //until they press enter or string is 16 characters long (fills entire LCD row)

    ArraySize = S;//copy S into ArraySize
    if(done == 0xd)//if Enter has been pressed
        ArraySize = ArraySize - 1;//decrement ArraySize
    S = 0;//Clear LCDString array index variable for next time

    for (j=0;j<ArraySize;j++)
    {
    temp_wr = LCDString[j];// write one char. at a time
    d_write();      // Send to LCD
```



```
    }
    end_line();     // end of line call
    return;         //return to main
}
//*****************************************
//* Subroutine: SuperMario                *
//*                                       *
//* Inputs: none                          *
//* Outputs: none                         *
//*                                       *
//* The purpose of this subroutine is to  *
//* play the Super Mario Song             *
//* Created by: Brandon Empie             *
//*****************************************
void SuperMario(void)
{
    //TMR1H & TMR1L must be cleared to prevent note cutoff
    for (j=0;j<24;j++)
    {
        CCPR2 = ((250000/NoteArray[j])-3); //CCPR2 = 250000/frequency - offset
        TMR1H = 0x00;           //Clear Timer 1 high byte
        TMR1L = 0x00;           //Clear Timer 1 Low byte
        T1CONbits.TMR1ON = 1;   //Timer 1 on
        Delay10KTCYx(NoteLength[j]);//Delay based on NoteLength array
        T1CONbits.TMR1ON = 0;   //Timer 1 off
        Delay10KTCYx(delay[j]); //Delay based on delay array
    }
    for (j=8;j<24;j++)
    {
        CCPR2 = ((250000/NoteArray[j])-3); //CCPR2 = 250000/frequency - offset
        TMR1H = 0x00;           //Clear Timer 1 high byte
        TMR1L = 0x00;           //Clear Timer 1 Low byte
        T1CONbits.TMR1ON = 1;   //Timer 1 on
        Delay10KTCYx(NoteLength[j]);//Delay based on NoteLength array
        T1CONbits.TMR1ON = 0;   //Timer 1 off
        Delay10KTCYx(delay[j]); //Delay based on delay array
    }
    for (j=0;j<21;j++)
    {
        CCPR2 = ((250000/NoteArrayA[j])-3); //CCPR2 = 250000/frequency - offset
        TMR1H = 0x00;           //Clear Timer 1 high byte
        TMR1L = 0x00;           //Clear Timer 1 Low byte
        T1CONbits.TMR1ON = 1;   //Timer 1 on
        Delay10KTCYx(NoteLengthA[j]); //Delay based on NoteLengthA array
        T1CONbits.TMR1ON = 0;   //Timer 1 off
        Delay10KTCYx(delayA[j]);//Delay based on delayA array
    }
   return;                     //return to main
}
```