```
/*
          __                          __
         /'{>                        /'{>
      ____) (____                 ____) (____
    //'--;   ;--'\\             //'--;   ;--'\\
   ///////\_/\\\\\\\           ///////\_/\\\\\\\
         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>


#pragma config FOSC = INTIO2    /* internal osc enable */
#pragma config FCMEN = OFF      /* Failsafe Clock Monitor Disabled */
#pragma config WDTEN = OFF      /* no Dog */
#pragma config XINST = OFF      /* Extended Instruction Set off */
#pragma config SOSCSEL = DIG    /* Configures PORTC0&1 for digital I/O */

//*****************************************************************
//*                                                               *
//*  ECNS-424                                                     *
//*  9 April 2021                                                 *
//*  In Depth Lab-3                                               *
//*                                                               *
//*  The purpose of this program is to use inter-integrated       *
//*  circuit communication to connect a 32kx8 serial EEPROM,      *
//*  TC-74 Temperature Sensor, and Blink-M Smart LED              *
//*                                                               *
//*  Created by: Brandon Empie                                    *
//*                                                               *
//*****************************************************************
unsigned int Y = 0; //initialize variable Y
unsigned int X = 0; //initialize variable X
unsigned int Z = 0; //initialize variable Z
unsigned int j;     //splash screen loop variable
unsigned int H = 0;  //init. variable H (Hundreds)
unsigned int T = 0;  //init. variable T (Tens)
unsigned int O = 0;  //init. variable O (Ones)
unsigned int R = 0; //initialize variable R
unsigned int G = 0; //initialize variable G
unsigned int B = 0; //initialize variable B
unsigned int clear = 1;//EEPROM clear variable
unsigned int ready = 1;//Used to check if the TC-74 Temperature Sensor was ready to be read

signed char TT;//Current Temperature Tens var
signed char TO;//Current Temperature Ones var

signed char HT;//High Temperature Tens var
signed char HO;//High Temperature Ones var

signed char LT;//Low Temperature Tens var
signed char LO;//Low Temperature Ones var
```

```
84
85
86  const unsigned int ADDRTHigh = 0x30; //High temp address lower byte
87  const unsigned int ADDRTLow = 0x20;  //Low temp address lower byte
88  unsigned int addr = 0x30; //address variable initilized for high temp address
89
90  signed char THigh; //Temperature high value variable
91  signed char TLow;  //Temperature low value variable
92  unsigned int TStatus = 0; //Status variables used to indicate if temperature read was negative (1) or positive
    (0)
93  unsigned int HStatus = 0; //High temp status
94  unsigned int LStatus = 0; //Low temp status
95
96  signed char storage; //EEProm read variable
97  signed char temperature; //TC-74 temp. sensor read in variable
98  signed char read_data = 0;     // Signed 8 bits -127 +128 deg Celcius.
99  unsigned char cmd = 1;  // Select TC74 Config Reg.
100
101 const char line1[7]={"BlinkM:"};//Line2 constant for splash screen
102 const char Red[6]={" Red  "}; //Constant used to set LCD
103 const char Green[6]={" Green"};//Constant used to set LCD
104 const char Blue[6]={" Blue "};//Constant used to set LCD
105
106 const char Current[11]={"Temp Now:  "};//Constant used to set LCD
107 const char High[11]={"High Temp: "};//Constant used to set LCD
108 const char Low[11]={"Low Temp:  "};//Constant used to set LCD
109
110 extern void LCDInit(void);      /* LCD initialize    */
111 extern void lcd_clr(void);      /* clear the lcd     */
112 extern void LCDLine_1(void);    /* get LCD to line 1 */
113 extern void LCDLine_2(void);    /* get LCD to line 2 */
114 extern void end_line(void);     /* end of line       */
115 extern void d_write(void);      /* write data to LCD */
116 extern void i_write(void);      /* position lcd curs.*/
117 extern unsigned int temp_wr;    /* pass ascii char   */
118
119 void read(void);        //function prototypes
120 void write(void);       //function prototypes
121 void BlinkM(void);      //function prototypes
122 void TempRead(void);    //function prototypes
123 void smdelay(void);     //function prototypes
124 void switches(void);    //function prototype
125 void Showoff(void);     //function prototype
126
127 void Blink(void);       //function prototype for ISR "Blink"
128 #pragma code low_vector=0x18    // tells compiler that this belongs @ loc 0x18
129 void low_interrupt (void)
130 {
131   _asm GOTO Blink _endasm   // inline assembly for interrupt vector
132 }
133 #pragma code              // normal code area from here down
134 /*****************************************************************************/
135 //*********************************************************************
136 //* ISR: Blink                                                        *
137 //*                                                                   *
138 //* Inputs: none                                                      *
139 //* Outputs: none                                                     *
140 //*                                                                   *
141 //* The purpose of this ISR is to set the Blink-M color/brightness,*
142 //* read the TC-74 temp sensor, and store high/Low recorded        *
143 //* temperatures in EEProm every 100mS                                *
144 //*                                                                   *
145 //* Created by: Brandon Empie                                         *
146 //*********************************************************************
147 #pragma interruptlow Blink
148 void Blink(void)
149 {
150     if(INTCONbits.TMR0IF)   // check to see if TMR0 caused the interrupt
151     {
152
153         BlinkM();  //call BlinkM
154
155         TempRead();//call TempRead
156         if(temperature < 0) //if current temp is negative
157             {
158                 TStatus = 1;    //TStatus = 1
159                 TT = ((temperature)/-10);       //parsing low temperature value by Tens place
160                 TO = (((-temperature)-(TT*10))); //parsing low temperature value by Ones place
161             }
162         else if(temperature >= 0) //if current temp is positive
163             {
164                 TStatus = 0;               //TStatus = 0
165                 TT = ((temperature)/10);       //parsing low temperature value by Tens place
```

```
                TO = (((temperature)-(TT*10))); //parsing low temperature value by Ones place
            }

        if(clear == 0) //if S4 has been pressed write current temp into high and low address location
        {
            addr = ADDRTHigh;
            write();//write current temp into high address location
            addr = ADDRTLow;
            write();//write current temp into low address location
            clear = 1; //reset clear variable
        }
        addr = ADDRTHigh; //read from high temp. address
        read();//call read
        if(storage == 0xFF) //if read 0xFF write over whats in the location with current temp
        {
            write();//call write
            read();//call read
        }
        if(storage != 0xFF)//if storage is not 0xFF
        {
            if(storage >= temperature)//if read temp is higher than current update High temp
            {
                THigh = storage; //Thigh = storage
            }
            else if(storage < temperature)//otherwise if read temp is lower than current write it to EEProm
            {
                write(); //write current temp into High address location
                THigh = temperature;//Thigh = temperature
            }
            if(THigh < 0) //if high temp is negative
            {
                HStatus = 1;              //Hstatus = 1
                HT = ((THigh)/-10);       //parsing low temperature value by Tens place
                HO = (((-THigh)-(HT*10))); //parsing low temperature value by Ones place
            }
            else if(THigh >= 0) //otherwise if high temp is positive or zero
            {
                HStatus = 0; //Hstatus = 0
                HT = ((THigh)/10);       //parsing low temperature value by Tens place
                HO = (((THigh)-(HT*10))); //parsing low temperature value by Ones place
            }
        }
        addr = ADDRTLow; //read low temp from EEprom
        read();//call read
        if(storage == 0xFF)//if EEprom stored temperature is 0xFF, write over it and re read
        {
            write();//call write
            read();//call read
        }
        if(storage != 0xFF) //if low temp in EEprom is not 0xFF
        {
            if(storage <= temperature)//is low EEprom temp. lower than or equal to current temp
            {
                TLow = storage;//send EEprom stored low temp to TLow, to be sent to LCD later
            }
            else if(storage > temperature)//otherwise is low temp in EEprom greater than current
            {
                write();//call write
                TLow = temperature;//TLow = temperature
            }
            if(TLow < 0)//if current low temp is negative
            {
                LStatus = 1;//Lstatus = 1
                LT = ((TLow)/-10);        //parsing low temperature value by Tens place
                LO = (((-TLow)-(LT*10))); //parsing low temperature value by Ones place
            }
            else if(TLow >= 0)//otherwise is current low temp 0 or positive
            {
                LStatus = 0;             //LStatus = 0
                LT = ((TLow)/10);        //parsing low temperature value by Tens place
                LO = (((TLow)-(LT*10))); //parsing low temperature value by Ones place
            }

        }
        TMR0H = 0xFC;           //Reset Timer 0 to restart count
        TMR0L = 0xF3;           //64755 based on .5uS * 256 * count = 100mS (65536-781 = 64755)
        INTCONbits.TMR0IF = 0;  // clear TMR0 Flag for next go-round
    }
}
////////////////////////////////////////////////////////////////
//                          MAIN                              //
////////////////////////////////////////////////////////////////
void main (void)
```

```
{
    OSCCON = 0x60;        //set clock to default 8mHz
    TRISB = 0;            // PORT B all outputs for LEDs
    LATB = 0;             // All outputs off initially
    TRISA = 0xFF;         //PORT A all inputs
    ANCON0 = 0;           // AN0, AN1, AN2, AN3 configured as digital
    ANCON1 = 0;           // All other channels are digital

    PORTAbits.RA0 = 1;    //POT channel set to analog
    ADCON2bits.ADFM = 0;  //Left Justify A/D
    ADCON2bits.ADCS = 4;  //FOSC/4
    ADCON2bits.ACQT = 2;  //A/D acquisition time select 4 TAD
    ADCON0bits.CHS = 0;   //A/D init. to channel 00 (AN0) for potentiometer
    ADCON0bits.ADON = 1;  //Turn on A/D

    T0CONbits.T0PS = 7;     //Timer 0 prescaler set to 256
    T0CONbits.PSA = 0;      //Timer0 prescaler assigned, Clock input comes from prescaler output
    T0CONbits.T0CS = 0;     //Internal instruction cycle clock (CLK0)
    T0CONbits.T08BIT = 0;   //timer 0 configured as a 16-bit counter
    INTCON2bits.TMR0IP = 0; //Timer0 overflow interrupt priority bit 0 = low
    INTCONbits.TMR0IE = 1;  //enables TMR0 overflow interrupt bit
    INTCONbits.TMR0IF = 0;  // clear TMR0 Flag for next go-round

    TMR0H = 0xFC;           //Set Timer 0 high and low byte
    TMR0L = 0xF3;           //0xFCF3 (64755) based on .5uS * 256 * count = 100mS (65536-781 = 64755)

    INTCONbits.GIEH = 1;    // enables high-priority interrupts
    RCONbits.IPEN = 1;      // enable priority interrupts (both low and hi)
    INTCONbits.GIEL = 1;    //enables all interrupts that have priority bit cleared (low priority))

    OpenI2C(MASTER, SLEW_OFF);
    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))

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

    LCDLine_1();    // Setup 1st line
    for (j=0;j<7;j++)
    {
            temp_wr = line1[j];// write one char. at a time
            d_write();         // Send to LCD
    }
    end_line();     // end of line call

    T0CONbits.TMR0ON = 1;//Enables Timer 0 on right before main loop

    while(1)
    {
        switches();         //call 'switches' function to see if S2 or S3 is pressed
        Showoff();          //call 'Showoff' to set LCD

    }
}

//****************************************
//* Subroutine: switches                 *
//*                                      *
//* Inputs: none                         *
//* Outputs: none                        *
//*                                      *
//* The purpose of this subroutine is to  *
//* check the switches and adjust their   *
//* array pointer as necessary           *
//*                                      *
//* Created by: Brandon Empie            *
//****************************************
void switches(void)
{
    if(PORTAbits.RA7 != 1)      //if S4 is pressed (0) clear max and min temp to current temp
        {
            clear = 0;          //clear = 0
        }
   if(PORTAbits.RA6 != 1)      //if S3 is pressed (0) cycle Temp Now(0), High(1), Low(2) repeating
        {
            Z = Z + 1;         //increment Z
            if(Z == 3)         //if Z is at max
                Z = 0;         //reset to current temp
        }
```

```
    if(PORTAbits.RA5 != 1)      //if S2 is pressed (0) cycle Red(0), Green(1), Blue repeating(2))
         {
             Y = Y + 1;         //Y = Y + 1
            // Delay1KTCYx(250);   //250mS delay
             if(Y == 3)         //is Y == 3?, if so set Y = 0
                 Y = 0;
         }
    return;                     //return to main
}

//***************************************
//* Subroutine: Showoff                 *
//*                                     *
//* Inputs: none                        *
//* Outputs: none                       *
//*                                     *
//* The purpose of this subroutine is to  *
//* Send all data to the LCD screen       *
//*                                     *
//* Created by: Brandon Empie           *
//***************************************
void Showoff(void)
{
   temp_wr = 0x87; //setup cursor 9 positions from left on 1st line of LCD
   i_write();      //setting cursor position

   switch(Y)
   {
       case 0:
           for(j=0;j<6;j++)
            {
            temp_wr = Red[j];// write one char. at a time
            d_write();         // Send to LCD
            }
            break;
       case 1:
           for(j=0;j<6;j++)
            {
            temp_wr = Green[j];// write one char. at a time
            d_write();         // Send to LCD
            }
            break;
       case 2:
           for(j=0;j<6;j++)
            {
            temp_wr = Blue[j];// write one char. at a time
            d_write();         // Send to LCD
            }
            break;
   }

   if(H == 0) //leave hundreds place blank if its zero
   {
       temp_wr = 0x20; //write nothing
       d_write();      //Send to LCD
   }
   else //otherwise write hundreds place
   {
       temp_wr = H + 48; //write Hundreds
       d_write();      //Send to LCD
   }
   if(H + T == 0) //if they are both zero write nothing to tens
   {
       temp_wr = 0x20; //write nothing
       d_write();      //Send to LCD
   }
   else                 //otherwise write both digits
   {
   temp_wr = T + 48; //write Tens
   d_write();      //Send to LCD
   }
   temp_wr = O + 48; //write Ones
   d_write();      //Send to LCD

   temp_wr = 0xC1; //setup cursor 3 positions from left on 2nd line of LCD
   i_write();      //setting cursor position
   temp_wr = 0x20; //write nothing
   d_write();      //Send to LCD

   temp_wr = 0xC2; //setup cursor 3 positions from left on 2nd line of LCD
   i_write();      //setting cursor position

   switch(Z)
```

```
415     {
416         case 0:                       //Temp Now
417             for(j=0;j<11;j++)
418              {
419              temp_wr = Current[j];// write one char. at a time
420              d_write();         // Send to LCD
421              }
422              if(TStatus == 1) //if current temp is negative
423              {
424              temp_wr = 0xCB; // set cursor 12 positions from left
425              i_write();      //setting cursor position
426              temp_wr = 0x2D; // write - sign
427              d_write();      // Send to LCD
428              }
429              else if(TStatus == 0) //otherwise if current temp is positive
430              {
431              temp_wr = 0xCB; //set cursor 12 positions from left
432              i_write();      //setting cursor position
433              temp_wr = 0x20; //write nothing
434              d_write();      //Send to LCD
435              }
436              temp_wr = 0xCC; //setup cursor 13 positions from left on 2nd line of LCD
437              i_write();      //setting cursor position
438              if(TT == 0)     //if current temp tens place is zero
439              {
440              temp_wr = 0x20; //write nothing
441              d_write();      //Send to LCD
442              }
443              else            //otherwise write Tens and ones place to LCD
444              {
445              temp_wr = TT + 48; //write temperature Tens
446              d_write();      //Send to LCD
447              }
448              temp_wr = TO + 48; //write temperature Ones
449              d_write();      //Send to LCD
450              break;
451         case 1:                      //Temp High
452              temp_wr = 0xC1; //setup cursor 2 positions from left on 2nd line of LCD
453              i_write();      //setting cursor position
454              for (j=0;j<11;j++)
455              {
456              temp_wr = High[j];// write one char. at a time
457              d_write();         // Send to LCD
458              }
459              if(HStatus == 1) //if High temp is negative
460              {
461              temp_wr = 0xCB; // set cursor 12 positions from left
462              i_write();      //setting cursor position
463              temp_wr = 0x2D; // write - sign
464              d_write();      // Send to LCD
465              }
466              else if(HStatus == 0) //otherwise if high temp is positve
467              {
468              temp_wr = 0xCB; //set cursor 12 positions from left
469              i_write();      //setting cursor position
470              temp_wr = 0x20; //write nothing
471              d_write();      //Send to LCD
472              }
473              if(HT == 0)     //if high temp tens place is zero
474              {
475              temp_wr = 0x20; //write nothing
476              d_write();      //Send to LCD
477              }
478              else            //otherwise write tens place
479              {
480              temp_wr = HT + 48; //write high temperature Tens
481              d_write();      //Send to LCD
482              }
483              temp_wr = HO + 48; //write high temperature Ones
484              d_write();      //Send to LCD
485              break;
486         case 2:                      //Temp Low
487             for(j=0;j<10;j++)
488              {
489              temp_wr = Low[j];// write one char. at a time
490              d_write();         // Send to LCD
491              }
492              if(LStatus == 1)    //if Low temp is negative
493              {
494              temp_wr = 0xCB; // set cursor 12 positions from left
495              i_write();      //setting cursor position
496              temp_wr = 0x2D; // write - sign
497              d_write();      // Send to LCD
```

```
            }
            else if(LStatus == 0)//if low temp is positive
            {
            temp_wr = 0xCB; //set cursor 12 positions from left
            i_write();      //setting cursor position
            temp_wr = 0x20; //write nothing
            d_write();      //Send to LCD
            }
            if(LT == 0)     //if low temp tens place is a zero
            {
            temp_wr = 0x20; //write nothing
            d_write();      //Send to LCD
            }
            else            //otherwise write tens and ones place for low temp
            {
            temp_wr = LT + 48; //write low temperature Tens
            d_write();      //Send to LCD
            }
            temp_wr = LO + 48; //write low temperature Ones
            d_write();      //Send to LCD
            break;
    }
    temp_wr = 0x43; //write "C"
    d_write();      //Send to LCD

    return;                     //return to main
}
//****************************************
//* Subroutine: read                     *
//*                                      *
//* Inputs: none                         *
//* Outputs: none                        *
//*                                      *
//* The purpose of this subroutine is to  *
//* read from the EEprom using I2C       *
//*                                      *
//* Created by: Brandon Empie            *
//****************************************
void read(void)
{
    //read below
    StartI2C();
    while(SSPCON2bits.SEN); //wait for start

    while(WriteI2C(0xA0)); // send EEPROM address

    while(WriteI2C(0x03));  //send EEPROM internal upper address
    while(WriteI2C(addr));  //send EEPROM internal lower address

    IdleI2C();//idle i2c
    RestartI2C();//restart I2C
    IdleI2C();//idle i2c

    while(WriteI2C(0xA1)); // send EEPROM address read

    storage = ReadI2C();
    NotAckI2C(); //acknowledge

    IdleI2C();//idle i2c
    StopI2C();//stop i2c

    return;
}
//****************************************
//* Subroutine: write                    *
//*                                      *
//* Inputs: none                         *
//* Outputs: none                        *
//*                                      *
//* The purpose of this subroutine is to  *
//* write from the EEprom using I2C      *
//*                                      *
//* Created by: Brandon Empie            *
//****************************************
void write(void)
{
    //write
    IdleI2C();//idle i2c
    StartI2C();//Start i2c
    while(SSPCON2bits.SEN); //wait for start

    while(WriteI2C(0xA0));  //EEPROM Address
    IdleI2C();//idle i2c
```

```
581
582         while(WriteI2C(0x03));  //send EEPROM internal upper address
583         IdleI2C();//idle i2c
584         while(WriteI2C(addr));  //send EEPROM internal lower address
585         IdleI2C();//idle i2c
586         while(WriteI2C(temperature));  //send data
587
588
589         IdleI2C();//idle i2c
590         StopI2C();//stop i2c
591         while(SSPCON2bits.PEN);//wait for stop condition to complete
592         while(EEAckPolling(0xA0));//verify write completes
593         return;
594     }
595     //****************************************
596     //* Subroutine: BlinkM                   *
597     //*                                      *
598     //* Inputs: none                         *
599     //* Outputs: none                        *
600     //*                                      *
601     //* The purpose of this subroutine is to  *
602     //* read the POT and set the color and    *
603     //* intensity of the Blink M using i2c    *
604     //*                                      *
605     //* Created by: Brandon Empie            *
606     //****************************************
607     void BlinkM(void)
608     {
609             ADCON0bits.GO = 1;          //Start A/D
610             while(ADCON0bits.GO);       //wait till A/D is done
611
612             X = ADRESH;                 //X = A/D result 8-bit
613
614             H = (X/100);                //parsing A/D value by Hundreds place
615             T = ((X-(H*100))/10);       //parsing A/D value by Tens place
616             O = (((X-(H*100))-(T*10))); //parsing A/D value by Ones place
617
618             if(Y == 0)  //Red
619             {
620                R = X;
621                G = 0x00;
622                B = 0x00;
623             }
624             if(Y == 1)  //Green
625             {
626                R = 0x00;
627                G = X;
628                B = 0x00;
629             }
630             if(Y == 2)  //Blue
631             {
632                R = 0x00;
633                G = 0x00;
634                B = X;
635             }
636         SSPCON2bits.SEN = 1;    // Start Condition Enable
637         while(SSPCON2bits.SEN); // Wait for start cond. to end
638
639         SSPBUF = 0x00;          // send BlinkM address
640         smdelay();
641         while(SSPSTATbits.R_W); // Wait till transmit is done
642         while(SSPCON2bits.ACKSTAT);    // Wait till acknowledged by slave
643
644         SSPBUF = 'o';           // send BlinkM command to stop script
645         smdelay();
646         while(SSPSTATbits.R_W); // Wait till transmit is done
647         while(SSPCON2bits.ACKSTAT);    // Wait till acknowledged by slave
648
649         SSPBUF = 'n';           // send BlinkM command to send RGB values below
650         smdelay();
651         while(SSPSTATbits.R_W); // Wait till transmit is done
652         while(SSPCON2bits.ACKSTAT);    // Wait till acknowledged by slave
653
654         SSPBUF = R;          // send BlinkM value for Red
655         smdelay();
656         while(SSPSTATbits.R_W); // Wait till transmit is done
657         while(SSPCON2bits.ACKSTAT);    // Wait till acknowledged by slave
658
659         SSPBUF = G;          // send BlinkM value for Green
660         smdelay();
661         while(SSPSTATbits.R_W); // Wait till transmit is done
662         while(SSPCON2bits.ACKSTAT);    // Wait till acknowledged by slave
663
```

```
664       SSPBUF = B;          // send BlinkM value for Blue
665       smdelay();
666       while(SSPSTATbits.R_W); // Wait till transmit is done
667       while(SSPCON2bits.ACKSTAT);    // Wait till acknowledged by slave
668
669       SSPCON2bits.PEN = 1;    // initiate STOP
670       while(SSPCON2bits.PEN); // Wait for STOP mode idle
671
672       return;
673   }
674   //****************************************
675   //* Subroutine: TempRead                 *
676   //*                                      *
677   //* Inputs: none                         *
678   //* Outputs: none                        *
679   //*                                      *
680   //* The purpose of this subroutine is to  *
681   //* read the TC-74 temp sensor and save   *
682   //* the value in temperature variable     *
683   //*                                      *
684   //* Created by: Brandon Empie            *
685   //****************************************
686   void TempRead(void)
687   {
688   while(ready == 1) //run loop until temp has been read
689       {
690       StartI2C();
691       while(SSPCON2bits.SEN); // wait for start
692
693       while(WriteI2C(0x9A));  // Send TC74 address (Write; LSB=0)
694                               // function returns a zero if success, i.e ack.
695                               // see C18 Lib PDF pp. 28
696   // the command is placed in the variable cmd below.
697   // 1 = read config, 0 = read temperature.
698
699       while(WriteI2C(cmd));   // Select TC74 Reg 1 (Config)
700                               // wait for ack from slave
701       IdleI2C();              // idle i2c
702       RestartI2C();           // Restart i2c
703       IdleI2C();              // idle i2c
704
705       while(WriteI2C(0x9B));  // Send TC74 address (Read; LSB=1)
706                               // wait for ack from slave
707
708       read_data = ReadI2C();  // Read register from TC74
709
710       NotAckI2C();            //not acknowlege
711
712       IdleI2C();              // idle i2c
713
714       StopI2C();              // stop i2c
715   // at this point, the variable read_data has the response from the TC74
716   // the next step would be to determine whether the TC74 is ready to
717   // read temperature, then go through the sequence again based on that
718   // response to either read the temp, or to read the config again?
719
720       if(cmd == 0)            // SET a breakpoint here & watch read_data
721       {
722               temperature = read_data;
723               ready = 0;
724       }
725       cmd = read_data & 0x40?0:1; //if ready for reading cmd updated with 0
726       }
727       ready = 1;
728
729       return;
730   }
731   //****************************************
732   //* SUB: smdelay                         *
733   //*                                      *
734   //* Inputs: none                         *
735   //* Outputs: none                        *
736   //*                                      *
737   //* The purpose of this subroutine is to  *
738   //* create a small delay                 *
739   //*                                      *
740   //* Created by: Brandon Empie            *
741   //****************************************
742   void smdelay (void) //small delay subroutine
743   {
744       return;
745   }
```