finalMain.c

```
#include <wiringPi.h>
#include <lcd.h>
#include <stdio.h>
#include <mcp23017.h>
#include "finalHeader.h"
#include "messaging.h"


//GLOBAL VARIABLES
int page; // variables to keep track of page and line
int line;
int lcdhdl;
int set;
int entry;
int hour;
int min;
extern unsigned char newTherm;
extern unsigned char newReg;
extern unsigned char addresses[126];
extern unsigned char devices[rows][columns];
extern int mcClock[3];
int devNumber;
int day;
char dayTime[2];
extern int hvacSetting;
extern int temps[126][5];
extern int failedCon[126][126];
extern int regFlow[126][126][2];
extern int hvacAuto;
extern int hvacStatus;
int mainHalt = 0; //used to halt the main program for alternate threads while they complete
critical tasks. (adding devices)
int threadGreenLight = 0; //gives permission to threads to coninue once the main reaches a safe
stopping point for mainHalt usage.

/*********************************************************************************************
PI_THREAD (myThread)
   Description: This thread is used to update the real time clock the the master control.
   A seperate thread was created for this function so that the real time clock can be
   updated accurately without any delay and time miscalculations that could be
   caused by other function delay or lag.
*********************************************************************************************/
PI_THREAD (myThread)
{
   while(1)
   {
      delay(20);
      updateTime();
   }
}

/*********************************************************************************************
PI_THREAD (displayThread)
   Description: This thread is used to update the lcd display on the master control.
   We used a seperate thread for this function to avoid any latency that could occur
   from user operation and navigation that could be caused by an other time consuming
   functions.
*********************************************************************************************/
PI_THREAD (displayThread)
{
   while(1)
   {
      delay(20);
      updateDisplay();
```

```
    }
}

int main(void)
{
  //the following assignments are for initializing variables to a know state.
   dayTime[0] = 'A';
   dayTime[1] = 'P';
   day = 0;
   set = 0;
   entry = 0;
   hour = 0;

   setups();//setup the wiringPi library so that we may have control over GPIO's
   //lcdClear(lcdhdl);
   while(digitalRead(pwrSwitch))// wait until the power switch is turned on to continue.
   {
      delay(500);
      printf("Turn on power Switch.\n");
   }

   bootSequence(); //This function initializes all the necessary arrays and performs an lcd test.
   initMC(&devNumber, addresses);//performs a system sync for all system devices.

   while(!digitalRead(syncSwitch))//waits until the sync switch is released from the initMC
function
   {
      delay(20);
   }
   page = 4;//we start at page 4 so that the user may input the system time of the system.
   line = 0;
   display(page, line, set);

   populateArrays(); //this function orginizes the addresses gathered form initMC()

   //start threads
   if(piThreadCreate(myThread))
   {
      printf("myThread did not start.\n");
   }
   if(piThreadCreate(displayThread))
   {
      printf("displayThread did not start.\n");
   }

   /* //===THE FOLLOWING ASSIGNMENTS ARE FOR TESTING PURPOSES.===
   hvacSetting = 0;  //auto: 0, cooling: 1, heating: 2, fan: 3
   temps[1][setTemp] = 76;
   temps[1][currTemp] = 78;
  temps[2][setTemp] = 70;
   temps[2][currTemp] = 72;

   int room1Set;
   int room1Cur;
   int room2Set;
   int room2Cur;

   printf("enter room1Set: ");
   scanf("%d",&room1Set);
   printf("enter room2Set: ");
   scanf("%d",&room2Set);
   temps[1][setTemp] = room1Set;
   temps[2][setTemp] = room2Set;
   */ //===END OF TEST ASSIGNMENTS===
```

```
   //MAIN LOOP BEGINS HERE!
   while(!digitalRead(pwrSwitch))
   {
      if(mainHalt)//if the user initiated an "add device", halt main until done.
      {
         printf("MAIN HALTED.\n");
         threadGreenLight = 1;//give the thread permission to continue.
         while(mainHalt)//wait until the thread releases the main halt.
         {
            delay(200);
         }
         threadGreenLight = 0;//once the thread has finished, continue main routines.
      }

      retrieveTemps();//retrieve the temeratures from the thermostat devices.
      delay(1500);
      /* //===THE FOLLOWING SEGMENT OF CODE IS FOR TESTING PURPOSES===
      //printf("enter room1Cur: ");
      //scanf("%d",&room1Cur);
      //temps[1][currTemp] = room1Cur;
      //printf("enter room2Cur: ");
      //scanf("%d",&room2Cur);
      //temps[2][currTemp] = room2Cur;
      */ //===END OF TEST CODE SEGMENT===

      calcDiff();//determine temperature difference and if the hvac needs to turn on.
      hvacControl();//turn on/off the hvac system if needed and calculate rates of change.
      setReg();//set the registers where they need to be.


     //break;
   }


   shutDownSequence();//sequency which turns off all LED's and clears the LCD.
   return 0;
}
```

Syncing.c

```
#include <wiringPi.h>
#include <lcd.h>
#include <stdio.h>
#include <stdlib.h>
#include <mcp23017.h>
#include "finalHeader.h"
#include "messaging.h"

#define ROW_COUNT  126
#define COLUMN_COUNT  11

int changeWarnCount; //count to determine if user should change hvac setting.
               //used in auto setting to determine if system should change hvacSetting
unsigned char devices[rows][columns];//This Array contains the organized addresses of all
devices.
extern int lcdhdl;//This variable contains the lcd handle returned from lcd initialization.
extern int page;//this variable contains the current page that is being displayed on the LCD.
extern int line;//this variable contains the current cursor position for menu navigation in the
lcd.
extern int set;//this variable contains the current item being edited by the user in set time.
unsigned char addresses[126];//This array contains the addresses returned from initMC();
extern int rtTimes[126][2];//This array contains the start times of the timers used.
int conStatus = 0;
int failedCon[126][126];   //number of times failed to connect with devices
                     //[0][0] == connections errors
                     //[0][1] == incorrect hvac setting warning
int temps[126][5];   //[Current temp | Set temp | Current Humidity | temp difference | startDiff]
               //temps[0][3]: 1 if hvac needs to turn on. 0 if temps are at desired.
               //use temps[0][hvacOut] "hvacOut" is the defined constant for this.

int regFlow[126][126][2];//first two dimensions orginize the data the same way the
                  //registers address is stored. in the third dimension
                  //the 0 index will store the sending airflow, while the
                  //1 index will store the receiving current airflow (in pressure).


float rateOfChange[ROW_COUNT][COLUMN_COUNT];//this array contains the rates of change for each
therm.

int hvacSetting = 0; //user setting.
                  //not set: 0 (auto)
                  //cooling: 1
                  //heating: 2
                  //Fan only:3
int hvacAuto; //variable for hvacSetting = auto (3). 0 for cooling, 1 for heating.
int hvacStatus; //current hvac status. 0 for off, 1 for on.

//unsigned char thermoID = 0x80;

/*****************************************************************************************
void initArray(void)
   This function initializes the 1 dimensional and 2 dimensional arrays which are used
   to store device addresses. The purpose of the 1 dimensional array is to provide a
   data structure for the protocol function to store the acquired addresses. Moreover,
   the 2 dimensional array is used to organize the 1 dimensional array addresses into
   a format that is easier to access and work with. The method in which the arrays will
   be organized can be seen below in the comment header for function populateArrays().
*****************************************************************************************/
void initArray(void)
{
   int i;
   int j;
```

```
   //int p;

   //using max number of devices as conditions.
   int r = rows;
   int c = columns;

   for(i = 0; i < r; i++)//row loop.
   {
      for(j = 0; j < c; j++)//column loop.
      {
         devices[i][j] = 0x00;
         //p = devices[i][j];
         //printf("Row: %d, Column: %d, Content: %d\n", i, j, p); //for testing.
      }
   }

   //filling in 1 dim. array with zero's
   for(i = 0; i<126; i++)
   {
      addresses[i] = 0x00;
   }
   /* //===THE FOLLOWING ASSIGNMENTS ARE USED FOR TESTING PURPOSES ===
   //test addresses
   addresses[0]=0x81;
   addresses[1]=0x7E;

   addresses[2]=0x7D;

   addresses[3]=0x82;
   addresses[4]=0x7C;
   addresses[5]=0x7B;
   */ //===END OF TEST ASSIGNMENTS ===

   //for debugging, printing the values in in the 1 dim. array.
   for(i = 0; i<10; i++)
   {
      printf("addresses[%d]: %#.2x\n\n", i, addresses[i]);
   }
   return;
}


/*****************************************************************************************
void poulateArrays(void)
   Description:
   This funcion organizes the addresses stored in the 1 dimensional array
   addresses[], which has been populated by the initMC() protocol function, into a
   2 dimensional array called devices[][]. This 2 dimensional array is orginized in a
   specific manner to allow for easy access and array address information to be acquired
   by the MC. The method for organization is as follows:

   index [0][0] contains the total number of thermostats contained within the array.
   this number changes whenever the user adds or removes thermostats from the MC.
   every subsequent column within row 0 contains the number of registers per thermostat
   where the column number represents the thermostat number (or room #).

   Beginning from index [1][0], every row index within column 0 contains a thermostat
   address. For example, index [1][0] = thermostat 1, index [2][0] = thermostat 2,
   etc. We will indirectly label each thermostat its own room number for later reference
   (thermostat 1 is room 1, thermostat 2 is room 2, etc).

   Finally every column index within a row, beginning with [1][1], will contain a
   register address which is associated with that rows thermostat.

   Below, an example is shown on how the 1 and 2 dimensional arrays are organized.
```

```
   1 dim: [Therm A | Reg A1 | Reg A2 | Reg An | Therm B | Reg B1 | Reg Bn | Therm X]
         .. and so on. where n is any number of registers or thermostats below 128

   2 dim:   [Therm t | #RegA | #RegB | #RegN]
         [Therm A | RegA1 | RegA2 | RegAN]
         [Therm B | RegB1 | RegB2 | RegBN]
         [Therm X | RegXN | RegXN | RegXN]
*******************************************************************************************/
void populateArrays(void)
{
   int i;
   int tempCount=0;  //Start the temp count at 0
   int registerCount=0;    //Start the registerCount at 1


   for(i=0; i<126; i++)
   {

      if(!(addresses[i]&thermoID) && addresses[i] != 0)  //If the address is less than 0x80 it's
a register 1000 0000
      {
         registerCount++;                                  //Incriment Register Count
         //printf("registerCount = %d\n", registerCount);
         devices[tempCount][registerCount]=addresses[i]; //Add the device to the temp row on the
Second Column
         devices[0][tempCount]=registerCount;         //Place the registerCount in the first row
but equivanent column index of the Temp Row
      }
      else if((addresses[i]&thermoID) && addresses[i] != 0)             //If the address is
greater than 0x80 it's a Tempurature Sensor
      {
         registerCount=0;                        //Set RegisterCount back to zero
         tempCount++;                            //increment tempCount
         //printf("tempCount: %d\n", tempCount);
         devices[tempCount][registerCount]=addresses[i]; //Place temp address into Column 0 and
corresponding row

         devices[0][0]=tempCount;                  //Place total Tempurature count in index
[0][0]
      }
   }

   //display the matrix to console.
   dispMatrix();
   return;
}

/*******************************************************************************************
int addTherm(void)
   Description: This function adds a thermostat address to the two dimensional array
   devices[][]. The function first calls the protocol function addDevice() to acquire
   connection with the new device and generate an address, then stores that address
   into a variable. It then incriments the thermostat count in index [0][0] and moves
   the address into the index slot after the the last thermostat address in column 0.
   This function returns a (1) if connection was established and an address was created
   successfully by the protocol function: addDevice(). If the process is unsuccessful,
   the function returns a 0.
*******************************************************************************************/
int addTherm(void)
{
   unsigned char devAddr;  //declarin variable where the address will be store.
   if(addDevice(&devAddr, 1)) //calling protocol function to add a new device.
   {
      //add device successfull.
      int r = devices[0][0];  //increment the thermostat count in the 2d array.
      r++;
```

```
      devices[0][0] = r;
      devices[r][0] = devAddr;   //store the new address in the next availble location
      dispMatrix();  //display the 2d array with new address.
      return 1;
   }
   else
   {
     //add device unsuccessful.
      printf("\nadd Thermo function unsuccessful.\n");
      return 0;
   }
}



/*****************************************************************************************
void addReg(unsigned char addToRoom)
   Description: This function adds a new register address to the two dimensional array
   devices[][]. This function stores the address in its corresponding thermostat row
   which is predifined by the user through menu selection.
*****************************************************************************************/
int addReg(int addToRoom)
{
   unsigned char devAddr;  //declaring the variable that will store new address.
   if(addDevice(&devAddr, 2)) //call the protocol function to add new device.
   {
      //add device successfull
      //the next three lines increments the register count to the desired room.
      int c = devices[0][addToRoom];
      c++;
      devices[0][addToRoom] = c;

      devices[addToRoom][c] = devAddr; //stores the new address in an availble slot.
      dispMatrix();  //display the 2d array with the new address.
      return 1;
   }
   else
   {
      //add device unsuccessful
      printf("\nadd Register function unsuccessful.\n");
      return 0;
   }
   //dispMatrix();
}

/*****************************************************************************************
void dispMatrix(void)
   Description: This function is used for developer feedback and debugging purposes.
   The function calculates a properly sized matrix of the array devices[][]. It only
   displays a sized matrix that will fit all values that are non zero in the array.
*****************************************************************************************/
void dispMatrix(void)
{

   int i;
   int j;

   //the following for loop looks for the largest column to be displayed.
   int check=0;
   int largestC = 0;
   for(i = 1; i <= devices[0][0]; i++)
   {
      check = devices[0][i];
      if(check > largestC)
      {
         largestC = check;
```

```
      }
   }
   largestC++;
   //the following prints the array with adjustable size depending on how many
   //devices are connected.
   if(devices[0][0]) //make sure there is at least 1 reg, ad 1 therm synced
   {
      for(i = 0; i <= devices[0][0]; i++)
      {
         for(j = 0; j<=largestC; j++)
         {
            if(j == 0)
            {
               printf("[");//the start of a new row
            }

            if(i == 0)
            {
               printf("   %d ", devices[i][j]); //variables in row 0 will be displayed as
integers.
            }
            else
            {
               printf("%#.2x ", devices[i][j]); //variables in all rows != 0 will be displayed as
hex values.
            }

            if(j == largestC)
            {
               printf("]\n"); //end of a row.
            }

         }

      }
   }
   else  //if there isnt a minimum of 1 therm, and 1 reg, then dont display.
   {
      printf("Matrix is Empty. No Addresses stored.\n");
   }


   return;
}

/*****************************************************************************************
void rmDevAddr(unsigned char address)
   Description: This function is used to remove an address from the 2 dimensional array
   devices[][]. This function is called when the user selects the option from the LCD
   menu. The function requires an input unsigned char value of the address. The function
   then determines whether that address is a thermostat or register then removes it
   and readjusts the array so that the addresses are shifted accordingly in place of
   the deleted address. If the address removed is for a register, the function shifts
   all other registers in that row after the column of the address to be removed to the
   left. If the address is for a thermostat, the whole row including all registers
   associated with that thermostat are removed and the last occupied row in the array
   is moved into its place.
*****************************************************************************************/
void rmDevAddr(unsigned char address)
{
   int i;
   int j;
   int x;
   int acc;
   int locR;   //the row location of the address.
   int locC;   //the column location of the address.
```

```
int devType; //1 for therm, 0 for reg.


/*
int check;
int largestC = 0;
//determine the largest column in the 2d array.
for(i = 1; i <= devices[0][0]; i++)
{
   check = devices[0][i];
   if(check > largestC)
   {
     largestC = check;
   }
}
*/

//find the location of the device.
for(i=0;i<=devices[0][0];i++)
{
   for(j=0;j<=devices[0][i];j++)
   {
      if(devices[i][j] == address)
      {
         locR = i;
         locC = j;
         printf("address location: [%d][%d]\n",i,j);

      }
   }
}

//record what type of device this is.
if(address & thermoID)
{
   devType = 1;

}
else
{
   devType = 0;
   //acc = devices[0][locR];
   //devices[0][locR] = acc - 1;
}

if(devType) //if devType = 1 = therm. if devType = 0 = reg.
{
   int lastTherm = devices[0][0];//the last therm = total number of therms.
   if(locR == devices[0][0])
   {
      for(j = locC; j <= devices[0][locR]; j++)
      {
         //if removing the last therm. remove all addresses in that row.
         devices[locR][j] = 0x00;
         failedCon[locR][j] = 0;

      }
      rtTimes[locR][thermTimer] = 0;
      for(x = 0; x < COLUMN_COUNT; x++)
      {
         rateOfChange[locR][x] = 0;
         if(x < 4)
         {
            temps[locR][x] = 0;
         }
      }
```

```
    }
    else
    {
       int y;
       if(devices[0][lastTherm]>devices[0][locR])
       {
          y = devices[0][lastTherm];
       }
       else
       {
          y = devices[0][locR];
       }
       for(j = locC; j<= y; j++)
       {
          //if not removing the last therm, copy the last therm into this removed
          //addresses row and replace the last row with 0's
          //do this for all arrays containing data for that thermostat.
          devices[locR][j] = devices[lastTherm][j];
          devices[lastTherm][j] = 0x00;

          failedCon[locR][j] = failedCon[lastTherm][j];
          failedCon[lastTherm][j] = 0;

          regFlow[locR][j][sendingAF] = regFlow[lastTherm][j][sendingAF];
          regFlow[lastTherm][j][sendingAF] = 0;

          regFlow[locR][j][receivingAF] = regFlow[lastTherm][j][receivingAF];
          regFlow[lastTherm][j][receivingAF] = 0;
       }
       rtTimes[locR][thermTimer] = rtTimes[lastTherm][thermTimer];
       rtTimes[lastTherm][thermTimer] = 0;
       for(x = 0; x<COLUMN_COUNT; x++)
       {
          rateOfChange[locR][x] = rateOfChange[lastTherm][x];
          rateOfChange[lastTherm][x] = 0;
          if(x < 4)
          {
            temps[locR][x] = temps[lastTherm][x];
             temps[lastTherm][x] = 0;
          }
       }
    }
    //decriment the therm count.
    acc = devices[0][0];
    devices[0][0] = acc - 1;
    //shift the column values in row 0.
    devices[0][locR] = devices[0][lastTherm];
    devices[0][lastTherm] = 0;
}
else
{
    //if the address is a reg. remove the reg. and shift all reg in that row
    //(after the removed reg), to the left.
    for(j = locC; j <= devices[0][locR]; j++)
    {
       devices[locR][j] = devices[locR][j+1];

       failedCon[locR][j] = failedCon[locR][j+1];

      regFlow[locR][j][sendingAF] = regFlow[locR][j+1][sendingAF];

       regFlow[locR][j][receivingAF] = regFlow[locR][j+1][receivingAF];
    }
    //decriment the register count for that row.
    acc = devices[0][locR];
    devices[0][locR] = acc - 1;
```

```
   }


   dispMatrix();
   return;
}

/*******************************************************************************************
void initTempsArr(void)
   description: This function initializes the temps[][] array with predefined values.
   These values can change for testnig purposes however, the default value will be 0.
*******************************************************************************************/
void initTempsArr(void)
{
   int i;
   int j;

   //fill the array with 0's
   for(i = 0; i<126; i++)
   {
      for(j = 0; j < 5; j++)
      {
         temps[i][j] = 0;
         //failedCon[i][j] = 0;
      }
   }
   //printf("done initTempsArr, temps[1][tempDiff]: %d\n",temps[1][tempDiff]);
   initFailedCon();
   return;
}

/*******************************************************************************************
void initFailedCon(void)
   description: This function initializes the failedCon array to contain 0's.
*******************************************************************************************/
void initFailedCon(void)
{
   int i;
   int j;

   for(i = 0; i < 126; i++)
   {
      for(j = 0; j < 126; j++)
      {
         failedCon[i][j] = 0;
      }
   }
   return;
}
/*******************************************************************************************
void retrieveTemps(void)
   Description: This function uses protocol functions to send a request to the
   thermostats to retrieve their current and desired temperatures. It then waits
   for their reply and stores the data into the temps[][] array. If there is an error
   when attempting to communicate a counter is incremented in an array where the row
   indexes correspond to the thermostat number.
*******************************************************************************************/
void retrieveTemps(void)
{
   //printf("in retriveTemps();\n");
   int data1 = 0;
   int data2 = 0;
   int i;
   int j;
   //int rooms = devices[0][0];
   unsigned char toAddr;
```

```
   unsigned char msgCommand;
   unsigned char source;
   int attempt = 0;
   //int retData1;
   //int retData2;

   for(i = 1; i <= devices[0][0]; i++)//only do this loop as many times as there are synced
therms
   {

      data1 = 0;
      data2 = 0;
      //retrieve the address to send to which is stored in the 2d array.
      toAddr = devices[i][0];
      //use protocol function to send data retrieval request.
      if((attempt = sendMessage(GET_TEMPS, toAddr, data1, data2)))
      {
         failedCon[i][0] = 0;
         j=0;
         //wait for their message, an iteration timeout is included here.
         while(!(attempt = getMessage(&msgCommand, &source, &data1, &data2, typeMC)))
         {
            printf("waiting for therm reply(Temps).\n");


            if(j >= 10) //if 20 consecutive connection attemps fail.
            {
               break;
            }
            delay(10);
            j++;

            //if(smartDelay(20))
            //{
            // return;
               //updateDisplay();
            //}
         }

      }
      if((msgCommand == RETURN_TEMPS && attempt && data1 > -150 && data1 < 150 && data2 > -150 &&
data2 < 150))
      {//store data for temps in temporary variables since GET_HUM command will
      //overide the data.
         printf("\nSuccessfull connection to therm: %d\n", i);
         temps[i][retCurrTemp] = data1;
         temps[i][retSetTemp] = data2;
      }
      else
      {//-999 will be an indicator that the values are out of range (impractical) or
      //no conneciton was established so no meaningful data was gathered.
         if(!attempt)
         {
            printf("\nUnsuccessfull communication with therm: %d\n\n", i);
         }
         temps[i][retCurrTemp] = -999;
         temps[i][retSetTemp] = -999;
      }
      msgCommand = 0x00;
      source = 0x00;

      data1 = 0;
      data2 = 0;
     /*
      delay(500);
      //now retrieving humidity.
```

```
   if((attempt = sendMessage(GET_HUM, toAddr, data1, data2)))
   {
      j=0;
      //wait for reply from thermostat.
      while(!(attempt = getMessage(&msgCommand, &source, &data1, &data2, typeMC)))
      {
         //printf("waiting for therm reply (Humidity).\n");


         if(j >= 10)
         {
            break;
         }
         delay(10);
         j++;
         //if(smartDelay(20))
         //{
         // return;
            //updateDisplay();
         //}
      }
         //store the data that was retrieved.
   }
   */
   if(attempt)
   {

      //store retrieved data.

      //temps[i][2] = data1;

      printf("retCurrTemp: %d\n", temps[i][retCurrTemp]);
      printf("retSetTemp: %d\n", temps[i][retSetTemp]);
      printf("retHum: %d\n\n", temps[i][retHum]);

      failedCon[i][0] = 0;

   }
   else if(failedCon[i][0] != 1)
   {
      printf("\nUnsuccessfull communication with therm: %d\n\n", i);
      //we decriment this 10 times before we consider it an error
      if(failedCon[i][0] <= 0 && (failedCon[i][0] > -5))
      {
         failedCon[i][0]--;

      }
      else if(failedCon[i][0] <= -5)
      {
         failedCon[i][0] = 1;
         printf("\nfailedCon: %d in i=%d\n", failedCon[i][0], i);
         printf("error: unable to re-establish connection with therm: %d\n\n", i);
      }
      printf("retCurrTemp: %d\n", temps[i][retCurrTemp]);
      printf("retSetTemp: %d\n", temps[i][retSetTemp]);
      printf("retHum: %d\n\n", temps[i][retHum]);
   }
   //printf("failedCon for therm: %d, value: %d",i,failedCon[i][0]);
   delay(500);
}
if(page == 0 && ((failedCon[0][0] == 1) || failedCon[0][1]))
{
   conStatus = 1;
   display(page, line, set);
}
else
```

```
   {
      updateStatus();
   }

}

/*******************************************************************************************
void initRegFlow(void)
   Description: This function initializes the regFlow[] array with predefined values.
   The values should be 0 but can be changed for testing purposes.
*******************************************************************************************/
void initRegFlow(void)
{
   int i;
   int j;
   int z;
   //fill array with 0's
   for(i = 0; i<126; i++)
   {
      for(j = 0; j<126; j++)
      {
         for(z = 0; z < 2; z++)
         {
            regFlow[i][j][z] = 0;
         }

      }

   }

   //The following assignments are for testing purposes.
   //comment these out for real assignments.
   //regFlow[1] = 80;
   //regFlow[2] = 50;

   return;
}

/*******************************************************************************************
void setReg(void)
   Description: This function uses the "sendMessage()" protocol function to send data
   to the synced registers. The values being sent are just a percentage value that
   ranges from 0 to 100.
*******************************************************************************************/
void setReg(void)
{
   //printf("Enter setReg();\n");
   int data1;
   int data2;
   int i;
   int j;
   unsigned char toAddr;
   unsigned char msgCommand;
   unsigned char source;
   int attempt;
   int k;

   //go through each row where there are thermostats
   for(i = 1; i<=devices[0][0]; i++)
   {
      //go through each column (register address) in the current row.
      for(j = 1; j<=devices[0][i]; j++)
      {
         //get the address we're sending to.
         toAddr = devices[i][j];
         printf("sending to register: %#.2x \n",toAddr);
```

```
         //use protocol function to send the GET_FLOW request.
         /*
         if((attempt = sendMessage(GET_FLOW, toAddr, data1, data2)))
         {
            k=0;
           //wait for a reply from the reg. iteration timeout included.
            while(!(attempt = getMessage(&msgCommand, &source, &data1, &data2, typeMC)))
            {
               //printf("waiting for reg reply (Pressure).\n");
               k++;

               if(k >= 10)
               {
                  break;
               }
               if(smartDelay(20))
               {
                  updateDisplay();
               }
            }


         }

         if(attempt)
         {
            //store the retrieved data.
            regFlow[i][j][receivingAF] = data1;
            //printf("Pressure recieved from reg: %d\n",regFlow[i][j][receivingAF]);
            //printf("Successful. sent: data1: %d data2: %d, to Address: %#.2x \n", data1, data2,
toAddr);
         */
         //printf("IJregFlow[%d][%d][sendingAF]: %d\n",i,j,regFlow[i][j][sendingAF]);
         data1 = (regFlow[i][j][sendingAF])*10;
         data2 = data1;
         //delay(500);
         printf("DATA SENT TO REGISTER[%d][%d]: %d\n",i,j,data1);

         if((attempt = sendMessage(SET_FLOW, toAddr, data1, data2)))
         {
            failedCon[i][j] = 0;
            //printf("SET_FLOW send successfull. sent D1: %d D2: %d to Address: %#.2x \n",data1,
data2, toAddr);

         }

         if(failedCon[i][j] != 1 && !attempt)
         {
           printf("\nUnsuccessfull communication with reg: %d in room: %d\n\n", j,i);
            //we decriment this 10 times before we consider it an error
            if(failedCon[i][j] <= 0 && (failedCon[i][j] > -5))
            {
               failedCon[i][j]--;
            }
            else if(failedCon[i][j] <= -5)
            {
               failedCon[i][j] = 1;
               printf("failedCon: %d where i=%d,j=%d",failedCon[i][j],i,j);
               printf("error: unable to re-establish connection with Reg: %d in room: %d\n\n",j,
i);
            }
         }
         //printf("failedCon for reg: %d, room: %d, value: %d",i,j,failedCon[i][0]);
         delay(500);
      }
```

```
   }
   return;
}

/*******************************************************************************************
int countErrors(void)
   Description: This funciton reads through the stored temperatures retrieved from
   retrieveTemps() and counts all the errors from each thermostat.
*******************************************************************************************/
int countErrors(void)
{
   failedCon[0][0] = 0;
   //failedCon[0][1] = 0;
   int i;
   int j;
   for(i = 1; i <= devices[0][0]; i++)
   {
      for(j = 0; j <= devices[0][i]; j++)
      {
         if(failedCon[i][j] == 1)
         {
            printf("\nIn countErrors before increment\nfailedCon[%d][%d]:
%d\n",i,j,failedCon[i][j]);
            failedCon[0][0]++;//this array location contains the total number of connection
errors.
            printf("\nIn countErrors after increment\nfailedCon[%d][%d]:
%d\n",i,j,failedCon[i][j]);
         }
      }
   }
   int failed = failedCon[0][0];
   //int failedReg = failedCon[0][1];
   printf("\nFAILURES: %d\n",failed);
   //delay(2000);
   return failed;
}

/*******************************************************************************************
void initRocArray(void)
   This function initializes the 2
   dimensional array that is used to
   store rate of change values. The
   array is initially populated with
   all zeros.
*******************************************************************************************/
void initRocArray(void)
 {
   int row, column;

   for (row = 0;row < ROW_COUNT; row++)
   {
      for (column = 0;column < COLUMN_COUNT; column++)
      {
         if(row == 0)
         {
            rateOfChange[row][column] = column*10;//The first row will contain the percentages
being sent.
         }
         else
         {
            rateOfChange[row][column] = 0;// all other rows will contain 0's
         }

      }
   }
}
```

```
/*******************************************************************************
void calcDiff(void)
   Description: This function reads through the stored data in the temps[][] array
   and calculates the difference between the set temperature and the current temperature.
   it then stores this result into its own column in the same array. This function only
   calculates an absolute difference of the temperatures since the state of the hvac
   will be determined in the hvacControl() array.
*******************************************************************************/
void calcDiff(void)
{
   int i;
   //int settingCount; //counter to determine if hvac setting should be changed.
   //int diff;
   changeWarnCount = 0;
   failedCon[0][1] = 0;
   int heatingCount = 0;
   int coolingCount = 0;
   int diffTotal = 0;
   temps[0][hvacOut] = 0;
   printf("current hvac setting: %d\n",hvacSetting);


   for(i = 1; i <= devices[0][0]; i++)
   {
      //printf("temps[i][currTemp]: %d\ntemps[i][setTemp]:
%d\n",temps[i][currTemp],temps[i][setTemp]);
      if(temps[i][currTemp] != -999)
      {

         if(hvacSetting == 2)//heating
         {

            temps[i][tempDiff] = (temps[i][setTemp]) - temps[i][currTemp];
            printf("\ntemp Diff for heating: %d\n", temps[i][tempDiff]);
            if(temps[i][currTemp] < (temps[i][setTemp] - 1))
            {
               temps[0][hvacOut] = 1;  //hvac needs to turn on.
            }
            else if(temps[i][currTemp] > (temps[i][setTemp] + 1))
            {
               printf("incrementing changeWarnCount in heating.\n");
               changeWarnCount++;
            }
         }
         else if(hvacSetting == 1)//cooling
         {

            temps[i][tempDiff] = temps[i][currTemp] - temps[i][setTemp];
            //printf("in calcDiff(); in hvacSetting 1, currTemp: %d, setTemp: %d\n",
temps[i][currTemp], temps[i][setTemp]);
            printf("\ntemp Diff for cooling: %d\n", temps[i][tempDiff]);
            if(temps[i][currTemp] > (temps[i][setTemp] + 1))
            {
               temps[0][hvacOut] = 1;
           }
            else if(temps[i][currTemp] < (temps[i][setTemp] - 1))
            {
               changeWarnCount++;
            }
         }
         else if(hvacSetting == 0)//auto
         {
            temps[i][tempDiff] = temps[i][setTemp] - temps[i][currTemp];
            printf("in auto setting temps[%d][tempDiff]: %d\n",i,temps[i][tempDiff]);
            if(0 < temps[i][tempDiff])//if posative
```

```
            {//heating
               if(temps[i][tempDiff]>1)//if out of comfort range
               {
                  printf("in auto setting(heating) temps[%d][tempDiff]:
%d\n",i,temps[i][tempDiff]);
                  heatingCount++;
                  diffTotal = diffTotal + temps[i][tempDiff];
               }
            }
            else if(0 > temps[i][tempDiff])
            {//cooling
               if(temps[i][tempDiff]<-1)//if out of comfort range
               {
                  coolingCount++;
                  temps[i][tempDiff] = abs(temps[i][tempDiff]);
                  printf("in auto setting (cooling) temps[%d][tempDiff]:
%d\n",i,temps[i][tempDiff]);
                  diffTotal = diffTotal - temps[i][tempDiff];
               }

            }

         }
      }

   }
   //if most devices need to change setting (more than half)
   if(hvacSetting != 3 && changeWarnCount > (devices[0][0]/2))
   {
      failedCon[0][1] = 1;
   }
   //printf("heatingCount: %d\ncoolingCount: %d\n", heatingCount, coolingCount);
   int counts = heatingCount + coolingCount;
   if(hvacSetting == 0 && counts)
   {

      temps[0][hvacOut] = 1;
      //printf("In auto and counts > 0. temps[0][hvacOut]: %d\n",temps[0][hvacOut]);
      if(heatingCount > coolingCount)
      {
         hvacAuto = 1;
      }
      else if(coolingCount > heatingCount)
      {
         hvacAuto = 0;
      }
      else //heatingCount == coolingCount
      {
         if(diffTotal > 0)
         {
            hvacAuto = 1;
         }
         else if(diffTotal < 0)
         {
            hvacAuto = 0;
         }
         else
         {
         //diffTotal == 0; temperature difference totals are equal. (rare stalemate)
            hvacAuto = -1;//this will turn on the fan to cycle air out and hopefully change temps
out of stalemate
         }

     }
   }
```

```
   printf("failedCon[0][1]: %d\nchangWarnCount: %d\ntemps[0][hvacOut]: %d\nhvacAuto: %d\n",
failedCon[0][1],changeWarnCount,temps[0][hvacOut],hvacAuto);
   return;
}

/*****************************************************************************************
void hvacControl(void)
   Description: This function controls the state of the hvac system. it also handles
   timers for each room to determine the amount of time it will take for the system
  to reach the desired temperature for that room. This is necessary information for
   calculating rate of changes and to attempt to reach all desired temperatures at the
   same time.
*****************************************************************************************/
void hvacControl(void)
{
   int i;
   int j;
   float maxTime = 0;
   int maxTimeIndex;
   int flowIndex;
   int regPerc;
   //the following assignments are for testing purposes.
   /*
   //rateOfChange[1][10] = 1.6;
   //rateOfChange[2][6] = 1.2;
   //rateOfChange[2][7] = 1.3;
   //rateOfChange[2][8] = 1.6;
   //rateOfChange[2][9] = 1.7;
   //rateOfChange[2][10] = 1.8;

   rtTimes[1][thermTimer] = 46000;
   rtTimes[2][thermTimer] = 46100;
   regFlow[1][1][sendingAF] = 10;
   regFlow[2][1][sendingAF] = 10;
   temps[1][startDiff] = 2;
   temps[2][startDiff] = 2;
   hvacStatus = 1;//currently on or off?
   temps[0][hvacOut] = 0;//does a room need a temp change?
   */
   //the above assignments are for testin purposes.

   if(hvacStatus) //hvac is currently on
   {
      printf("HVAC IS CURRETNLY ON\n");
      if(1)//hvac should stay on <temps[0][hvacOut]> <- this was in if statement, replaced with 1
for testing.
      {
         int outOfRangeCount = 0;
         //printf("HVAC SHOULD STAY ON\n");
         for(i=1;i<=devices[0][0];i++)
         {
            printf("in for loop. room: %d, timer: %d, diff:
%d\n",i,rtTimes[i][thermTimer],temps[i][tempDiff]);
            if(rtTimes[i][thermTimer] && temps[i][tempDiff] <= 0)//check if temps were acquired
            {//if temps were acquired for certain rooms, calculate their rate of change for that
            //flow rate.
               //printf("in 2nd for loop.\n");
               flowIndex = regFlow[i][1][sendingAF];//what index was used to set the regFlow.
               if(rateOfChange[i][flowIndex] == 0.0 && flowIndex != 0)
               {//if there is no rate of change data, we will fill its row with theoretical
               //values from this real rate of change.
                  if(flowIndex == 10)
                  {
                     int endTime = rtTimerEnd(i,thermTimer);
                     printf("temps[%d][startDiff]: %d, endTime:
%d\n",i,temps[i][startDiff],endTime);
```

```
                        rateOfChange[i][flowIndex] =
(float)(temps[i][startDiff]*60)/(float)(endTime);
                        //printf("roc[%d][flowIndex]: %f\n",i,rateOfChange[i][flowIndex]);
                    }
                    else
                    {
                        int endTime = rtTimerEnd(i,thermTimer);
                        printf("endTime: %d\n",endTime);
                        rateOfChange[i][flowIndex] =
(float)(temps[i][startDiff]*60)/(float)(endTime);
                        float modifier = (float)(flowIndex)/(float)(10);
                        rateOfChange[i][10] = (rateOfChange[i][flowIndex])/modifier;
                    }
                    rtTimes[i][thermTimer] = 0;//reset the timer for this room.
                    printf("calculating theoretical RoC data:\n");
                    for(j = 1; j < COLUMN_COUNT; j++)
                    {
                        if(j != flowIndex && j != 10)
                        {
                            //printf("IM HERE!!!\n");
                            float modifier = (float)(j)/(float)(10);//get percentage to calculate
theoretical rates.
                            rateOfChange[i][j] = rateOfChange[i][flowIndex]*modifier;
                        }
                        printf("rateOfChange[%d][%d]: %f\n",i,j,rateOfChange[i][j]);
                    }
                }
                else//if there is rate of change data.
                {
                    rateOfChange[i][flowIndex] =
(float)(temps[i][startDiff]*60)/(float)(rtTimerEnd(i,thermTimer));
                    rtTimes[i][thermTimer] = 0;
                }
                adjustReg(i, 0);
                setReg();//close the registers for the rooms that have reached their desired temp.
            }
            else if(rtTimes[i][thermTimer] && temps[i][tempDiff] > 0)
            {//if temps have not been reached (this might happen if the temp diff is == 1)
                //we want to reach a temp difference = 0 to satisfy user expectations.
                outOfRangeCount++;
                printf("room: %d, still out of temp range\n", i);
            }
            else if(rtTimes[i][thermTimer] == 0 && temps[i][tempDiff] > 1)
            {
                rtTimerStart(i, thermTimer);
                adjustReg(i, 10);
                setReg();
            }
        }
        if(!outOfRangeCount)
        {//if all desired temperatures have been reached, turn off hvac system.
            printf("turning off system. \n");
            digitalWrite(COOL, 1);
            digitalWrite(FAN, 1);
            digitalWrite(HEAT, 1);
            hvacStatus = 0;
        }
    }
    //testing if below is needed (moving it above)
    /*
    else //hvac should turn off.
    {
        printf("HVAC SHOULD TURN OFF\n");
        int outOfRangeCount = 0;
      for(i=1;i<devices[0][0];i++)
        {
```

```
            //we will only turn off hvac once all temps have reached the desired temp
            if(rtTimes[i][thermTimer] && temps[i][tempDiff] <= 0)
            {
               flowIndex = regFlow[i][1][sendingAF];//what index was used to set the regFlow.
               rateOfChange[i][flowIndex] =
(float)(temps[i][startDiff]*60)/(float)(rtTimerEnd(i,thermTimer));

               adjustReg(i,0);//close registers for the rooms that have reached their desired
temps
               rtTimes[i][thermTimer] = 0;//reset the timer for this room.
            }
            else if(rtTimes[i][thermTimer] && temps[i][tempDiff] > 0)
            {//if temps have not been reached (this might happen if the temp diff is == 1)
               //we want to reach a temp difference = 0 to satisfy user expectations.
               outOfRangeCount++;
               printf("room: %d, still out of temp range\n", i);
            }

            if(!outOfRangeCount)
            {//if all desired temperatures have been reached, turn off hvac system.
               printf("turning off system. \n");
               digitalWrite(COOL, 1);
               digitalWrite(FAN, 1);
               digitalWrite(HEAT, 1);
            }
         }
      }
      */
   }
   else //hvac is currently off
   {
      printf("HVAC IS CURRENTLY OFF\n");
      if(temps[0][hvacOut])//hvac should turn on
      {
         printf("HVAC SHOULD TURN ON\n");
         int onTime;
         for(i = 1; i <= devices[0][0]; i++)//calc maxTime the system should stay on
         {
            if(temps[i][tempDiff] > 1 && rateOfChange[i][10])
            {
               onTime = (temps[i][tempDiff]*60)/rateOfChange[i][10];
               if(onTime>maxTime)
               {
                  maxTimeIndex = i;
                  maxTime = onTime;
               }
            }
         }//what happens if no therms have data? ================================
         printf("maxTime: %f\n",maxTime);
         if(hvacSetting == 1 || hvacSetting == 2 || hvacSetting == 0)//cooling, heating, auto
         {
            printf("hvac setting is 1, 2, or 0.\n");
            for(i =1; i <= devices[0][0]; i++)//set registers i=room #
            {
               if(temps[i][tempDiff]>1)//if this room needs a temp change
               {
                  printf("room: %d, needs a temp change.\n", i);
                  if(rateOfChange[i][10] == 0)
                  {//if no RoC data for that room
                     printf("room: %d, has no ROC data\n", i);
                     adjustReg(i,10); //adjust regesters for this room to 100 percent.
                     //start timer? ============================================
                  }
                  else//we have RoC data to use for this room.
                  {
                     printf("room: %d, has ROC data\n",i);
```

```
                    float timeLow;
                    float timeHigh;


                    for(j = 0; j < COLUMN_COUNT; j++)
                    {
                       timeLow = ((temps[i][tempDiff]*60)/rateOfChange[i][j]);
                      timeHigh = ((temps[i][tempDiff]*60)/rateOfChange[i][(j+1)]);
                       if(i == maxTimeIndex)
                       {
                          regPerc = 10;
                          break;
                       }
                       else if(timeLow >= maxTime && timeHigh <= maxTime)
                       {
                          printf("timeLow: %f\ntimeHigh: %f\nmaxTime: %f\n",timeLow, timeHigh,
maxTime);
                          if(timeLow == maxTime || timeHigh == maxTime)
                          {
                             if(timeLow == maxTime)
                             {
                                regPerc = j;
                             }
                             else
                             {
                                regPerc = j+1;
                             }
                          }
                          else if((abs(maxTime - timeLow)) < (abs(timeHigh - maxTime)))
                          {
                             printf("using low index: %d\n", j);
                             //use low index (j)
                             regPerc = j;
                             break;
                          }
                          else
                          {
                             printf("using high index: %d\n", (j+1));
                             //use high index (j+1)
                             regPerc = j+1;
                             if(regPerc > 10)
                             {
                                printf("regPerc out of range. = 10\n");
                                regPerc = 10;
                             }

                             break;
                          }

                       }
                    }
                    int tempval = regPerc;
                    printf("regesters in room: %d, using %d percent\n", i, tempval);

                    //regPerc = 10;


                    adjustReg(i,regPerc);
                 }
                 rtTimerStart(i,thermTimer);   //start timer for this room which needs a temp
change.
                 temps[i][startDiff] = temps[i][tempDiff];
                 printf("started timer for room %d, start time: %d\n", i,
rtTimes[i][thermTimer]);
              }
```

```
            }
            //regPerc = 10;
            adjustReg(i,regPerc);
            printf("setting Registers and turning on system\n");
            setReg();//physically set registers

            if(hvacSetting == 0)//auto
            {
               printf("hvacSetting on auto.\n");
               if(hvacAuto == 1)//heating
               {
                  printf("auto will use heating\n");
                  digitalWrite(COOL, 1);
                  digitalWrite(FAN, 0);
                  digitalWrite(HEAT, 0);
               }
               else if(hvacAuto == 0)//cooling
               {
                  printf("auto will use cooling\n");
                  digitalWrite(COOL, 0);
                  digitalWrite(FAN, 0);
                  digitalWrite(HEAT, 1);
               }
               else if(hvacAuto == -1)//fan
               {
                  printf("auto will use fan\n");
                  digitalWrite(COOL, 1);
                  digitalWrite(FAN, 0);
                  digitalWrite(HEAT, 1);
               }
            }
            else if(hvacSetting == 1)//cooling
            {
               printf("hvacSetting on cooling.\n");
              digitalWrite(COOL, 0);
               digitalWrite(FAN, 0);
               digitalWrite(HEAT, 1);
            }
            else if(hvacSetting == 2)//heating
            {
               printf("hvacSetting on heating.\n");
               digitalWrite(COOL, 1);
               digitalWrite(FAN, 0);
               digitalWrite(HEAT, 0);
            }

         }
         else if(hvacSetting == 3) //Fan only.
         {
            printf("hvacSetting on fan.\n");
            digitalWrite(COOL, 1);
            digitalWrite(FAN, 0);
            digitalWrite(HEAT, 1);
         }
         hvacStatus = 1;
      }
      else//hvac should stay off.
      {
         printf("HVAC SHOULD STAY OFF\n");
      }
   }

   return;
}

/*****************************************************************************************
```

```
void adjustReg(int i, int flowIndex)
   Description: this function adjusts all registers in a room to a specific flow rate.
*******************************************************************************************/
void adjustReg(int i, int flowIndex)
{
   int j;
   for(j = 0; j <= devices[0][i]; j++)
   {
      regFlow[i][j][sendingAF] = flowIndex;
   }
   return;
}

/*******************************************************************************************
void updateStatus(void)
   Description: This funciton checks the current status of errors and warnings and
   displays the status to the user on the lcd home screen.
*******************************************************************************************/
void updateStatus(void)
{
   if(conStatus)
   {
      conStatus = 0;
      display(page, line, set);
   }

   return;
}
```

Setups.c

```
#include <wiringPi.h>
#include <lcd.h>
#include <stdio.h>
#include <mcp23017.h>
#include <time.h>
#include "finalHeader.h"

//GLOBAL VARIABLES
extern int page; // variables to keep track of page and line
extern int line;
extern int lcdhdl;
extern int set;

/*********************************************************************************************
void setups(void)
   Description: This function initializes the gpio functionality using the wiringPi
   library as well as set all used pines to a known default state. The LCD is also
   initialized here.
*********************************************************************************************/
void setups(void)
{
   int i;
   wiringPiSetupGpio();
   mcp23017Setup (100, 0x20); //Gpio expander initialization.
   for (i=0;i<8;++i)    //loop to set pin modes for LCD. (on GPIO expander pins)
   {
      pinMode(100+i,OUTPUT);

   }
   //The following sets the gpio's to a default known state.
   pinMode(FAN, OUTPUT);
   pinMode(COOL, OUTPUT);
   pinMode(HEAT, OUTPUT);
   pinMode(pwrLED, OUTPUT);
   //pinMode(syncLED, OUTPUT);
   pinMode(entSwitch, INPUT);
   pinMode(dwnSwitch, INPUT);
   pinMode(upSwitch, INPUT);
  pinMode(syncSwitch, INPUT);
   pinMode(rstSwitch, INPUT);
   pinMode(pwrSwitch, INPUT);

   digitalWrite(FAN, HIGH);
   digitalWrite(COOL, HIGH);
   digitalWrite(HEAT, HIGH);
   digitalWrite(pwrLED, HIGH);
   //digitalWrite(syncLED, HIGH);
   digitalWrite(pwrLED, LOW);

   lcdinitialization();


   return;
}

/*********************************************************************************************
void bootSequence(void)
   Description: This function initializes all arrays used in this program to a known
   state of 0. As well as perform an LCD test.
*********************************************************************************************/
void bootSequence(void)
{
```

```
   //lcdBoot();
   initArray();
   initTimeArr();
   initRegFlow();
   initTempsArr();
   initRocArray();

   return;
}

/*****************************************************************************************
void shutDownSequence(void)
   Description: This function sets all led off and clears the LCD before the program
   terminates.
*****************************************************************************************/
void shutDownSequence(void)
{
   //turn off all LED's, relay signals, and clear lcd.
   lcdClear(lcdhdl);
   digitalWrite(pwrLED, HIGH);
   digitalWrite(COOL, HIGH);
  digitalWrite(HEAT, HIGH);
   digitalWrite(FAN, HIGH);
   return;
}
```

Time.c

```
#include <wiringPi.h>
#include <lcd.h>
#include <stdio.h>
#include <mcp23017.h>
#include <time.h>
#include "finalHeader.h"

int seconds;
int timeFlag = 0;
extern int page;

/*********************************************************************************************
int getSec(void)
   Description: This function returns the current seconds from the time struct created
   by time_t provided by <time.h>. the function first retrieves the epoch time, then
   formats it into a structure containing hours, minutes, seconds, etc. This function
   only retrieves the seconds.
*********************************************************************************************/
int getSec(void)
{
   time_t rawtime;
   struct tm *timestruct;

   time(&rawtime);   //retrieve epoch time.
   timestruct = gmtime(&rawtime);   //organize epoch time into a struct
   seconds = timestruct->tm_sec; //retrieve only the seconds of the struct.

   return seconds;
}

/*********************************************************************************************
void initTimeArr()
   Description: This function initializes the array mcClock[3] which is created in the
   global variables. This funciton will be used to keep track of real time.
   It fills the array with predefined values (most likely 0's but these values can be
   changed for testing pursposes.)
*********************************************************************************************/
int mcClock[4];   //array to store time. [hour(0) | minute(1) | seconds(2) | am/pm(3]

/*********************************************************************************************
void initTimeArr()
   Description: This function initializes the array, which will contain the values
   of the MC's real time clock, into a known state.
*********************************************************************************************/
void initTimeArr()
{
   int i;
   for(i=0;i<3;i++)
   {
      if(i==0)
         mcClock[i] = 12;
      else
         mcClock[i] = 59;
   }
   mcClock[arrAP] = 0;
   return;
}

/*********************************************************************************************
void updateTime(void)
   Description: This function provides a quick single iteration method of updating the
   real time clock. It retrieves and compares the seconds count from getSec() function
```

```
   with the value stored in the seconds index in the array mcClock[]. If this value
   is equal to 0, than the minutes need to be incrimented. However, it only does this if
   the value retrieved from getSec() is not equal to the stored seconds. This is done
   so that it only incraments the minutes once. once the minute count re
*******************************************************************************************/
void updateTime(void)
{
  if(page == 4)//if we are setting the time, do not adjust time values
   {
     return;
   }
   int currSec = getSec();//retrieve the current epoch seconds.
   if(currSec == 0 && currSec != mcClock[arrSec])
   {//if the seconds make a transition from 59 to 0 then the minute must be adjusted.

      if(mcClock[arrMin] == 59)
      {//if the minute makes a transition from 59 to 0, the hour must be adjusted
         timeFlag = 1;//This flag signals that the lcd needs to be updated to reflect the new
time.
         mcClock[arrMin] = 0;//reset minutes back to 0 if we incrimented from minute 59.
         if(mcClock[arrHr] == 12)
         {//if the hour is 12 we need to reset it back to 1 like in standard time.
            mcClock[arrHr] = 1;

         }
         else//just incriment the hour if it is not 12.
         {
            mcClock[arrHr]++;
            if(currSec == 0 && currSec != mcClock[arrSec] && mcClock[arrHr] == 12)
            {//if we are at hour 12 for the first time we need to adjust the am/pm setting.
               mcClock[arrAP] = (mcClock[arrAP]+1)%2;
            }
         }
      }
      else//if the minute is not 59 then just increment it.
      {
         timeFlag = 1;//Flag which signals that the lcd needs to be updated to reflect the new
time.
         mcClock[arrMin]++;
      }
      mcClock[arrSec] = currSec;//store the returned seconds from epoch time.
   }
   else
   {
      mcClock[arrSec] = currSec;
   }


   //parseTime();
   if(timeFlag && page == 0)//if the time changes (hour and/or minute) the lcd needs to be
updated.
   {
      dispTime();
      timeFlag = 0;
   }
   //printf("Current Time: %d:%d.%d %d\n", mcClock[arrHr], mcClock[arrMin], mcClock[arrSec],
mcClock[arrAP]);
   return;
}

/*******************************************************************************************parseTim
e(void)
   Description: This function parses the hour and minute into individual single digit
   values so that they may be used for other conditions if necessary or to display
   the time as independant digits.
*******************************************************************************************/
```

```
int parsedTime[4];
void parseTime(void)
{
   int hr = mcClock[arrHr];//get the current MC time.
   int min = mcClock[arrMin];//get the current MC time.
   int hrT;
   int hrO;
   int minT;
   int minO;

   if(hr >= 10)//if the hour is a double digit number.
   {
     hrT = 1;//store the tenths place.
      hrO = hr % 10;//retrieve the one's place.
   }
   else//the hour is a single digit number.
   {
      hrT = 0;//store the tenths place
      hrO = hr;//store the one's place.
   }

   if(min >= 10)//if the minute is a double digit number.
   {
      minT = min/10;//devide the minutes by ten to get the tenths place.
      minO = min%10;//retrieve the one's place of the minute.
   }
   else//the minute is a single digit number.
   {
      minT = 0;//store the tenth's place.
      minO = min;//store the one's place.
   }

   //store the parsed time.
   parsedTime[0] = hrT;
   parsedTime[1] = hrO;
   parsedTime[2] = minT;
   parsedTime[3] = minO;

   //printf("mcClock hr: %d\n\n", mcClock[arrHr]);
   printf("Parsed Time: %d%d:%d%d\n",parsedTime[0],parsedTime[1],parsedTime[2],parsedTime[3]);

   return;
}

/*******************************************************************************************
void rtTimerStart(int timerNum, int timerType)
   Description: This function stores the current time into an array in total seconds
   which will be used to determine the time passed (in real time) when the timer is
   stopped.
*******************************************************************************************/
int rtTimes[126][2];//column 1 is for thermostat timers, column 2 are for anything else.
void rtTimerStart(int timerNum, int timerType)
{
   int startSec;
  /*
   rtTimes[0] = mcClock[arrHr];
   rtTimes[1] = mcClock[arrMin];
   rtTimes[2] = mcClock[arrSec];
   rtTimes[3] = mcClock[arrAP];
   */

   //calculating the start time.
   startSec = (mcClock[arrHr]*3600)+(mcClock[arrMin]*60)+mcClock[arrSec];
   if(mcClock[arrAP])
   {//if the time of day is PM then add appropriate seconds.
      startSec = startSec + (mcClock[arrHr]*12);
```

```
   }
   rtTimes[timerNum][timerType] = startSec;//store the start time.
   return;
}

/*******************************************************************************************
int rtTimerEnd(int timerNum, int timerType)
   Description: This function ends a timer. It takes the start time which was stored
   in the rtTimerStart() function and calculates how much time has passed since then.
   The array which contains the timers supports two types of timers, one for the
   temperature rate of changes, and the other for miscellaneous use.
*******************************************************************************************/
int rtTimerEnd(int timerNum, int timerType)
{
   int startSec;
   int endSec;
   int resultSec;
   //3600 seconds in one hour.
   startSec = rtTimes[timerNum][timerType];//retrieve the start time for this timer.

   endSec = (mcClock[arrHr]*3600)+(mcClock[arrMin]*60)+mcClock[arrSec];//calculate current time
in seconds.
   if(mcClock[arrAP])//if the hour is passed 12, add 12 hours worth of seconds to the end time.
   {
      endSec = endSec + (mcClock[arrHr]*12);
   }

   //determine elapsed time.
   if(endSec > startSec)
  {
      resultSec = endSec-startSec;
   }
   else
   {
      resultSec = startSec - endSec;
   }
   //rtTimes[timerNum][timerType] = 0;//reset the timer to 0;
   return resultSec;
}
```

finalHeader.h

```
#ifndef     FinalHeader_H
#define     FinalHeader_H

//Define constants
#define FAN             22
#define COOL            13
#define HEAT            26
#define LCD_D4       104
#define LCD_D5       105
#define LCD_D6       106
#define LCD_D7       107
#define LCD_RS       102
#define LCD_E        103
#define pwrLED       6
#define syncLED         23
#define entSwitch    16
#define dwnSwitch    25
#define upSwitch     24
#define syncSwitch      17
#define rstSwitch    5
#define pwrSwitch    27

#define rows         128
#define columns      128

#define  arrHr       0
#define arrMin       1
#define arrSec       2
#define  arrAP       3

#define thermTimer      0
#define miscTimer    1

#define retCurrTemp     0
#define retSetTemp      1
#define retHum       2

//3-dimensional array z axis index reference constants
#define sendingAF 0  //sending AirFlow
#define  receivingAF 1  //receiving AirFlow

//temps[][]: [Current temp | Set temp | Current Humidity | temp difference | start difference]
#define currTemp  0
#define setTemp      1
#define  tempDiff 3
#define startDiff 4
#define hvacOut      3

//Define Prototypes

   //menuFunctions.c prototypes
   void lcdinitialization(void);
   void lcdBoot(void);
   void updateDisplay(void);
   void mainDisplay(void);
   void settings(int line);
   void settingConfig(int line);
   void errorPage1(int line);
   void errorPage2(void);
   void warningPage(void);
   void display(int, int, int);
   //void SuccessMessage(int line);
```

```
  void syncMenu(int line);
 void manageDeviceMenu(int line);
  void addDeviceMenu(int line);
  void removeDeviceMenu(int line);
  void SuccessMessage(int line, int page);
  void setTimeDisplay(int);
  void page0(void);
   void page1(void);
   void page2(void);
   void page3(void);
   void page4(void);
   void page5and6(void);
   void page7(void);
   void page8(void);
   void page9(void);
   void page10(void);
  void page11(void);
  void page12(void);
  void page13(void);
  void dispTime(void);
  void remDevWarn(void);
  void toRoomMenu(void);


  //syncing.c prototypes
  void initArray(void);
  void populateArrays(void);
  int addTherm(void);
  int addReg(int addToRoom);
  void dispMatrix(void);
  void rmDevAddr(unsigned char address);
  void retrieveTemps(void);
  void setReg(void);
  void initRegFlow(void);
  void initTempsArr(void);
  int countErrors(void);
  void calcDiff(void);
  void hvacControl(void);
  void initRocArray(void);
  void adjustReg(int i, int flowIndex);
  void initFailedCon(void);

  //time.c prototypes
  int getSec(void);
  void initTimeArr(void);
  void updateTime(void);
  void parseTime(void);
  void rtTimerStart(int timerNum, int timerType);
  int rtTimerEnd(int timerNum, int timerType);


  //misc
  int smartDelay(int i);
  void setups(void);
  void bootSequence(void);
  void shutDownSequence(void);
  void updateStatus(void);


#endif
```