```
final_balloon.ino
/**
 * Main file controlling the arduino for a balloon launch experiment
 * Sets up all sensors and logger
 * Fills array with calls to getter methods in header files to get a
data array
 * Uses calls to header files to log the data to a microSD card, and
 * once every ten loops radio transmits data live
 * Also checks for when to cutdown the balloon
 */
//#include <Wire.h> not needed: included elsewhere, in BME.cpp
#include "humidity.h"
#include "spectrometer/spectrometer.h"
#include "tx.h"
#include "sd_logger.h"
#include "sensors.h"
#include "analog_individuals.h"

/*
 * Index | data item
 * 0 | Time
 * 1 | GPS
 * 2 | Altitude
 * 3 | Pressure
 * 4 | Outside Temperature (C)
 * 5 | Humidity (%)
 * 6 | Inside Payload temperature (C)
 * 7 | Spectrometer data (Alec's individual)
 * 8 | UVB data (Caroline's individual)
 * 9 | Methane data (John's individual)
 * 10 | Solar data (Vivian's individual)
 * 11 | Light Intensity Data (Vivian's individual)
 *
 * Luke and Simon's individuals are on a seperate arduino
 */
#define numObs 12
String data_arr[numObs];
const double PRESSURE_TO_CUT_DOWN = 1.08;
double pressure_value = 100.0;
String pressure_string = "";
int cut_down_pin = 30;
String start_time = "8:32:15";
String current_time = "8:32:15";
double seconds_passed_to_cutdown = 5400.0;
String current_temp = "21.7";
double temp_to_cutdown = -10.3;
int num_times_cutdown_criteria_met = 0;
int tx_counter = 0;

/**
```

```
 * Sets up all sensors by calling their setup methods in other files
 * Prepares cutdown by setting cutdown pin to output
 * Get's the starting time
 */
void setup() {
  //start serial comms with computer
  Serial.begin(9200);
  delay(4000);

  pinMode(cut_down_pin, OUTPUT);

  //setup tx and SD card logger;
  setupTx();
  setupLogger();

  // setup clock and the temp and GPS sensors and BME
  setupBME();
  Serial.println("bme setup complete");
  setupSensors();
  Serial.println("sensor setup complete");

  //individual experiments:
  setupSpectrometer(); //Alec's
  setupIndividuals(); //Caroline's, John's, Vivian's

  start_time = getClock();
}

/**
 * Returns the difference between 2 times in terms of seconds
 */
int getTimeDifference(String time1, String time2)
{
  return (getTimeInSeconds(time1) - getTimeInSeconds(time2));
}

/**
 * Returns the number of seconds from a time string that is formatted
with hour: minute: second
 */
double getTimeInSeconds(String theTime)
{
  int firstColon = theTime.indexOf(':');
  String hours = theTime.substring(0, firstColon);
  int secondColon = theTime.indexOf(':', firstColon + 1);
  String minutes = theTime.substring(firstColon + 1, secondColon);
  String seconds = theTime.substring(secondColon + 1);

  double hourSeconds = hours.toDouble() * 3600;
  double minuteSeconds = minutes.toDouble() * 60;
```

```
  return hourSeconds + minuteSeconds + seconds.toDouble();
}

/**
 * Loop runs constantly while the arduino has power
 * Fills data array with sensor data (See index at the top of the
file)
 * Calls sd logger method to log the data array
 * Once every ten loops calls radio transmission method in tx.h to
transmit data array down
 * Checks safety measures and pressure reading to see if cutdown
should be triggered
 */
void loop() {
  current_time = getClock();
  data_arr[0] = current_time;

  data_arr[1] = getGPS();

  data_arr[2] = getAltitude();

  pressure_string = getPressure();
  data_arr[3] = pressure_string;
  pressure_value = pressure_string.toDouble();

  String currentTemp = getBMETemp();
  data_arr[4] = getBMETemp();

  data_arr[5] = getBMEHumidity();

  data_arr[6] = getTemp();

  data_arr[7] = checkSpectrometer();

  data_arr[8] = getUV();

  data_arr[9] = getMethane();

  data_arr[10] = getSolar();

  data_arr[11] = getLightIntensity();

  tx_counter++;
  if (tx_counter == 10 ){
      sendDataAsBytes(data_arr, numObs);
      tx_counter = 0;
  }

  logDataToSD(data_arr, numObs);
```

```
  //todo: test cutdown safety measures
  if (num_times_cutdown_criteria_met <= 10) { //don't perpetually run
this loop post-cutdown
    if (getTimeDifference(current_time, start_time) >
seconds_passed_to_cutdown) { //1.5 hours passed
      if (current_temp.toDouble() < temp_to_cutdown) { // under -10
degrees C
        if (pressure_value < PRESSURE_TO_CUT_DOWN) {  //@100k feet
          num_times_cutdown_criteria_met += 1;
          if (num_times_cutdown_criteria_met > 10) { //these
measurements happen 10 times
            digitalWrite(cut_down_pin, HIGH);
          }
        }
      }
    }
  }
  delay(200);
}
```

analog_individuals.h

```
/**
 * code for Vivian's, Caroline's, and John's individual experiments,
 * which involve reading analog input.
**/

#ifndef INC_6THSENSE_ANALOG_INDIVIDUALS_H
#define INC_6THSENSE_ANALOG_INDIVIDUALS_H

#define uv_pin A7
#define methane_pin A5
#define solar_pin A3
#define light_intensity_pin A4

/**
 * Setup sensors by setting their pins to input
 */
void setupIndividuals() {
    pinMode(uv_pin, INPUT);
```

```
    pinMode(methane_pin, INPUT);
    pinMode(solar_pin, INPUT);
    pinMode(light_intensity_pin, INPUT);
}

/**
 * Returns string of value of the UV analog sensor
 */
String getUV() {
    return String(analogRead(uv_pin));
}

/**
 * Returns string of value of the Methane analog sensor
 */
String getMethane() {
    return String(analogRead(methane_pin));
}

/**
 * Returns string of value of the Solar analog sensor
 */
String getSolar() {
    return String(analogRead(solar_pin) * 5.0/1024.0);
}

/**
 * Gets string of value of the Light Intensity analog sensor
 * Implements equation to convert analog value into voltage
 * Returns the voltage
 */
String getLightIntensity() {
    float voltage = analogRead(light_intensity_pin) * 5.0/102.0;
    return String(voltage);
}
#endif //INC_6THSENSE_ANALOG_INDIVIDUALS_H
```

humidity.h

```
/**
 * Methods that setup and get values from the BME280 Temperature and
Humidity sensor
 * This sensor is outside the payload so it will get exterior
temperature and humidity
 */
//#include <SPI.h>
#include "adafruit/Adafruit_BME280.h"
#include "adafruit/Adafruit_BME280.cpp"

Adafruit_BME280 bme; // I2C address 0x77

float sensorReading; //create global variable to store sensor reading

/**
 * Sets up the BME sensor and prints to serial the success of the
setup
 */
void setupBME() {
    bool status;
    Serial.println("starting BME");
    status = bme.begin();
    Serial.println("bme status: "); Serial.println(status);
    if (!status) {
        Serial.println("Could not find a valid BME280 sensor, check
wiring!");
        while (1);
    }
}

/**
 * Returns string with the humidity percentage from the BME sensor
 */
String getBMEHumidity() {
    String theHumidity = String(bme.readHumidity());
    return theHumidity;
}

/**
 * Returns string with the Temperature in Celsius from the BME sensor
 */
String getBMETemp()
{
    String theTemp = String(bme.readTemperature());
    return theTemp;
}
```

sensors.h

```
/**
 * GPS, RTC, Clock, and barometer methods
 */

#include <SoftwareSerial.h>

#include <Wire.h>
#include "RTClib/RTClib.h"
#include "RTClib/RTClib.cpp"

//Sets the RX and TX pins for GPS Serial communication
SoftwareSerial gpsSerial(10, 11); // RX, TX (TX not used)
const int sentenceSize = 80;

char sentence[sentenceSize];

//values for the thermistor calculation
int ThermistorPin = A1;
int Vo;
float R1 = 10000;
float logR2, R2, T, Tc, Tf;
float c1 = 1.009249522e-03, c2 = 2.378405444e-04, c3 =
2.019202697e-07;

//variables and values for the barometer and altitude calculations
int pressure = A0;
double pressvalue = 0;
double pressunits = 0;
double altitude = 0;
const double PRESSURE_SEALEVEL = 101325;
const double PRESSURE_11K = 22629.5;
const double PRESSURE_20K = 5473.875;


RTC_DS1307 rtc; //i2c address 0x68

/**
 * Sets up GPS sensor, all others do not need to be set up
 */
void setupSensors()
```

```
{
  gpsSerial.begin(9600);

  // not needed. This method simply calls Wire.begin(), which is
called by the BME sensor.
  //rtc.begin();
}

/**
 * Decodes GPS Serial and fills buffer with value
 * Depending on index, can get longitude, latitude, Direction in each
way, and more
 */
void getField(char* buffer, int index)
{
  int sentencePos = 0;
  int fieldPos = 0;
  int commaCount = 0;
  while (sentencePos < sentenceSize)
  {
    if (sentence[sentencePos] == ',')
    {
      commaCount ++;
      sentencePos ++;
    }
    if (commaCount == index)
    {
      buffer[fieldPos] = sentence[sentencePos];
      fieldPos ++;
    }
    sentencePos ++;
  }
  buffer[fieldPos] = '\0';
}

/**
 * Returns String of GPS Coordinates and direction
 * Reads in gps serial one character at a time until the terminator
character is reached
 * Then used getField to decode the string
 * Puts the different parts of the Coordinates together and returns it
 */
String getGPS()
{
  bool gotSentence = false;
  while(!gotSentence)
  {
    static int i = 0;
    if (gpsSerial.available())
    {
```

```
      char ch = gpsSerial.read();
      if (ch != '\n' && i < sentenceSize)
      {
        sentence[i] = ch;
        i++;
      }
      else
      {
        sentence[i] = '\0';
        i = 0;
        char field[20];
        getField(field, 0);
        if (strcmp(field, "$GPRMC") == 0)
        {
          gotSentence = true;
          getField(field, 3);  // number
          String lat_num = String(field);
          getField(field, 4); // N/S
          String lat_direction = String(field);

          getField(field, 5);  // number
          String long_num = String(field);
          getField(field, 6);  // E/W
          String long_direction = String(field);

          return String("Lat: " + lat_num + lat_direction + " Long: "
+ long_num + long_direction);
        }
      }
    }
  }
  Serial.println(gotSentence);
}

/**
 * Returns String with current time using Real Time Clock Chip (RTC)
 */
String getClock()
{
  DateTime now = rtc.now();
  String hour = String(now.hour(), DEC);
  String minute = String(now.minute(), DEC);
  String second = String(now.second(), DEC);
  String total = String(hour + ":" + minute + ":" + second);
  return total;
}

/**
 * Gets interior temperature from thermistor using an analog pin
 * Then performs calculation to get the temperature in celsius
```

```
 * Returns String with interior temperature in celsius
 */
String getTemp()
{
  Vo = analogRead(ThermistorPin);
  R2 = R1 * (1023.0 / (float)Vo - 1.0);
  logR2 = log(R2);
  T = (1.0 / (c1 + c2*logR2 + c3*logR2*logR2*logR2));
  Tc = T - 273.15;
  Tf = (Tc * 9.0)/ 5.0 + 32.0;
  return String(Tc);
}

/**
 * Gets pressure value from baromter using an analog pin
 * Converts value to kilopascals (kPa)
 * Returns string with pressure value in kPa
 */
String getPressure()
{
  //take the average of 10 readings
  int sumPressureVoltages = 0;
  for (int i = 0; i < 10; i++) {
    sumPressureVoltages += analogRead(pressure);
  }
  pressvalue = ((double) sumPressureVoltages) / 10;
  // gets pressure in kPa
  pressunits = (0.2554 * (pressvalue)) - 25.295;
  return String(pressunits);
}

/**
 * Gets Altitude using expontial equation and returns that value
 */
double getAltitude()
{
  double presskpa = getPressure().toDouble();
  double pressurePa = presskpa * 1000;
  if(PRESSURE_11K < pressurePa)
  {
    altitude = (1 - pow(pressurePa / PRESSURE_SEALEVEL, 1 / 5.255816))
* 44329;
  }
  else if(PRESSURE_11K > pressurePa && pressurePa > PRESSURE_20K)
  {
    altitude = 10999 - 6341.4 * log(pressurePa / (PRESSURE_SEALEVEL *
0.22361));
  }
  else
  {
```

```
    altitude = (pow(pressurePa / PRESSURE_SEALEVEL, 1 / -34.16319) -
0.988626) * 198903;
  }
  return altitude;
}
```

sd_logger.h

```
/**
 * Code for the local microSD card logger. Sets up all csv headers and
includes
 * methods to log a data array. Final file should look like a csv for
easy processing.
 * Note that a file called "logger.txt" needs to be present in the
root directory
 * of the SD card for this code to work.
 */

#ifndef INC_6THSENSE_SD_LOGGER_H
#define INC_6THSENSE_SD_LOGGER_H

#include <SD.h>
File myFile;

/**
 * Sets up the microSD logger to be logged later
 * Also begins by logging the data column headers for the csv format
 * Will print error message to serial if setup fails
 */
void setupLogger() {
    const int chipSelect = 53;
    // On the Ethernet Shield, CS is pin 4. It's set as an output by
default.
    // Note that even if it's not used as the CS pin, the hardware SS
pin
    // (10 on most Arduino boards, 53 on the Mega) must be left as an
output
    // or the SD library functions will not work.
    pinMode(chipSelect, OUTPUT);
```

```
    if (!SD.begin(chipSelect)) {
        Serial.println("initialization failed!");
        return;
    }
    Serial.println("SD init done.");

    myFile = SD.open("logger.txt", FILE_WRITE);

    // if the file opened okay, write to it:
    if (myFile) {
        Serial.print("Writing headers to logger.txt...");

        //column heads
        myFile.println("Time, GPS, Pressure, Outside_temp, Humidity,
Inside_temp, Spectrometer, UVB, Methane, Solar");
        // close the file:
        myFile.close();
        Serial.println("done.");
    } else {
        // if the file didn't open, print an error:
        Serial.println("error opening the logger.txt");
    }
}

/**
 * Logs an array of Arduino Strings with size of the array being
num_items
 * Will put commas in between logging entries to keep csv format
 */
void logDataToSD(String data[], int num_items) {
    myFile = SD.open("logger.txt", FILE_WRITE);
    if (myFile) {
        for (int i = 0; i < num_items; i++) {
            myFile.print(data[i]);
            if (i < num_items - 1) {
                myFile.print(", ");
            }
        }
        myFile.println(" ");
        myFile.close();
    } else {
        Serial.println("error opening logger.txt");
    }
}

#endif //INC_6THSENSE_SD_LOGGER_H
```

tx.h

```
/**
 * Methods to setup radio and send radio transmissions
 */
#define shutdownpin 22

/**
 * Sets up radio transmitter by setting shutdown pin to output and
high
 */
void setupTx() {
    pinMode(shutdownpin, OUTPUT);
    digitalWrite(shutdownpin, HIGH);
    Serial1.begin(9600);
}

/**
 * Fills a byte array with an Arduino String
 */
void tBA(String n, byte *arr) {
    for (int i = 0; i < n.length(); i++) {
        byte b = int(n.charAt(i));
        arr[i] = b;
    }
}

/**
 * Individually feeds each byte to the transmitter
 */
void txArr(byte *buffer, int size) {
    for (int i = 0; i < size; i++) {
        Serial.write(buffer[i]);
        Serial1.write(buffer[i]);
    }
}

/**
 * Radio transmits the start sequence then the array of Strings
delimited by six dashes
 * to the low level tranmission function
```

```
 */
void sendDataAsBytes(String data[], int num_items) {
    byte start_sequence[] = {42, 42, 42, 42, 42, 42};
    byte next_data_sequence[] = {45, 45, 45, 45, 45, 45};

    txArr(start_sequence, 6);
    for(int i = 0; i < num_items; i++) {
        delay(100);
        byte value[data[i].length()];
        tBA(data[i], value);
        txArr(value, data[i].length());
        if (i < num_items - 1) {
            delay(100);
            txArr(next_data_sequence, 6);
        }
    }
    Serial.println(" ");
}
```

secondary_arduino.ino

```
/**
 * Runs the secondary arduino by keeping track of time with calls to
ellapsedMillis,
 * and logging accelerometer data to the OpenLog microSD Logger
 */
#include <SoftwareSerial.h>
#include "elapsedMillis/elapsedMillis.h"
#include <MPU6050_tockn.h>
#include <Wire.h>

SoftwareSerial OpenLog(4, 5);
elapsedMillis timeElapsed;
int count;
int accelCount;
#define numAccelCounts 100
String accel_arr[numAccelCounts];
MPU6050 mpu6050(Wire);

/**
 * Sets up the OpenLog to be able to log
```

```
 * Sets up the mpu6050 accelerometer chip and calibrates it's gyro
 */
void setup() {
    Serial.begin(9600);
    OpenLog.begin(9600);
    delay(500);
    count = 0;
    accelCount = 0;
    timeElapsed = 0;
    mpu6050.begin();
    mpu6050.calcGyroOffsets(true);
}

/**
 * Each loop mpu6050 gets updated and the data goes into an
accelerometer data array
 * Every 100 observations the accelerometer will log it's data with
OpenLog
 * Logging takes time and makes it so the accelerometer is not getting
data during that time
 * so in order to get more frequent accelerometer data it only logs
once every 100 times
 *
 * When Logging, the data will only be logged if the accelerometer
data is good, so often there
 * will not be the full 100 observations logged
 *
 * Every 10,000 milliseconds aka 10 seconds the number of milliseconds
is logged, so that
 * the accelerometer data can be matched up with the altitude data in
the primary Arduino
 * by using time stamps
 */
void loop() {
//    if (Serial.available() > 0) {
//      if (Serial.read() > -1) {
//        count++;
//      }
//    }
    if (timeElapsed > 10000) {
        OpenLog.print("Millis: ");
        OpenLog.println(millis());
//        OpenLog.print("CPM: ");
//        OpenLog.println(count * 6);
        timeElapsed = 0;
//        count = 0;
    }
    mpu6050.update();
    accel_arr[accelCount] = getTheAccel();
    accelCount ++;
```

```
    if (accelCount == numAccelCounts - 1)
    {
      accelCount = 0;
      for (int i = 0; i < numAccelCounts; i ++)
      {
        String the_accel_data = accel_arr[i];
        if (the_accel_data.length() > 7)
        {
          OpenLog.print(the_accel_data);
          OpenLog.print(",\n");
        }
      }
    }
}

/**
 * Gets the Accelerometer observation and returns it
 * Uses mpu6050 library to get the X, Y, and Z axis
 * acceleration values then seperates them by commas
 */
String getTheAccel()
{
  String accx = String(mpu6050.getAccX());
  String accy = String(mpu6050.getAccY());
  String accz = String(mpu6050.getAccZ());
  return String(accx + "," + accy + "," + accz);
}
```