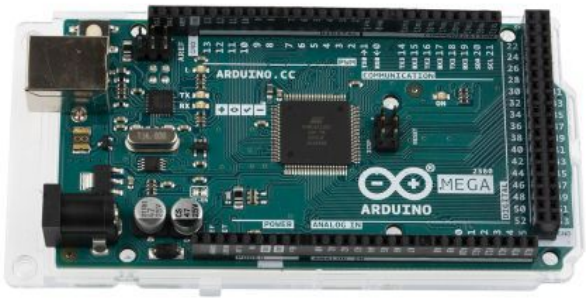
ChamberV1

Hardware

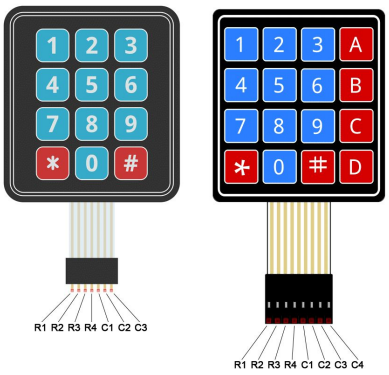
In order to function properly the following components must be included.

Board



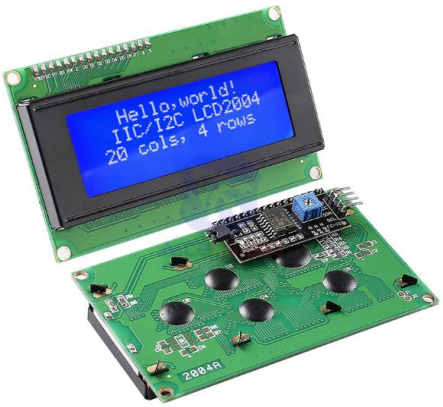
This iteration is designed to be used with an Arduino Mega board.

Keypad



A keypad of size 4x3 or 4x4 can be used, however only a 4x3 is needed.

20x4 Display (I2C)

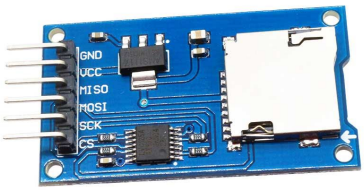


This iteration is designed for use with a 20x4 LCD display connected via I2C.

RTC

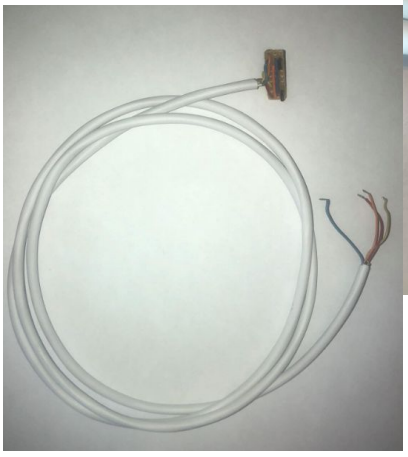
This iteration is designed for use with a ds3231 RTC connected via I2C. As of now it is connected directly to 5v and does not use a battery.

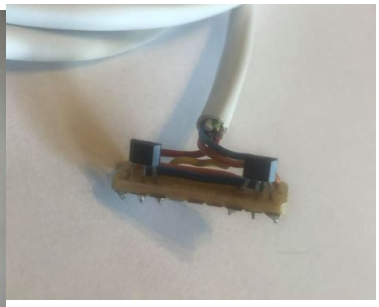
MicroSD adapter



This iteration is designed for use with a MicroSD adapter connected via SPI.

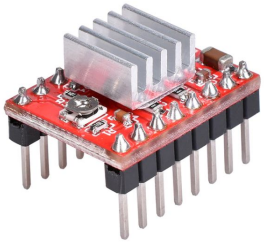
Hall effect Sensor





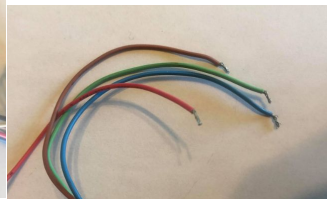
A custom Hall effect sensor module is used. Vcc and ground are wired together, and each sensor has its own unique output wire.

Stepper driver A4988



The stepper motor is driven by an a4988 stepper driver.

Stepper motor



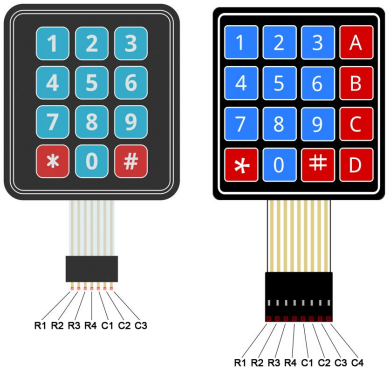
The stepper motor itself is wired for unipolar operation, but by ignoring the 2 (white and black) ground wires and just using the 4 coil wires, we can wire it just as a bipolar motor.

Power

The motor is run off of a 12v LiPo battery. The board itself is run of a 9v battery

Wiring + Defining Pin Numbers

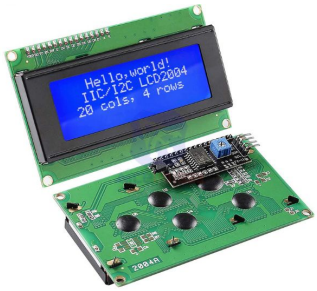
Keypad



The Keypad can be wired anywhere, from left to right the pins must be defined within the arduino sketch as follows.

```
//Pins for keypad
#define kpadr1 R1
#define kpadr2 R2
#define kpadr3 R3
#define kpadr4 R4
#define kpadc1 C1
#define kpadc2 C2
#define kpadc3 C3
```

20x4 Display (I2C)

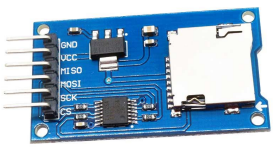


The display is wired into SDA, SCL, VCC and Ground

RTC

The RTC is wired into SDA, SCL, VCC and Ground

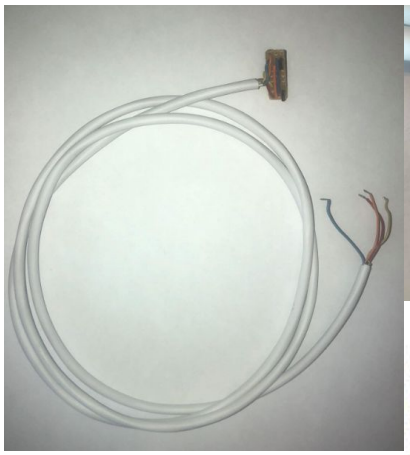
MicroSD adapter

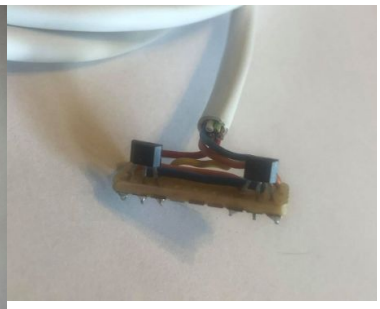


The MicroSD adapter is connected to SPI via MISO (PIN 50), MOSI (PIN 51), SCK (PIN 52). The CS pin must be wired in and defined in the sketch.

```
//SD PIN
#define cs PIN
```

Hall effect Sensor

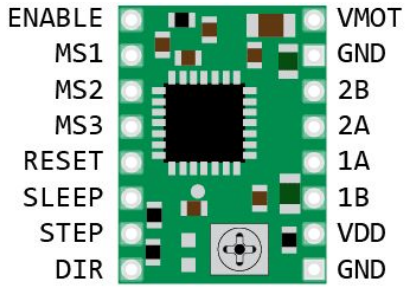




The hall effect sensor is wired red to VCC, blue to Ground. Orange is the home (closed) position sensor, Yellow is away (open) position sensor. Define them in the sketch here. Pull up the yellow and orange wires to VCC

```
//Hall effect sensors
#define awaySensor 10
#define homeSensor 12
```

Stepper and driver




```
//Stepper pins
#define dirPin DIR
#define stepPin STEP
#define sleepPin SLEEP
```

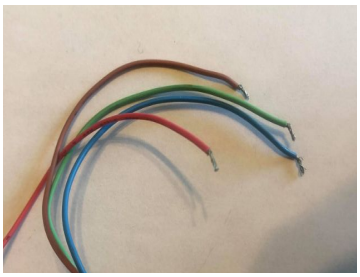
For the motor to run the correct direction and not destroy the chamber you must make sure it is wired correctly.

RESET pulled up to VCC

SLEEP, STEP, and DIR must be connected and defined.

VMOT to 12v power

VDD to Arduino VCC



The motor must be wired to the displayed pinout.

Green - 1B

Brown - 1A

Red - 2A

Blue - 2B

Power

The 12v battery is connected to the stepper driver. A 100uf capacitor is connected in parallel to smooth out potential voltage spikes.

The arduino itself is connected to a 9v battery via the dc in

PCB

The PCB makes it easy to set up the chamber. Simply connect the hall effect sensor and stepper motor to the board, being careful to match the wire colours. Connect the red wire to 5v and black to ground. Each connecting wire is labeled, the white and grey wires connect the smaller I2C board to the main board. Connect white to SDA (PIN 20) and grey to SCL (PIN 21). The sd reader must be connected via SPI. Connect MISO to PIN 50, MOSI to PIN 51 and SCK to PIN 52 and CS to PIN 49. Connect HOME to PIN 2 and AWAY to PIN 3, and connect DIR to PIN 10, STEP to PIN 9 and SLEEP to PIN 8. Connect the keypad to PINS 31 33 35 37 39 41 43. The HOME, AWAY, DIR, STEP, SLEEP and keypad can all be customized in the arduino sketch, but these are the default locations. Once setup, power the board with a power source such as a 9v battery, and power the pcb with a 12v battery.

Operation

When prompted for an input, use the keypad to enter a number. “*” will clear the current input, and “#” will enter it into the system.

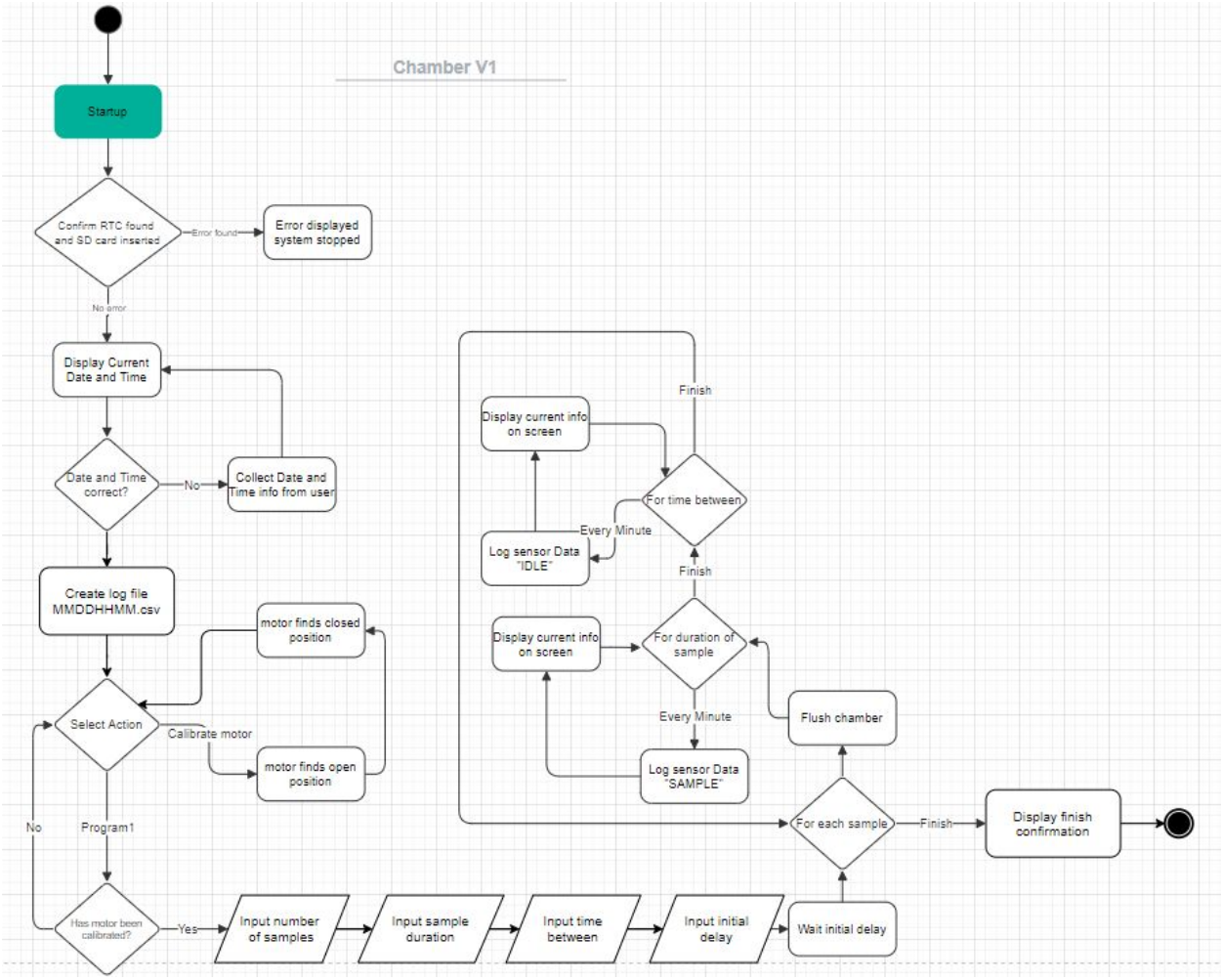
When booted, the first thing the chamber will do is make sure the RTC and SD card modules are functional, if not, an error is displayed. The date and time will be confirmed and a CSV file will be created with the name “MMDDHHMM.csv”. The system will prompt for program selection, currently 1 will run the standard program, and 3 will calibrate the motor. In order to run the program, the motor must be calibrated.

Data is logged in a CSV format. “YYYY-MM-DD HH:MM:SS, ACTION, sensor data”. ACTION is what the chamber was doing such as operating the lid, sampling, or idling. Data is recorded every minute, or when an action happens.

When option 3 is selected the motor is calibrated. Please make sure the chamber is free from any obstructions. The chamber lid will open fully, and then close.

When option 1 is selected, the user will be asked for the number of samples, the sample duration, the time between samples, and an initial delay. The system will then wait the initial delay and then start its main task. For each sample, the chamber will first flush by opening and holding the lid open for a set amount of time, and then closing. Then the sample will start. After the sample duration, the system will idle for the time between samples, and then start the process again.

Flow chart


Chamber V1
Startup
Confirm RTC found and SD card inserted
Error found
Error displayed system stopped
No error
Display Current Date and Time
Date and Time correct?
No
Collect Date and Time info from user
Create log file MMDDHHMM.csv
motor finds closed position
Select Action
Calibrate motor
motor finds open position
No
Program1
Has motor been calibrated?
Yes
Input number of samples
Input sample duration
Input time between
Input initial delay
Wait initial delay
For each sample
Finish
Display finish confirmation
Display current info on screen
For time between
Finish
Every Minute
Log sensor Data "IDLE"
Display current info on screen
For duration of sample
Finish
Every Minute
Flush chamber
Log sensor Data "SAMPLE"

Customization

Flush Time

The amount of time the chamber spends open to flush the system is customizable by changing the flushTime definition.

```
//Time for flush
#define flushTime 10
```

Editing Sensors

Sensors used for logging data can be easily added or removed. To add a sensor that will have its data logged, simply connect it any way to the arduino. Within the sketch, you will initialise the sensor and it library. Find the logSensors() function and customize the data String however you want. Make sure to keep the values separated by comments.

Software

The sketch for the chamber uses a number of standard arduino libraries, as well as a few custom ones for adding easier access to functions for the connected modules.

Custom Libraries:

MyLCD2

I made this library to streamline using a 20x4 display for both printing and in conjunction with a keypad

PUBLIC METHODS:

MyLCD2() requires &keypad and &lcd

MAIN SKETCH:

```
//Create MyLCD
MyLCD2 myLcd(&ckeypad, &clcd);
```

MyLCD2:

```
LiquidCrystal_I2C *_clcd;
Keypad *_ckeypad;

MyLCD2::MyLCD2(Keypad * ckeypad, LiquidCrystal_I2C * lcd) {
  _ckeypad = ckeypad;
  _clcd = lcd;
}
```

initialize() must be called to start the LCD

```
void MyLCD2::initialise() {
  _clcd->init();
  _clcd->backlight();
  _clcd->setCursor(0, 0);
  _clcd->print("MYLCD Started");
  delay(1000);
  _clcd->clear();
}
```

displayInfo() takes four strings in and prints one on each row of the LCD screen

```
void MyLCD2::displayInfo(String l1, String l2, String l3, String l4) {
  _clcd->clear();
  _clcd->setCursor(0, 0);
  _clcd->print(l1);
  _clcd->setCursor(0, 1);
  _clcd->print(l2);
  _clcd->setCursor(0, 2);
  _clcd->print(l3);
  _clcd->setCursor(0, 3);
  _clcd->print(l4);

}
```

displayTime() takes a DateTime and displays the HH:MM in the bottom right of the LCD

```
void MyLCD2::displayTime(DateTime now) {
  _clcd->setCursor(15, 3);
  _clcd->print(String(twoChar(now.hour()) + ":" + (twoChar(now.minute()))));
}
```

getInt() takes a string input, and returns an int. The menu acts dynamically and allows for the user to see what they are inputting, and clear the value, if an incorrect value is entered, the input can be cleared with (*), or confirmed with (#)

```
int MyLCD2::getInt(String question) {
  char key;
  while (true) {
    _clcd->setCursor(0, 0);
    _clcd->print(question);
    key = getinput();
    if (key == '#') {
      _val = _tempVal.toInt();
      clearVal();
      return _val;
    } else if (key == '*') {
      clearVal();
    } else {
      printInput(key);
    }
  }
}
```

PRIVATE METHODS:

getinput() waits for an input from the keypad and returns the entered char.

```
char MyLCD2::getinput() {
  char tempkey;
  tempkey = _ckeypad->getKey();
  while (true) {
    tempkey = _ckeypad->getKey();
    if (tempkey != NO_KEY) {
      return tempkey;
    }
  }
}
```

clearVal() clears the temporary input variable and clears the LCD display.

```
void MyLCD2::clearVal() {
  _tempVal = "";
  _clcd->clear();
}
```

printInput() adds the inputted char to the temporary string, and displays the updated string on the second line

```
void MyLCD2::printInput(char in) {
  _clcd->setCursor(0, 1);
  _tempVal = _tempVal + in;
  _clcd->print(_tempVal);
}
```

MySD

MySD() requires &rtc and CS pin

MAIN SKETCH:

```
//Create MySD
MySD mysd(&rtc, cs);
```

MySD:

```
MySD::MySD(RTC_DS3231 *rtc, int cs) {
  pinMode(cs, OUTPUT);
  _cs = cs;
  _rtc = rtc;
}
```

initialise() returns boolean true if sd card starts properly, and false if there was an error.

```
bool MySD::initialise() {
  return SD.begin(_cs);

}
```

setFileName() sets the _filename for writing to the sd card.

```
void MySD::setFileName(String filename) {
  _filename = filename;
}
```

logData() takes in a string for current action, and a string with all the sensor data WITHOUT a terminating ",". Written to the SD card is "YYYY/MM/DD HH:MM:SS, action,data,\n"

```
void MySD::logData(String action, String data) {
  File file = SD.open(_filename, FILE_WRITE);
  DateTime now = _rtc->now();
  String thisTime = String(now.year() + String("/") + now.month() + String("/") + now.day() + String(",")
  file.println((thisTime + String(",") + action + String(",") + data + String(",")));
  file.close();

}
```

MyStepper

MyStepper() requires &stepper and int sleepPin

MAIN SKETCH:

```
//Create MyStepper
MyStepper mystepper(&steppermotor, sleepPin);
```

MyStepper:

```
MyStepper::MyStepper(AccelStepper *stepper, int sleepPin) {
  _stepper = stepper;
  _sleepPin = sleepPin;
}
```

isCalibrated() returns true if the motor has been calibrated

```
bool MyStepper::isCalibrated() {
  return _calibrated;
}
```

calibrateStepper() checks if home and away sensors have been set. The chamber then opens until it reaches the open sensor. When it reaches the sensor, it saves the position

and then closes until it reaches the closed position. Returns string when calibration successful or if HOME and AWAY Sensors have not been set yet.

```
String MyStepper::calibrateStepper() {
  digitalWrite(_sleepPin, HIGH);
  delay(500);
  _stepper->setMaxSpeed(350);
  _stepper->setAcceleration(225);

  if (_homeSet & _awaySet) {
    _stepper->move(10000);
    while (!digitalRead(_awaySensor) == LOW) {
      _stepper->run();
    }
    _awayPosition = _stepper->currentPosition();
    _stepper->setCurrentPosition(_awayPosition); //stops the motor without deceleration
    delay(500);
    _stepper->move(-10000);
    while (!digitalRead(_homeSensor) == LOW) {
      _stepper->run();
    }
    _homePosition = _stepper->currentPosition();
    _stepper->setCurrentPosition(_homePosition); //stops the motor without deceleration

    _calibrated = true;
    digitalWrite(_sleepPin, LOW);
    delay(500);
    return "CALIBRATION SUCCESFUL";
  } else {
    return "Please set HOME and AWAY";
  }
```

goTo() moves steps in positive (CW) or negative (CCW) direction. Does not stop for any sensors so be aware

```
void MyStepper::goTo(int pos) {
  _stepper->setMaxSpeed(350);
  _stepper->setAcceleration(200);
  digitalWrite(_sleepPin, HIGH);
  _stepper->move(pos);
  while (!_stepper->distanceToGo() == 0 ) {
    _stepper->run();
  }

}
```

goHome() moves the stepper to the saved home position. Will stop when either the position is reached or sensor reads LOW. only runs if calibrated.

```
String MyStepper::goHome() {
  if (_calibrated) {
    _stepper->moveTo(_homePosition);
    for (int i = 0; i < 50; i++) {
      _stepper->run();
    }
    while (!digitalRead(_homeSensor) == LOW & !_stepper->distanceToGo() == 0) {
      _stepper->run();
    }
    _stepper->setCurrentPosition(_homePosition); //stops without needing deceleration
    return "Moved Home";
  } else {
    return "Please Calibrate";
  }

}
```

goAway() moves the stepper to the saved away position. Will stop when either the position is reached or sensor reads LOW. only runs if calibrated.

```
String MyStepper::goAway() {
  if (_calibrated) {
    _stepper->moveTo(_awayPosition);
    for (int i = 0; i < 50; i++) {
      _stepper->run();
    }
    while (!digitalRead(_awaySensor) == LOW & !_stepper->distanceToGo() == 0) {
      _stepper->run();
    }
    _stepper->setCurrentPosition(_awayPosition); //stops without needing deceleration
    return "Moved Away";
  } else {
    return "Please Calibrate";
  }
}
```

sleep() puts the motor to sleep, do not call under high load

```
void MyStepper::sleep() {
  digitalWrite(_sleepPin, LOW);
  delay(500);
}
```

wake() wakes up the motor

```
void MyStepper::wake() {
  digitalWrite(_sleepPin, HIGH);
  delay(500);
}
```

setHomeSensor() sets the homeSensor pin and sets _homeSet boolean to true

```
void MyStepper::setHomeSensor(int pin) {
  _homeSensor = pin;
  _homeSet = true;
}
```

setAwaySensor() sets the awaySensor pin and sets _awaySet to true.

```
void MyStepper::setAwaySensor(int pin) {
  _awaySensor = pin;
  _awaySet = true;
}
```

Main Sketch:

StandardRun() collecting number of samples, sample length, time between, and initial delay from the user. Waits the initial delay. and for each sample, the chamber is flushed, sample time is waited, time between is waited and then does it again. Logging all sensor data each minute.

```
void StandardRun() {
  while (!boolVariables) {
    numSamples = myLcd.getInt("How many samples?");
    sampleLength = myLcd.getInt("Duration (min)");
    timeBetweenSamples = myLcd.getInt("Time Between? (min)");
    initDelay = myLcd.getInt("InitialDelay? (min)");
    boolVariables = true;
  }
  for (int i = 0; i < initDelay; i++) { //Initial delay, prints lcd to display what is going on
    myLcd.displayInfo(String("Waiting: " + String(initDelay - i) + " mins"), "until First sample", String(String(dht.readHumidity()) + "%        " + String(dht.readTemperature())
    myLcd.displayTime(rtc.now());
    delay(minuteToMili(1));
  }
  for (int i = 0; i < numSamples; i++) {//run for each sample
    flushChamber();
    for (int x = 0; x < sampleLength; x++) {
      mysd.logData("Sample logging", logSensors());
      displayCurrentInfo(x, i);
      delay(minuteToMili(1));
    }
    for ( int x = 0; x < timeBetweenSamples; x++) {//After the "sample" while idling, data is still logged but the fan does not run (saves power)
      myLcd.displayInfo(String("SAMPLE " + String(i + 1) + " OF " + String(numSamples) + " mins"),
                        String("ELAPSED " + String(sampleLength) + " OF " + String(sampleLength)), "Currently Idle", "");
      myLcd.displayTime(rtc.now());
      mysd.logData("Idle logging", logSensors());
      delay(minuteToMili(1));
    }
  }
  displayCurrentInfo(numSamples - 1, sampleLength);
  while (true);

}
```

displayCurrentInfo() is used when running the program. Displays which sample it is on, time elapsed in the sample, current humidity and temperature, and current time.

```
void displayCurrentInfo(int x, int i) {
  myLcd.displayInfo(String("SAMPLE " + String(i + 1) + " OF " + String(numSamples)),
                    String("ELAPSED " + String(x) + " OF " + String(sampleLength) + " mins"),
                    String(String(dht.readHumidity()) + "%        " + String(dht.readTemperature()) + "C"),
                    String(""));
  myLcd.displayTime(rtc.now());
}
```

closeLid() moves the stepper to the home position and puts it to sleep, logs the action.

```
void closeLid() {
  mystepper.goHome();
  mystepper.sleep();
  mysd.logData("Closed lid", logSensors());
}
```

openLid() wakes the stepper and moves it to the open position, logs the action.

```
void openLid() {
  mystepper.wake();
  mystepper.goAway();
  mysd.logData("Opened lid", logSensors());
}
```

flushChamber() opens and holds the lid open for a determined amount of time, then closes it. Logs chamber flushed

```
void flushChamber() {
  myLcd.displayInfo("Flushing Chamber", "Opening Lid", "", "");
  openLid();
  delay(secToMili(flushTime));
  myLcd.displayInfo("Flushing Chamber", "Closing Lid", "", "");
  closeLid();
  mysd.logData("Chamber Flushed", logSensors()); //Logging Chamber flush
}
```

minToMili() returns minute input to milli for use with delay()

```
long minuteToMili(int mins) {
  return mins * 60000;
}
```

secToMili() returns seconds input to milli for use with delay()

```
long secToMili(int sec) {
  return sec * 1000;
}
```

logSensors() returns a string of sensor data, this function should be customized with appropriate sensors. Do not add a final ","

```
String logSensors() {
  String data = "";
  data = String(dht.readHumidity()) + "," + String(dht.readTemperature());
  return data;
}
```

menu() runs until boolProgramSelect is set to true. Prompts for an input from the user, plan is to reference a manual for a program list.

```
void menu() {
  while (!boolProgramSelect) {
    switch (myLcd.getInt("Choose Program")) {
      case 1: //standardRun()
        if (mystepper.isCalibrated()) {
          boolProgramSelect = true;
          programNum = 1;
          clcd.clear();
          clcd.print("1 Selected");
          delay(1000);
          break;
        } else {
          myLcd.displayInfo("Calibrate Stepper", "Option 3", "", "");
          delay(4000);
          clcd.clear();
          break;
        }
      case 3: //Calibrate stepper
        myLcd.displayInfo("Make sure chamber", "is clear of objects", "calibrating...", "reset to cancel");
        delay(5000);
        mystepper.calibrateStepper();//Calibrates stepper
        mysd.logData("Calibrated Motor", logSensors()); //logs calibration
        myLcd.displayInfo("CALIBRATED", "", "", "");
        delay(2000);
        myLcd.displayInfo("", "", "", "");
        break;
      default:
        clcd.clear();
        clcd.print("Unknown Selection");
        delay(2000);
        break;
```

checkTime() calls confirmTimeScreen and waits for user input. if the user input (*), setTime() is called, and after the time is set it is confirmed again. If (#) is inputted, the time is confirmed, the CSV file is created, boolCheckTime is set true and the program moves on.

```
void checkTime() {
  while (!boolCheckTime) {
    DateTime now = rtc.now();
    confirmTimeScreen(now);
    switch (myLcd.getinput()) {
      case '*':
        setTime();
        break;
      case '#':
        boolCheckTime = true;
        fname = twoChar(now.month()) + twoChar(now.day()) + twoChar(now.hour()) + twoChar(now.minute()) + ".csv";
        mysd.setFileName(fname);
        myLcd.displayInfo("FILE CREATED",fname,"","")
        delay(2000);
        clcd.clear();
        break;
      default:
        break;
    }
  }
}
```

confirmTimeScreen() Displays the known date and time, as well as info for NO or YES.

```
void confirmTimeScreen(DateTime now) {
  myLcd.displayInfo(
    String("Correct date / time?"),
    String(String(now.year()) + "/" + twoChar(now.month()) + "/" + twoChar(now.day())),
    String(twoChar(now.hour()) + ":" + twoChar(now.minute())),
    String("*-NO           #-YES"));
}
```

SetTime() collects year, month, day, hour, and minute info from user.

```
void setTime() {
  clcd.clear();
  yr = myLcd.getInt("YEAR?");
  mth = myLcd.getInt("MONTH?");
  dy = myLcd.getInt("DAY?");
  hr = myLcd.getInt("HOUR? (24 HOUR)");
  mn = myLcd.getInt("MINUTE?");
  rtc.adjust(DateTime(yr, mth, dy, hr, mn, 0));
  delay(100);
}
```

CheckRTCandSD() checks if the RTC and SD are connected and working, and displays the appropriate error if not.

```
void CheckRTCandSD() {
  if (!rtc.begin()) { //Checks to see if RTC is working, if not, freeze the program
    myLdd.displayInfo("ERROR","CHECK RTC","","");
    while (true) {
    }
  }
  if (!mysd.initialise()) {//Checks to see if SD is working, if not, freeze the program
    myLcd.displayInfo("ERROR","CHECK SD CARD","","");
    while (true) {
    }
  }
}
```

twoChar() formats int to string from "5" -> "05" or "15" -> "15".

```
String twoChar(int in) {
  if (in < 10) {
    return "0" + String(in);
  }
  else
    return String(in);
}
```