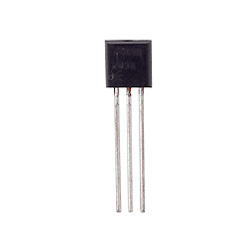
Testing Methods:

**<thing still needed: information on the pressure sensor…I guess we are using it…and problems encountered and what we need to fix. Plus look at the sections, add what is missing, move sections where they are better fit, and see what else can be changed. Thanks!**>

**(Intro)**

Our design had modifications on the sensors that were supplied. At the primary stages, we were faced with the choice of using LM34 temperature sensors. These allowed us to get the temperature from separate sensors with each reading on a separate pin. The benefit of this sensor was that the implementation with the program was fairly simple. The downside of using this sensor was that each sensor needed its own pin on the microcontroller and would take up too much space on the final product.

Below is a picture of the sensor:

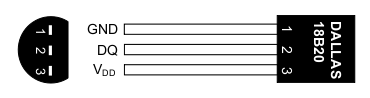
<http://www.parallax.com/Store/Sensors/TemperatureHumidity/tabid/174/CategoryID/49/List/0/SortField/0/Level/a/ProductID/87/Default.aspx>

To solve the problem of the LM34 sensors taking up the pins on the board we switched to the Dallas Sensors. The Dallas Temperature Sensors have the ability to be connected in a parasite. This meaning that one sensor is the master sensor and the others are connected to it in series making them the slave sensors. This is made possible thanks to the fact that each temperature sensor has its own unique address that is read from each sensor with its own data read out. While these Dallas Sensors freed up the pins and allowed us to use n-number of sensors the code that went along with it proved to be rather cumbersome.

The Dallas Sensors only require a common ground and one pin on the arduino microcontroller. Due to the fact that the Dallas Temperature sensors are only connected to one pin, our code needed to not only read in the data from the sensor but also the address associated with the sensor to discriminate between the different sensors. Schematic for Dallas Sensor connection is displayed below in Figure\_\_ (**or here? Or somewhere nearby?...halp..**).

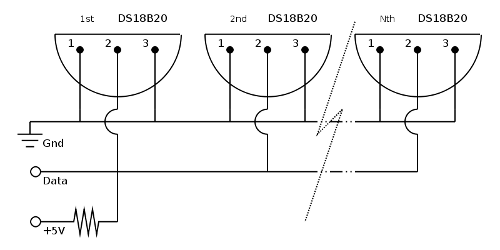
To begin, here is the actual sensor and its inner workings:

**<insert picture of the sensor & explain the pins>**

<http://www.google.com/imgres?imgurl=http://www.retorte.ch/public/arduino_temperature_sensor/maxim_ds18b20.png&imgrefurl=http://www.retorte.ch/tools/experiments_and_snippets/arduino_temperature_sensor/&usg=__Q1fEmSGsdVIcRMWH8mHk4PdnY84=&h=98&w=370&sz=8&hl=en&start=40&zoom=1&tbnid=cv6QwMZZzEHpJM:&tbnh=51&tbnw=192&ei=Y1a6TcDcDczegQfevezODw&prev=/search%3Fq%3DDallas%2BTemperature%2Bsensor%26hl%3Den%26rlz%3D1C1SKPC_enUS410US410%26biw%3D1064%26bih%3D748%26site%3Dsearch%26tbm%3Disch0%2C1562&um=1&itbs=1&iact=hc&vpx=317&vpy=154&dur=413&hovh=52&hovw=196&tx=94&ty=23&page=3&ndsp=20&ved=1t:429,r:11,s:40&biw=1064&bih=748>

The Dallas Temperature Sensor, specifically the DS18B20 is in a casing very similar to that of the LM34, so physically there is no difference on how they sit on the device.

**<insert parasite schematic & explain a little more in detail the interconnections (like which pin does what and delay timinigs)>**

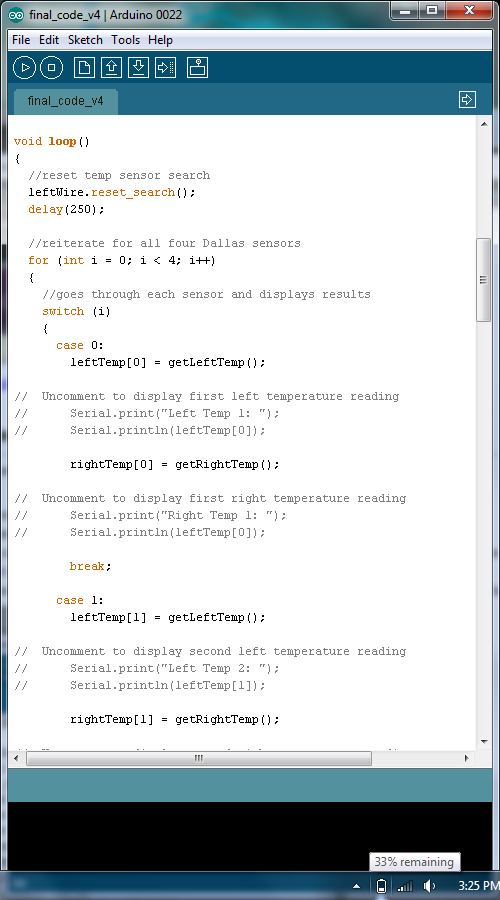
<http://www.google.com/imgres?imgurl=http://www.strangeparty.com/wordpress/uploads/2010/12/DS18B20.png&imgrefurl=http://www.strangeparty.com/2010/12/13/arduino-1-wire-temperature-sensors/&usg=__lTAYs0efYz5U7k4oqhJF6Dn9ajQ=&h=250&w=500&sz=15&hl=en&start=0&zoom=1&tbnid=tCup0qEORWL4lM:&tbnh=122&tbnw=244&ei=6Fe6TdOjEubs0gHFt4HGAQ&prev=/search%3Fq%3DDallas%2BTemperature%2Bsensor%2Bparasite%2Bconnection%26um%3D1%26hl%3Den%26rlz%3D1C1SKPC_enUS410US410%26biw%3D1064%26bih%3D748%26tbm%3Disch&um=1&itbs=1&iact=hc&vpx=406&vpy=117&dur=4042&hovh=159&hovw=318&tx=168&ty=94&page=1&ndsp=20&ved=1t:429,r:2,s:0>

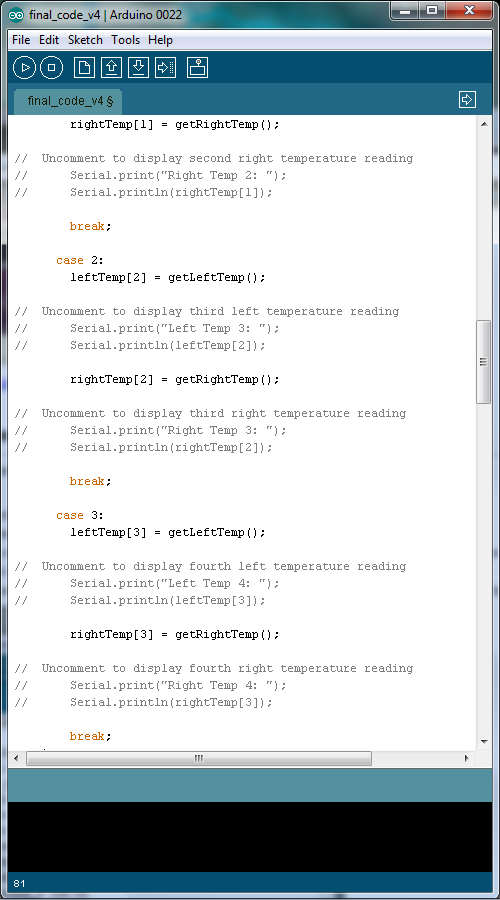
Due to the complexity of the Dallas sensors they require a library to reference in the program to operate properly.

Using the Dallas Temperature library for Arduino, we were able to obtain information from each sensor (data and address). The address is obtained and stored to call each sensor individually later. When each sensor is called a data read out is obtained and decoded to output a Celsius temperature reading.

<**here we explain the code?**>

Down below in Figure\_\_\_ is a screenshot of our main loop. This loop goes through each temperature sensor and stores the temperature reading. This is done through calling a ’getTemperature()’ method that obtains the address of the sensor as well as the data reading, which is also converted to a readable temperature. Based on the temperature reading a pulse wave output is sent to the mosfet that controls the voltage to the TEC. To maintain a safe temperature interface between the TEC and the user a ‘TECCtrl()’ method is called. This method is responsible for giving more voltage to the mosfet when the interface temperature is too warm. As well as lowering the voltage when the interface temperature is too low.



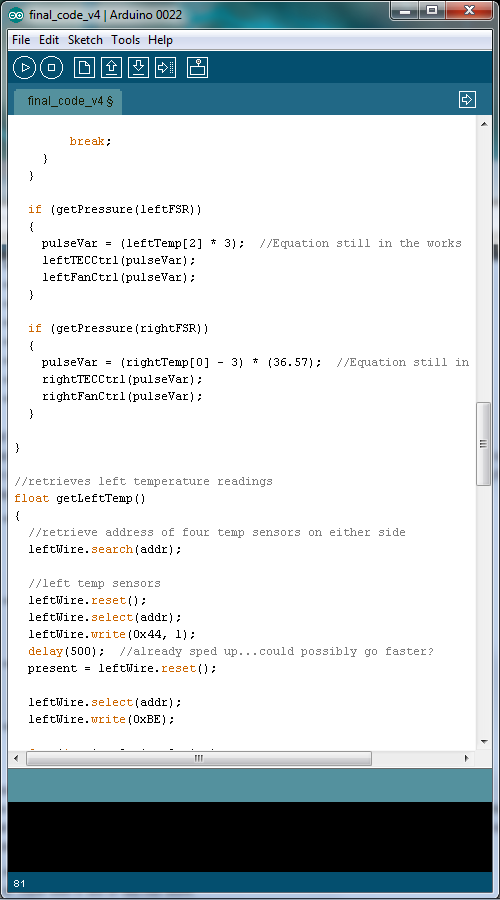


<**Mosfet controlling section**>

The mosfet switches are turning off and on at a rate which we set through a calculation which relates the temperature of the skin and the min and max at which the mosfet should be running. As an example, at minimum temperature of 3 degrees celcius (**or whatever it is..idk lol**) the mosfet should not be turned on. Yet, at the maximum we want the mosfet to run which is 50%, it must reach this only if it is at the maximum allowed temperature. If any higher, then the mosfet switching must be killed and all processes must be stopped.

Below is a section of the code which controls the mosfet with our predetermined equations along with an on/off button which is coded in as the pressure sensor. What is supposed to happen here is first read the pressure sensor. Since the device is only supposed to work when pressure is applied to it, that is it is actually put onto an individual, we have to check the pressure sensor. If pressure, then we are allowed to have the mosfet switches do their work and allow power to be supplied to the TECs. Then, during a scenario in which it is applied to an individual, we then alter the rate of switching to go along with the temperature of the skin.

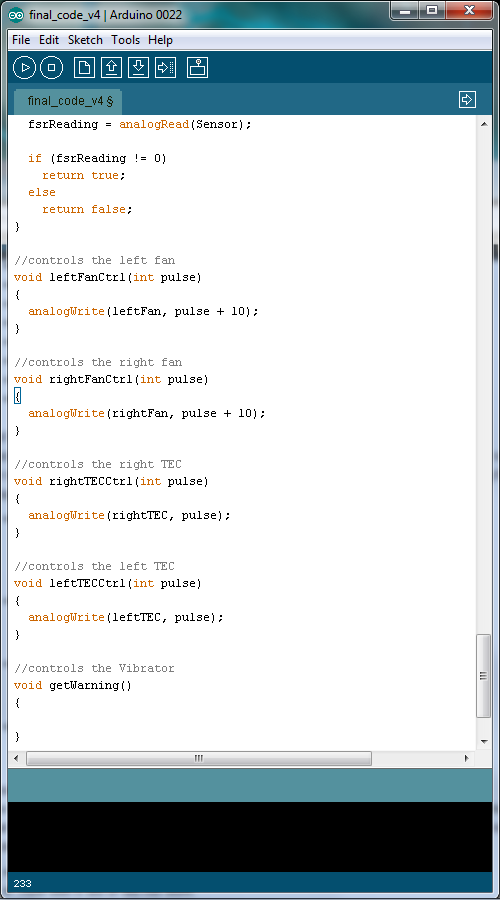
<**fix this picture with the correct code once it is finished!!!!!!!!!!!!!**>



<**Fan section and TEC?**

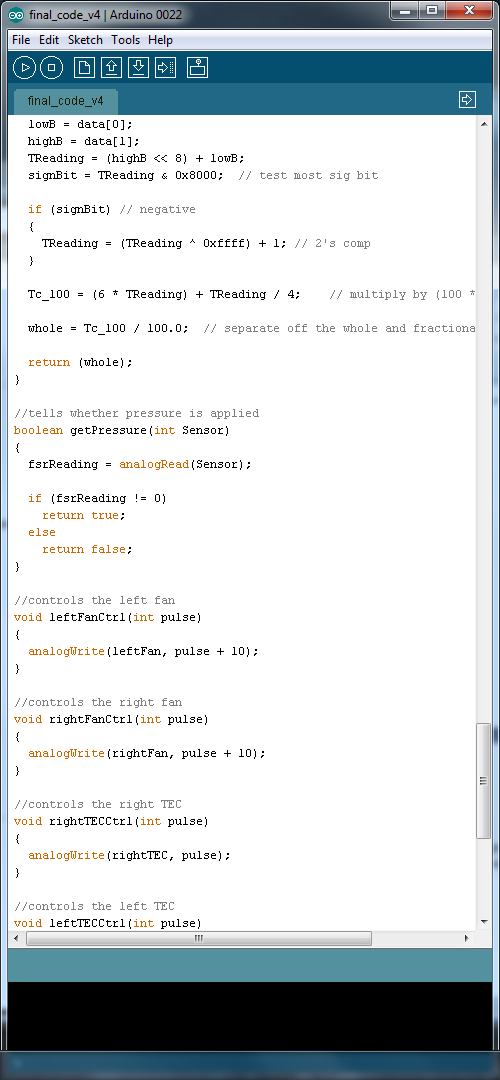
Finally, we have our fans that work directly on the TECs. The operation for a fan is very simple, ON during use and remain on for a few more seconds after a TEC is turned off to expel excess heat, or OFF if the whole system is not turned on. Below is a snippet of the code based upon the fan.

<**this code looks unfinished….we need to get this done and update this part>**



(**pressure section**)

The best on/off switch is whether the device is attached to skin, therefore we have a pressure sensor near the bottom of the device. This is programmed into the device as a master on/off switch and a safety feature. In order to implement this, we work with the following code:



The application of the pressure sensor as an on off switch is the simplest way to utilize the code required for a pressure sensor. In our appendix in part (**IDK WHAT Part of the appendix our reference code is in….)** we see the difference between that code and what is required for our concept.

**<wrap up? I’ll leave it up to you guys.>**

With these individual snippets of code, once placed together, they form a piece of code that is unique to our design goal.