Connor Hoang

ID: 2829138

EECS 113

12 June 2024

Final Project Report

# Code

## 1. Setting up

This section of code includes the imported files I used, the pin assignments and their input and output options, and global variables that I use later in the project. The imported files were brought from the Freenove Kit’s file of example Python codes that utilizes the kit’s peripherals.

| #Connor Hoang  #ID: 2829138  import RPi.GPIO as GPIO  import Freenove\_DHT as DHT  import threading  import time  import requests  from datetime import datetime  from PCF8574 import PCF8574\_GPIO  from Adafruit\_LCD1602 import Adafruit\_CharLCD  from time import sleep, strftime  done = False #to finish program  #pin assignments  #PIR  PIRsense = 17  PIRLed = 12  #DHT  DHTsense = 27  ACled = 18  Heatled = 23  incButt = 20  decButt = 21  doorButt = 16  #variables  desiredTemp = 80  currTemp = 0  weatherInd = 81  PIRon = False  opening = False #false = closed, true = open  GPIO.setmode(GPIO.BCM)  GPIO.setwarnings(False)  #setup buttons and pins  def setup():  #PIR  GPIO.setup(PIRLed, GPIO.OUT)  GPIO.setup(PIRsense, GPIO.IN)  #DHT  GPIO.setup(DHTsense, GPIO.IN)  GPIO.setup(ACled, GPIO.OUT)  GPIO.setup(Heatled, GPIO.OUT)  GPIO.setup(incButt, GPIO.IN)  GPIO.setup(decButt, GPIO.IN)    #door  GPIO.setup(doorButt, GPIO.IN)    #start all LEDs off  GPIO.output(PIRLed, GPIO.LOW)  GPIO.output(ACled, GPIO.LOW)  GPIO.output(Heatled, GPIO.LOW) |
| --- |

## 2. PIR Infrared Motion Detector

Starting with the first task was the PIR infrared motion detector. This function when initially run will stay running. It’ll first check if the readings from the PIR are high or low. If it’s low, meaning there’s zero detection, then it will start a count down from 10 to 0. After 10 seconds of no motion detection, the green LED will turn off. If it detects any motion during the countdown then it will stop counting down.

| def PIRrun():  global PIRon  alrOff = True  while True:  if GPIO.input(PIRsense) == GPIO.LOW: # if no PIR readings, then wait 10 seconds before turning off  count = 10  while count > 0:  if GPIO.input(PIRsense) == GPIO.LOW:  count -= 1  if not alrOff:  print(getTime(), 'LIGHTS OFF IN ', count)  sleep(1)  else:  break    if count <= 0:  GPIO.output(PIRLed, GPIO.LOW)  if not alrOff: #ensures no print repeat  print(getTime(), "LIGHTS OFF")  PIRon = False  alrOff = True  else:  PIRon = False  alrOff = True  else: # if there are PIR readings, keep/turn lights on  GPIO.output(PIRLed, GPIO.HIGH)  if alrOff: #ensures no print repeat  print(getTime(), "LIGHTS ON")  PIRon = True  alrOff = False  else:  PIRon = True  alrOff = False |
| --- |

## 3. Hygrothermograph DHT

The next device was the Hygrothermograph DHT which was used to make an HVAC system. Initially, we grab some global variables we need to keep track of as they update and then create a DHT class object. In this code there were troubles running into the CIMIS issues, so instead I hardcoded the irvine humidity level which was everaged out to be 77 on 6/12/24. Then, like the PRI, it’ll enter the while loop and it keep running until the program is terminated. Once in the loop, it’ll first check the data read from the DHT to ensure that it’s working correctly. After checking that the device and its readings are normal, we’ll check if there’s an opening from a door or window. If there’s no opening, then the DHT will collect data and calculate the current temperature in the room and the weather index. If there is an opening, then we’ll turn off the LEDs and turn off the HVAC system.

| def DHTrun():  global opening  global weatherInd  global currTemp  alrPrinted = False  dht = DHT.DHT(DHTsense) #create DHT class object  # cannot get CIMIS to work  #resp = requests.get('http://et.water.ca.gov/api/data?appKey=(43ac6ff3-36d5-44b7-8bec-4434abbbe3bd)&targets=75&startDate=2024-05-27&endDate=2024-05-27&dataItems=hly-rel-hum&unitOfMeasure=E')  #post = resp.float()  irvHumidity = 77 # static value    while(True):  for i in range(0,15):  chk = dht.readDHT11() #read DHT11  if (chk == dht.DHTLIB\_OK): #is DHT11 normal  #print("\t\t\t|DHT11,OK!")  #print("\t\t\t|Dr : ", opening)  break  sleep(0.3)    if not opening: #If there's no openings, then HVAC on and DHT working  #fire ex  #weatherInd = 100    currTemp = (int)((dht.temperature \* 1.8) + 32)  #print("DHT Temp : %dF"%(dhtTemp))  weatherInd = currTemp + (0.05\*irvHumidity)  #print("Weather Index : %dF"%(weatherInd))    alrPrinted = False  else: # if opening, then HVAC off  GPIO.output(ACled, GPIO.LOW)  GPIO.output(Heatled, GPIO.LOW)  if not alrPrinted:  print(getTime(), "HVAC OFF")  alrPrinted = True |
| --- |

## 4. LCD

The main thread running was this LCD function. It was made into the main thread function because, in my opinion, if there’s no where to display the data or information to the user, then there’s no point in using up energy to collect data for no one. This LCD function controls the LCD and displays the four main components: “DesiredTemp/CurrentTemp,” “Door: Open/Close,” “HVAC: AC/OFF/HEATER,” and PIR “LIGHT: ON/OFF.” All of the collected data that’s sent to be global is accessed and then the LCD decides what the output will show from there. If a pop-message needs to be displayed, then it will call one of the functions so the remaining display calls don’t disrupt the pop-up. I used multiple “Display \_\_\_ Already” variables to make sure the pop-up messages to the LCD and terminal weren’t being constantly spammed. In addition, the LEDs will be controlled here, because if the decisions were already being made I figured it would be more optimal to have the outputs come from one thread instead of multiple threads that made the same decisions through if-statements.

| def LCDrun():  global desiredTemp  global weatherInd  global opening  global PIRon  global currTemp    #makes sure it's not continuously printing in terminal  dispACAlr = False  dispHeatAlr = False  dispOffAlr = False  dispFireAlr = False  dispOAlr = False    mcp.output(3,1)  lcd.begin(16,5)  while not done:  #DesiredTemp/CurrTemp  lcd.setCursor(0,0)  sleep(1)  lcd.message("%d/%d"%(desiredTemp, currTemp))    lcd.setCursor(6,0)  lcd.message(getLCDTime())  #Door/Window open/closed  lcd.setCursor(12,0)  if opening:  if not dispOAlr:  print(getTime(), "DOOR OPEN")  dispO()  dispOAlr = True  lcd.message("Dr:O")  else:  if dispOAlr:  print(getTime(), "DOOR CLOSED")  dispO()  dispOAlr = False  lcd.message("Dr:C")      #HVAC Msgs  if opening:  lcd.setCursor(0,1)  lcd.message("H:OFF ")  GPIO.output(ACled, GPIO.LOW) # no leds on  GPIO.output(Heatled, GPIO.LOW)  else:  if weatherInd > 95: #fire alarm  opening = True  if not dispFireAlr:  dispFire()  dispFireAlr = True  dispOffAlr = False  dispACAlr = False  dispHeatAlr = False  else:  if weatherInd >= desiredTemp +3: # too hot -> AC on  if not dispACAlr:  dispAC()  print(getTime(), "HVAC AC")  dispACAlr = True  dispHeatAlr = False  dispOffAlr = False  lcd.setCursor(0,1)  lcd.message("H:AC ")  else:  if weatherInd <= desiredTemp-3: # too cold, heat on  if not dispHeatAlr:  dispHeat()  print(getTime(), "HVAC HEATER")  dispHeatAlr = True  dispACAlr = False  dispOffAlr = False  lcd.setCursor(0,1)  lcd.message("H:HEAT")  GPIO.output(Heatled, GPIO.HIGH) #Heater on  GPIO.output(ACled, GPIO.LOW)  else:  if dispOffAlr:  dispHOff()  dispOffAlr = True  dispACAlr = False  dispHeatAlr = False  lcd.setCursor(0,1)  lcd.message("H:OFF ")  GPIO.output(ACled, GPIO.LOW) # no leds on  GPIO.output(Heatled, GPIO.LOW)    #PIR  if PIRon:  lcd.setCursor(11,1)  lcd.message(" L:ON")  else:  lcd.setCursor(11,1)  lcd.message("L:OFF") |
| --- |

## 5. Display Pop-ups

These are functions that don’t run unless commanded to and their purpose is to display the pop up messages: “Fire - Evacuate,” “AC on,” “Heater on,” “HVAC off,” and “door/window is open/closed.” These are separate functions outside of the threaded ones because these will pause the LCD thread, the main display, which gives these time to display without needing to rely on the timing of the other displaying messages. In the dispFire(), when using a hard-coded variable, the loop that decreases the weather index by 1 each time will keep the program running. If it were to stay above 95, then the program would not progress forward.

| def dispFire(): # Display on term and LCD of fire  global weatherInd  lcd.clear()  lcd.setCursor(0,0)  lcd.message("FIRE - EVACUATE")  lcd.setCursor(0,1)  lcd.message("DR/WINDWS OPEN")  sleep(3)  dispHOff()  dispO()  for i in range(0,3): # Flash lights  GPIO.output(Heatled, GPIO.HIGH) #Heater on  sleep(.5)  GPIO.output(ACled, GPIO.HIGH) #AC on  sleep(.5)  GPIO.output(Heatled, GPIO.LOW) #Heater off  sleep(.5)  GPIO.output(ACled, GPIO.LOW) #AC off      while weatherInd >= 95:  weatherInd = weatherInd - 1  print(getTime()," FIRE TEMP: ", weatherInd, "F")  sleep(1)    sleep(0.5)  def dispAC():  lcd.clear()  lcd.setCursor(4,0)  lcd.message("AC is on")  sleep(3)  GPIO.output(ACled, GPIO.HIGH) #AC/blue led on  GPIO.output(Heatled, GPIO.LOW) #Heat/red led on  lcd.clear()  sleep(0.5)  def dispHeat():  lcd.clear()  lcd.setCursor(2,0)  lcd.message("Heater is on")  sleep(3)  GPIO.output(ACled, GPIO.LOW) #AC/blue led off  GPIO.output(Heatled, GPIO.HIGH) #Heat/red led on  lcd.clear()  sleep(0.5)  def dispHOff():  lcd.clear()  lcd.setCursor(2,0)  lcd.message("HVAC is off")  sleep(3)  GPIO.output(ACled, GPIO.LOW) # HVAC off, both leds off  GPIO.output(Heatled, GPIO.LOW)  lcd.clear()    def dispO():  lcd.clear()    if opening:  lcd.setCursor(3,0)  lcd.message("door/window")  lcd.setCursor(5,1)  lcd.message("open!")  else:  lcd.setCursor(3,0)  lcd.message("door/window")  lcd.setCursor(5,1)  lcd.message("closed!")    sleep(3)  lcd.clear()  sleep(0.5) |
| --- |

## 6. Miscellaneous

These are functions that are short, self explanatory, and not used very often. The getTime is used for the 5% extra credit which implements an internal clock system which is displayed in the terminal (see Images at the end). I’ve also included an LCD time function to display only the hour and minutes so it can fit on the LCD display. The destroy function on the other hand is used after all threads are stopped and so the function will clean up the LCD and turn off the LEDs.

| def getTime():  return datetime.now().strftime('%H:%M:%S')  def getLCDTime():  return datetime.now().strftime('%H:%M')  def destroy():  lcd.clear()  GPIO.cleanup() |
| --- |

## 7. Buttons

This is a minor thread that simply waits for the buttons to be pressed. Once pressed it’ll update the global variables accordingly. I’ve included sleep delays to ensure a button doesn’t take in multiple inputs from one click.

| def inc(): # increase desired temp  global desiredTemp  sleep(0.2)  if desiredTemp < 95:  desiredTemp += 1  print("Desired Temp : %dF"%(desiredTemp))  def dec(): # decrease desired temp  global desiredTemp  sleep(0.2)  if desiredTemp > 65:  desiredTemp -= 1  print("Desired Temp : %dF"%(desiredTemp))  def toggleDr(): # toggle door/window opening  global opening  if opening:  opening = False  else:  opening = True  #print("Dr changed: ", opening)    def buttons(): # when buttons are clicked, trigger variable changes  while True:  if GPIO.input(doorButt)==GPIO.LOW:  toggleDr()  sleep(.5)    if GPIO.input(incButt)==GPIO.LOW:  inc()  sleep(0.2)  else:  if GPIO.input(decButt)==GPIO.LOW:  dec()  sleep(0.2) |
| --- |

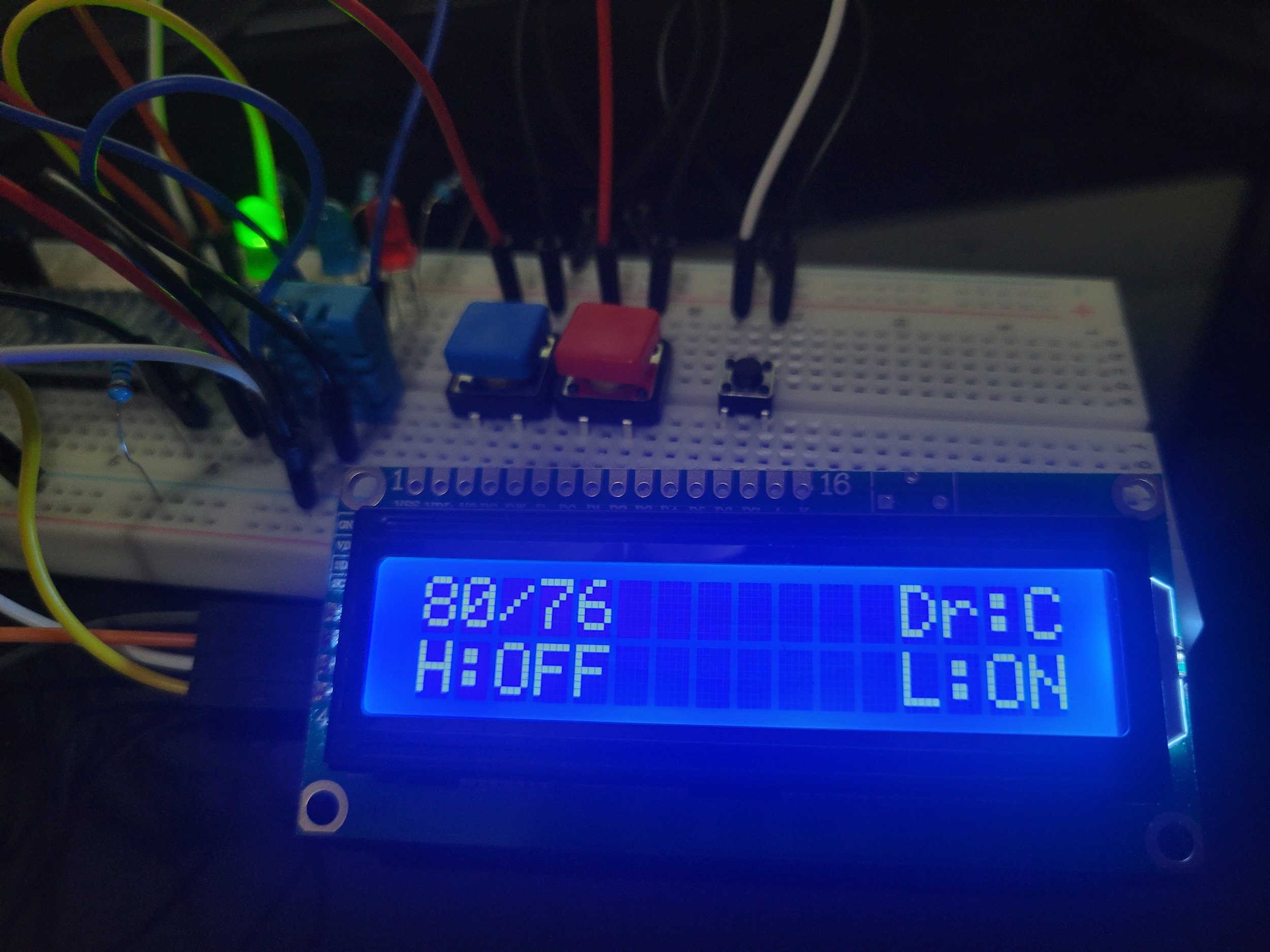
## 8. Main

This is the “main” function in python which is where the code is read from. It’ll start and setup the buttons, variables, and then the LCD. After everything is setup, the “try” will run the thread of functions below. Buttons, PIR, and DHT are all run as daemon threads, meaning that if no other thread is running then it will stop. This makes them rely on the main LCD thread, because once the LCD stops working, the rest of the functions will stop as well. The exception includes an interrupt such as ctrl+c. If this exception is caught, then the program will stop.

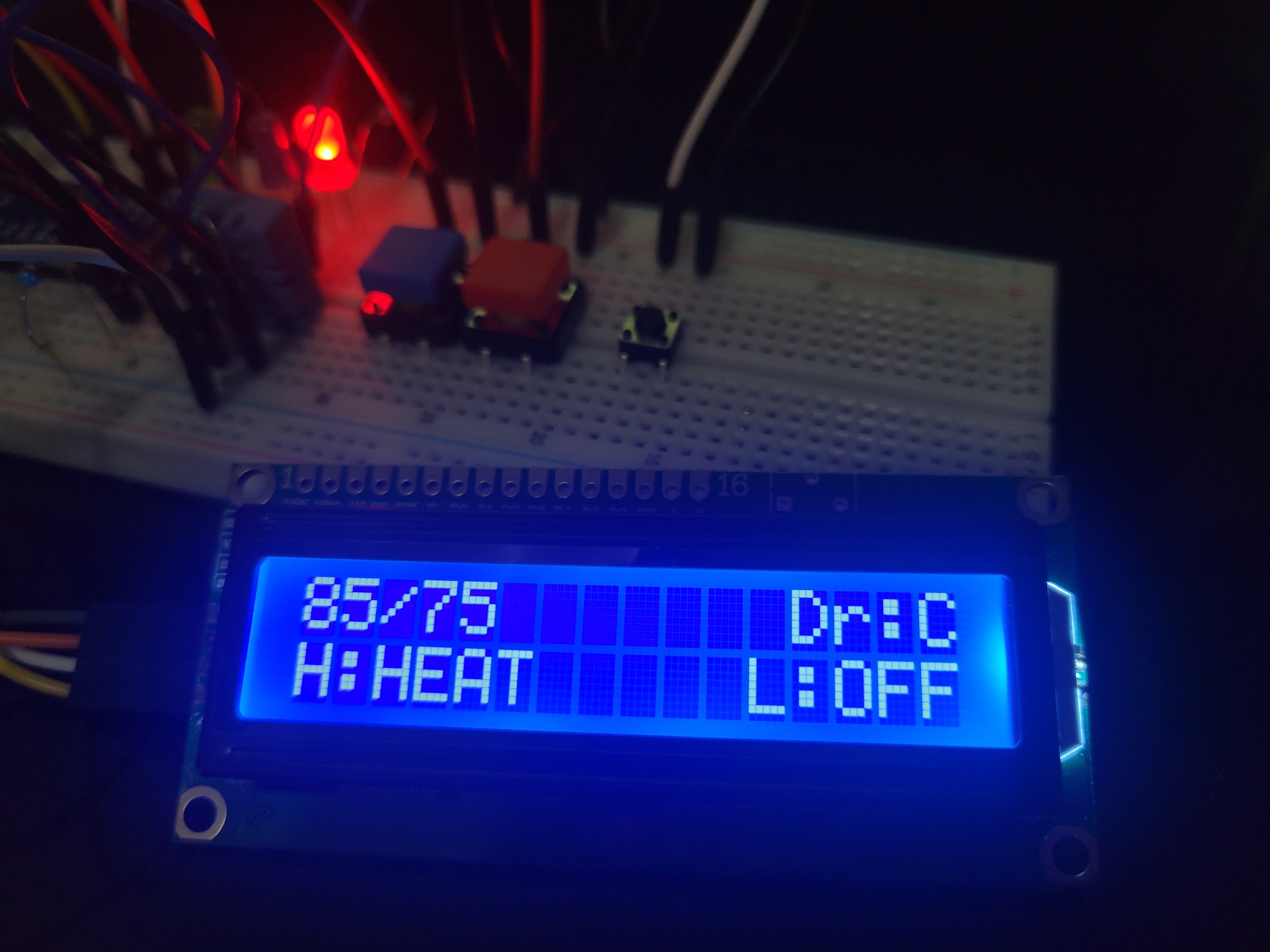
| if \_\_name\_\_ == '\_\_main\_\_':  print('Program Starting...')  setup()    PCF8574\_address = 0x27 # I2C address of the PCF8574 chip.  PCF8574A\_address = 0x3F # I2C address of the PCF8574A chip.  try:  mcp = PCF8574\_GPIO(PCF8574\_address)  except:  try:  mcp = PCF8574\_GPIO(PCF8574A\_address)  except:  print ('I2C Address Error !')  exit(1)  lcd = Adafruit\_CharLCD(pin\_rs=0, pin\_e=2, pins\_db=[4,5,6,7], GPIO=mcp)    try:  buttons = threading.Thread(target=buttons, daemon = True)  PIRthread = threading.Thread(target=PIRrun, daemon = True)  DHTthread = threading.Thread(target=DHTrun, daemon = True)  LCDthread = threading.Thread(target=LCDrun)    buttons.start()  PIRthread.start()  DHTthread.start()  LCDthread.start()    buttons.join()  PIRthread.join()  DHTthread.join()  LCDthread.join()    except KeyboardInterrupt:  done = True  destroy() |
| --- |

# Images

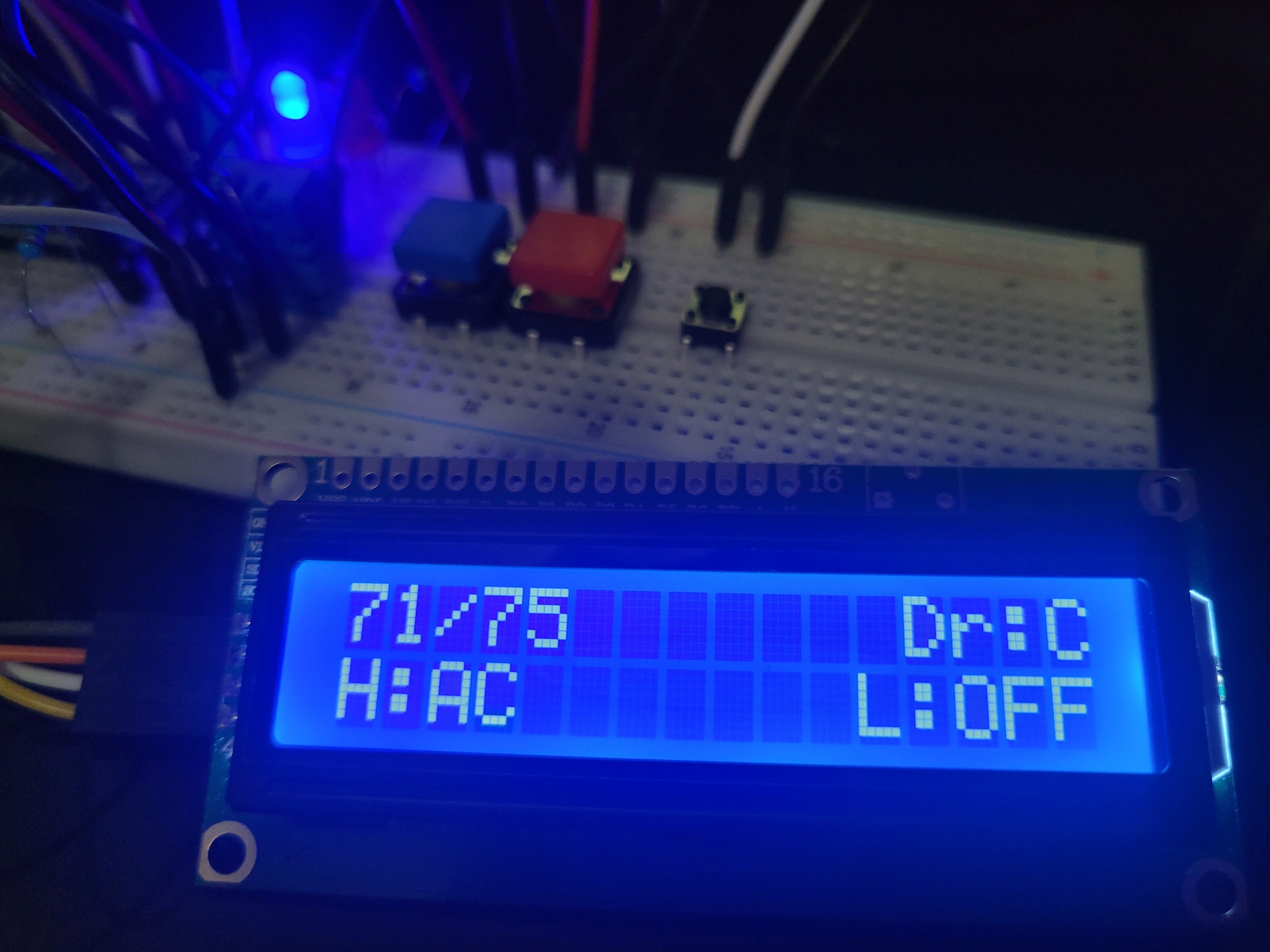
## Initial



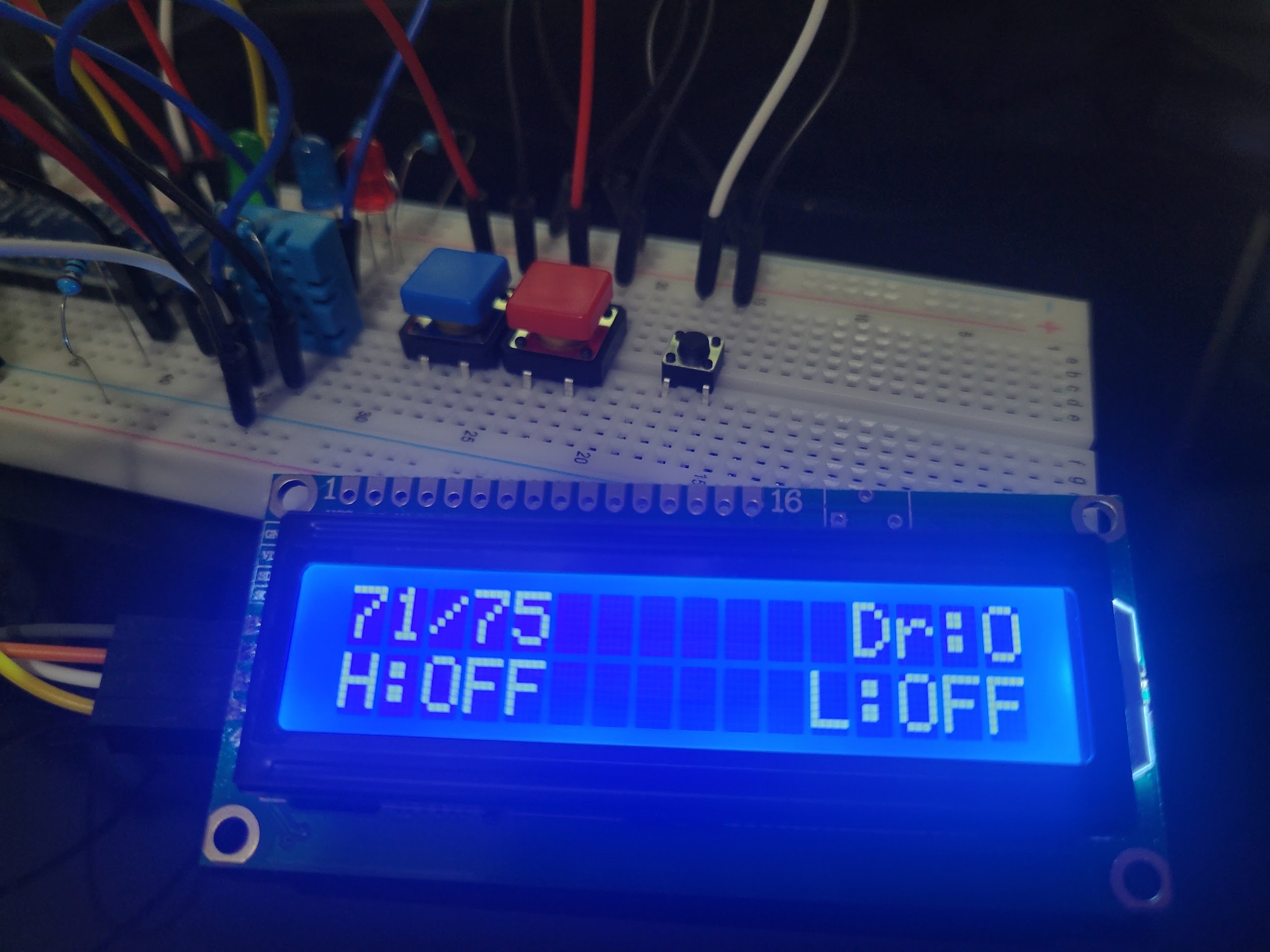
## Heater on



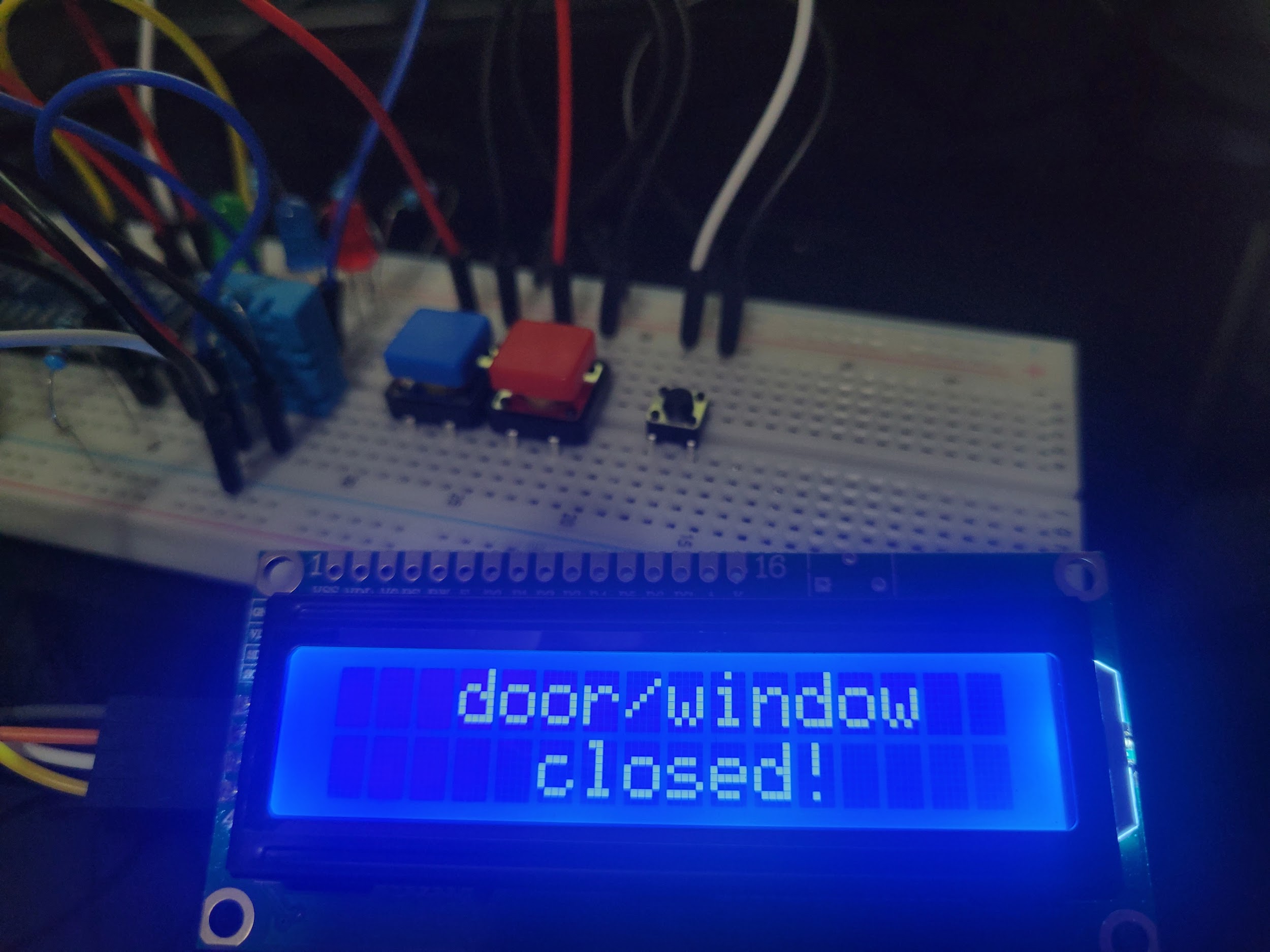
## AC on



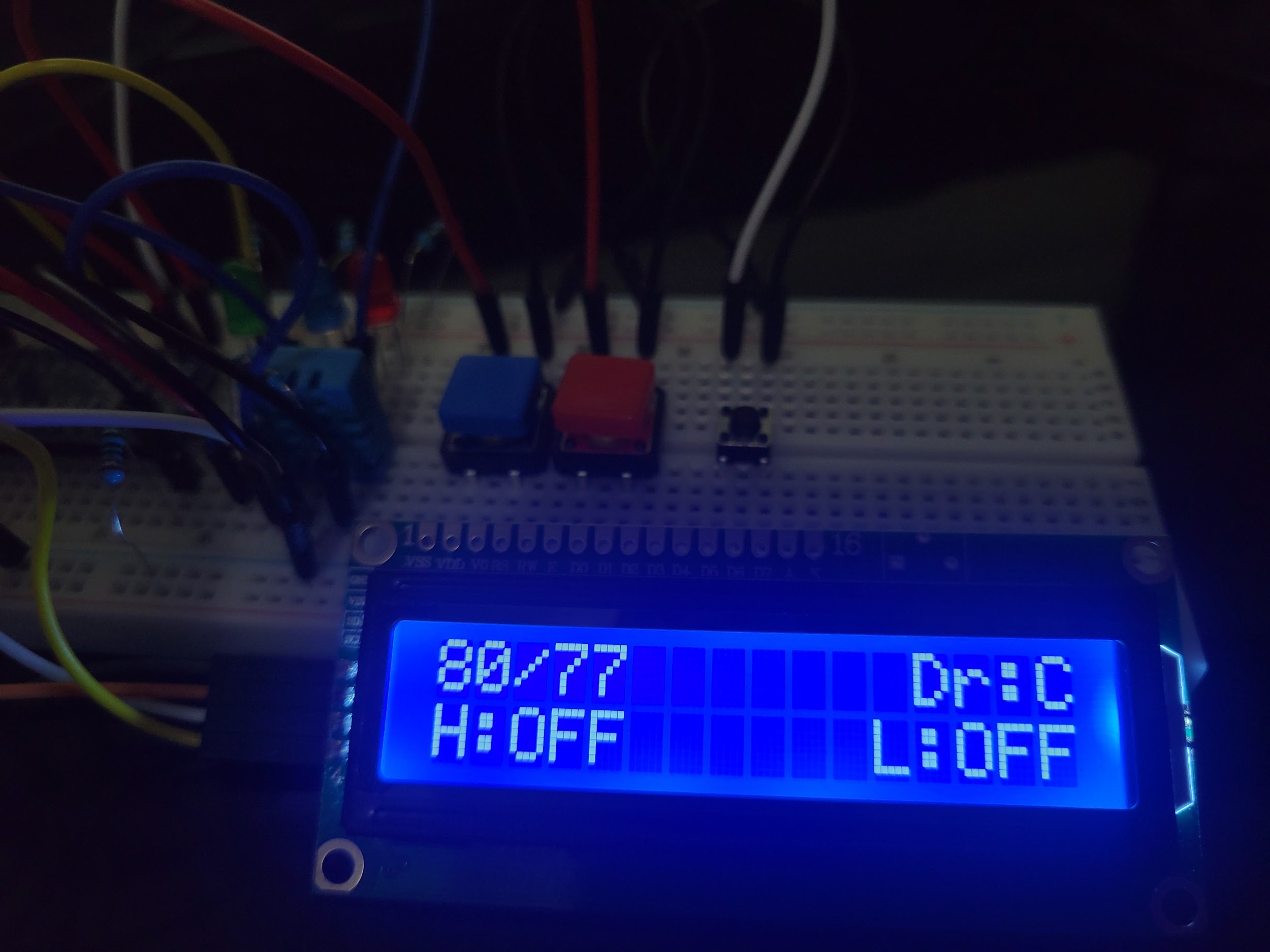
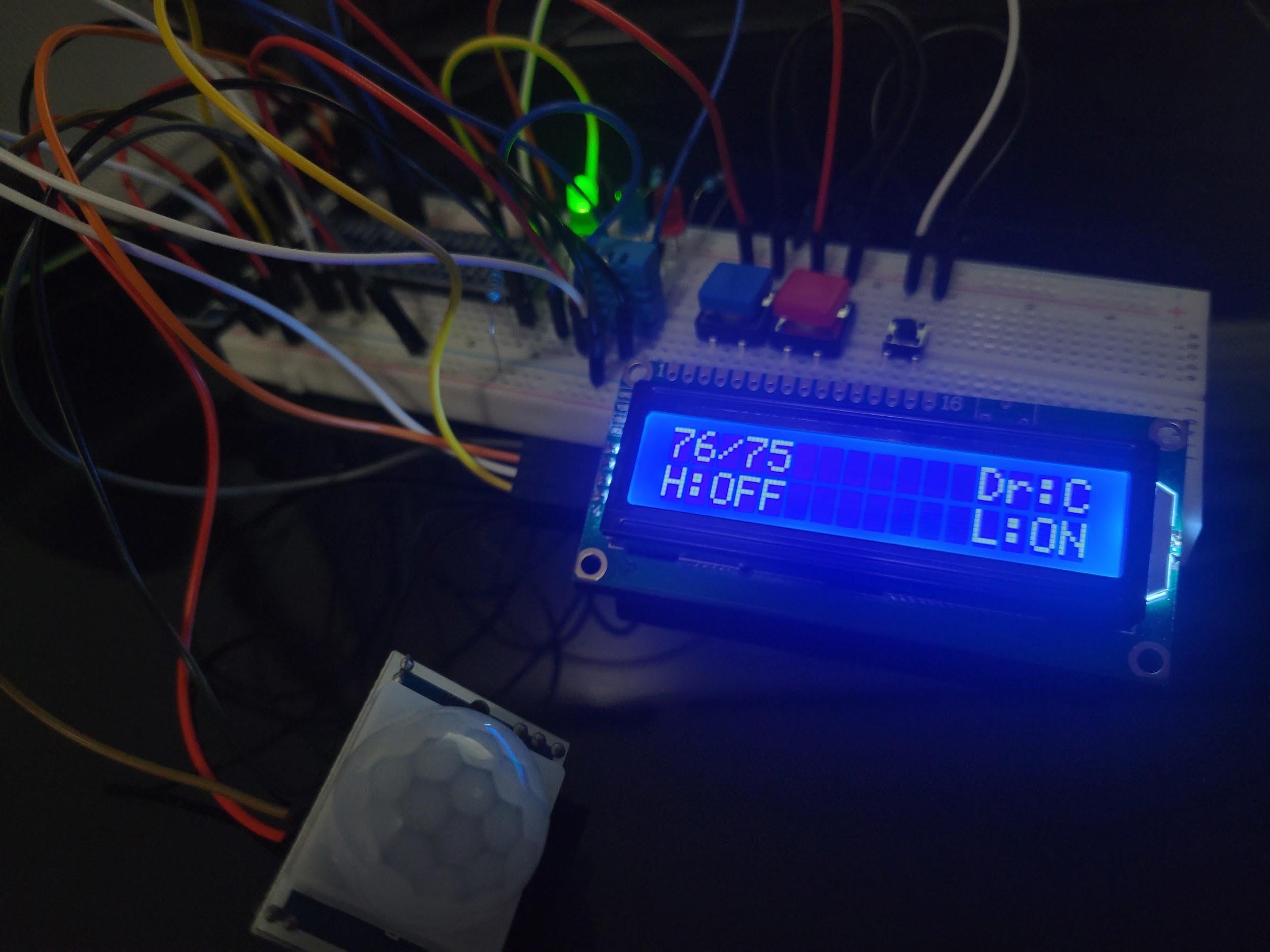
## Door/window open



## Door/window closed



## PIR lights on

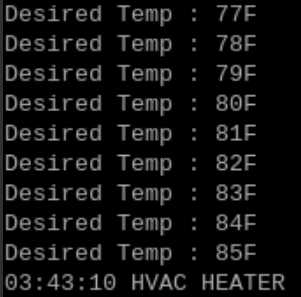


## LCD Display including Time (Hours:Minutes)

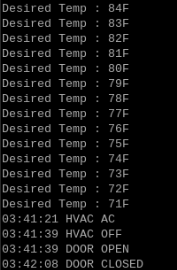


# Terminal output including Timestamps

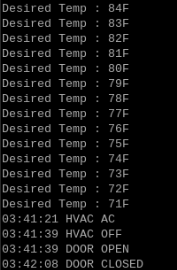
## Heater on



## AC on



## Door/Window open/closed



## PIR lights off/on

