```cpp
/*
 * This is the .cpp file for the PMS7003 sensor
 * This code was written exclusively by MECH 45X Team 26
 */
#include "PM.h"

PM_7003::PM_7003() {
    current_byte = 0;
    packetdata.frame_length = MAX_FRAME_LENGTH;
    frame_length = MAX_FRAME_LENGTH;
}

PM_7003::~PM_7003() {
}

int PM_7003::getpm(void) {
    return pm_avgpm2_5;
}

bool PM_7003::run_PM_sensor(void) {
    /*
     * run the PM sensor
     * Start serial connection
     *
     * drain_serial() and read_sensor() until enough values have been read
     * to take the average
     */
    Serial1.begin(9600);
    read_count = 1;
    done_reading = false;
    frame_sync_count = 0;
    pm_avgpm2_5 = 0;
    while(!done_reading && frame_sync_count < MAX_FRAME_SYNC_COUNT) {
        drain_serial();
        delay(500);
        read_sensor();
    }

    Serial1.end();

    if(done_reading) {
        Serial.println("----------------------------");
        Serial.println("Done reading from PM sensor");
        Serial.println("----------------------------");
        Serial.println(" ");
        return true;
    }
    else if(!done_reading && frame_sync_count >= MAX_FRAME_SYNC_COUNT){return false;}
}

void PM_7003::drain_serial(void) {
    /*
    * Drains serial buffer if there are more than 32 entries
    * Reads entries to drain serial buffer
    */
    if (Serial1.available() > 32) {
        drain = Serial1.available();
        Serial.println("-- Draining buffer: ");
        Serial.println(Serial1.available(), DEC);
        for (int drain_index = drain; drain_index > 0; drain_index--) {Serial1.read();}
    }
}

void PM_7003::frame_sync(void) {
  /*
   * syncs frames for PM sensor
   * checks that frames are being read in correct order
   * exits when it confirms that frames are being read correctly
  */
```

```
    sync_state = false;
    frame_count = 0;
    byte_sum = 0;

    while (!sync_state && frame_sync_count < MAX_FRAME_SYNC_COUNT){
        current_byte = Serial1.read();

        if(current_byte == FIRST_BYTE && frame_count == 0) {
            frame_buffer[frame_count] = current_byte;
            packetdata.start_frame[0] = current_byte;
            byte_sum = current_byte;
            frame_count = 1;
        }
        else if(current_byte == SECOND_BYTE && frame_count == 1){
            frame_buffer[frame_count] = current_byte;
            packetdata.start_frame[1] = current_byte;
            byte_sum = byte_sum + current_byte;
            frame_count = 2;
            sync_state = true;
        }
        else{
            frame_sync_count++;
            Serial.println("frame is syncing");
            Serial.print("Current character: ");
            Serial.println(current_byte, HEX);
            Serial.print("frame count: ");
            Serial.println(frame_sync_count);
            delay(500);

            if(frame_sync_count >= MAX_FRAME_SYNC_COUNT) {
                Serial.println("------------------------");
                Serial.println("Max frame count exceeded");
                Serial.println("------------------------");
            }

        }
    }
}

void PM_7003::read_sensor(void) {
    /*
     * Sync the frames
     * read bytes and fill frame_buffer
     * use data_switch to calculate different parameters
     * print_messages once all values have been read.
     * done_reading = true if enough values have been read
     */
    frame_sync();

    while(sync_state == true && Serial1.available() > 0) {
        current_byte = Serial1.read();
        frame_buffer[frame_count] = current_byte;
        byte_sum = byte_sum + current_byte;
        frame_count++;
        uint16_t current_data = frame_buffer[frame_count-1]+(frame_buffer[frame_count-2
        ]<<8);
        data_switch(current_data);

        if (frame_count >= frame_length && read_count <= MAX_READ_COUNT) {
            print_messages();
            pm_avgpm2_5 = pm_avgpm2_5 + pm2_5;
            read_count++;
            break;
        }
    }

    if (read_count > MAX_READ_COUNT) {
        pm_avgpm2_5 = exp((pm_avgpm2_5/MAX_READ_COUNT + 109314)/15990)*10000;
        done_reading = true;

    }
}

void PM_7003::data_switch(uint16_t current_data) {
    /*
     * data_switch uses current data and frame_count
     * to assign values to parameters
     */
    switch (frame_count) {
    case 4:
        packetdata.frame_length = current_data;
        frame_length = current_data + frame_count;
        break;
    case 6:
        packetdata.concPM1_0_factory = current_data;
        break;
    case 8:
        packetdata.concPM2_5_factory = current_data;
        break;
    case 10:
        packetdata.concPM10_0_factory = current_data;
        break;
    case 12:
        packetdata.concPM1_0_ambient = current_data;
        break;
    case 14:
        packetdata.concPM2_5_ambient = current_data;
        break;
    case 16:
        packetdata.concPM10_0_ambient = current_data;
        break;
    case 18:
        packetdata.countPM0_3um = current_data;
        break;
    case 20:
        packetdata.countPM0_5um = current_data;
        break;
    case 22:
        packetdata.countPM1_0um = current_data;
        break;
    case 24:
        packetdata.countPM2_5um = current_data;
        break;
    case 26:
        packetdata.countPM5_0um = current_data;
        break;
    case 28:
        packetdata.countPM10_0um = current_data;
        break;
    case 29:
        current_data = frame_buffer[frame_count-1];
        packetdata.version = current_data;
      break;
    case 30:
        current_data = frame_buffer[frame_count-1];
        packetdata.error = current_data;
        break;
    case 32:
        packetdata.checksum = current_data;
        byte_sum -= ((current_data>>8)+(current_data&0xFF));
        break;
    default:
        break;
    }
}

void PM_7003::print_messages(void){
    /*
```

```
     * Print messages to string and Serial screen
     */
    Serial.println("-----------------------");
    Serial.print("PMS 7003 - Reading #");
    Serial.println(read_count);
    Serial.println("-----------------------");
    sprintf(print_buffer, ", %02x, %02x, %04x, ",
        packetdata.start_frame[0], packetdata.start_frame[1], packetdata.frame_length);
    sprintf(print_buffer, "%s%04d, %04d, %04d, ", print_buffer,
        packetdata.concPM1_0_factory, packetdata.concPM2_5_factory, packetdata.concPM10_0_factory);
    sprintf(print_buffer, "%s%04d, %04d, %04d, ", print_buffer,
        packetdata.concPM1_0_ambient, packetdata.concPM2_5_ambient, packetdata.concPM10_0_ambient);
    sprintf(print_buffer, "%s%04d, %04d, %04d, %04d, %04d, %04d, ", print_buffer,
        packetdata.countPM0_3um, packetdata.countPM0_5um, packetdata.countPM1_0um,
        packetdata.countPM2_5um, packetdata.countPM5_0um, packetdata.countPM10_0um);
    sprintf(print_buffer, "%s%02d, %02d, ", print_buffer,
        packetdata.version, packetdata.error);

    pm2_5 = packetdata.countPM1_0um - packetdata.countPM2_5um + packetdata.countPM0_5um - packetdata.countPM1_0um + packetdata.countPM0_3um - packetdata.countPM0_5um;
    Serial.println(print_buffer);
    Serial.println("-----------------------");
    delay(500);
}
```