// Includes following functions:

// checkMedTimes() \*Returns true if the current time indicates that it is time to dispense medication based on the stored medtimes\*

// buttonPress() \*interrupt triggers this function when the button is pressed, effectively raises flag to exit passive loop\*

#include <RTClib.h>

#include <Wire.h>

#include <SerLCD.h>

#include <Adafruit\_MCP23X17.h>

#include <Adafruit\_Fingerprint.h>

#define BUTTON 2

#define DLATCH 3

#define BUZZER 4

#define MOTOR1 5

#define MOTOR2 6

#define MOTOR3 7

#define MOTOR4 8

//Fingerprint Scanner Serial

#define mySerial Serial1

//Keypad number of rows and columns

#define ROWS 4

#define COLS 4

Adafruit\_Fingerprint finger = Adafruit\_Fingerprint(&mySerial); //Fingerprint Scanner

RTC\_DS3231 rtc; //Real Time Clock

Adafruit\_MCP23X17 mcp; //Keypad I2C adapter

SerLCD lcd;

char keys[ROWS][COLS] = { //Keypad map

{'D', '#', '0', '\*'},

{'C', '9', '8', '7'},

{'B', '6', '5', '4'},

{'A', '3', '2', '1'}

};

uint8\_t rowPins[ROWS] = {0, 1, 2, 3}; // Connect to the row pinouts of the keypad

uint8\_t colPins[COLS] = {4, 5, 6, 7}; // Connect to the column pinouts of the keypad

int activeCoil; //Indicates what coil(s) (MOTORs 1, 2, 3, and 4) are currently active in the stepper motor

int medTimes[3][2]; //Stores up to 3 unique times that medicine must be dispensed

int lastDisp; //Unixtime for the last time medicine was dispensed

bool scan; //Set to true on activation of standby button, triggers fingerprint scan process

bool timeForMeds; //Indicates whether it is an appropriate time to dispense a medication

uint8\_t id; //Fingerprint ID number currently being used by scanner

char fingerprints[20]; //All stored fingerprints (max of 20) with role (U or A) stored at index ID - 1

int numPrints; //The total number of prints that are/need to be stored currently

char lastScannedRole; //Role associated with the most recently scanned user

int halfStepsToDispense; //Number of half steps the motor goes through to dispense the next slot of meds

int slotsRemaining; //Keeps track of how many slots in the cylinder still contain meds

void setup() {

// put your setup code here, to run once:

slotsRemaining = 7;

halfStepsToDispense = 129;

//Start Keypad Setup

// Serial.begin(9600);

// Wire.begin();

//

// if (!mcp.begin\_I2C()) { // Default address is 0x20

// Serial.println("Error finding MCP23017. Check wiring!");

// while (1);

// }

//

// // Set up the row pins as outputs

// for (int i = 0; i < ROWS; i++) {

// mcp.pinMode(rowPins[i], OUTPUT);

// mcp.digitalWrite(rowPins[i], HIGH); // deactivate row

// }

//

// // Set up the column pins as inputs with pull-up resistors

// for (int i = 0; i < COLS; i++) {

// mcp.pinMode(colPins[i], INPUT\_PULLUP);

// }

//

// Serial.println("Keypad initialized.");

//End Keypad Setup

//Start LCD Screen Setup

// lcd.begin(Wire);

// lcd.setBacklight(255, 255, 255); //Set screen backlight to bright white

// lcd.setContrast(0); //Set to maximum contrast for readability

// lcd.clear();

//End LCD Screen Setup

//Start Scanner Setup

// while (!Serial); // For Yun/Leo/Micro/Zero/...

// delay(100);

// Serial.println("\n\nAdafruit Fingerprint sensor enrollment");

//

// // set the data rate for the sensor serial port

// finger.begin(57600);

//

// if (finger.verifyPassword()) {

// Serial.println("Found fingerprint sensor!");

// } else {

// Serial.println("Did not find fingerprint sensor :(");

// while (1) { delay(1); }

// }

//End Scanner Setup

pinMode(MOTOR1, OUTPUT);

pinMode(MOTOR2, OUTPUT);

pinMode(MOTOR3, OUTPUT);

pinMode(MOTOR4, OUTPUT);

pinMode(BUZZER, OUTPUT);

pinMode(BUTTON, INPUT);

attachInterrupt(digitalPinToInterrupt(2), buttonPress, CHANGE);

activeCoil = 1;

//setupSequence();

}

void loop() {

Serial.println();

Serial.println("Slots Remaining:");

Serial.println(slotsRemaining);

Serial.println("ActiveCoil:");

Serial.println(activeCoil);

Serial.println();

delay(1000);

motorHalfSteps(halfStepsToDispense);

slotsRemaining--;

// put your main code here, to run repeatedly:

// timeForMeds = checkMedTimes();

// if(scan) {

// digitalWrite(BUZZER, LOW);

// clearScreen();

// screenPrint(1, "Button detected");

// screenPrint(2, "Please Scan Print");

// lastScannedRole = checkPrint();

//

// if(lastScannedRole != 'N' && timeForMeds) {

// dispenseMeds();

// }

// else if(lastScannedRole == 'A') {

// //Admin options

// clearScreen();

// screenPrint(0, "Select Action:");

// screenPrint(1, "A - Rerun Setup");

// screenPrint(2, "B - Open Door");

// screenPrint(3, "Other - Cancel");

// switch(getKeyInput()) {

// case 'A':

// setupSequence();

// break;

// case 'B':

// //openDoor();

// digitalWrite(DLATCH, LOW);

// clearScreen();

// screenPrint(0, "Door Open:");

// screenPrint(1, " Press any key once");

// screenPrint(2, " door is closed to ");

// screenPrint(3, " reengage door lock");

//

// getKeyInput();

// digitalWrite(DLATCH, HIGH);

// clearScreen();

// break;

// default:

// break;

// }

// }

// else {

// clearScreen();

// screenPrint(0, "Invalid print found ");

// screenPrint(1, " or not time for a ") ;

// screenPrint(2, " Med Pass");

// delay(5000);

// clearScreen();

// }

// scan = false;

// } else if (timeForMeds) {

// digitalWrite(BUZZER, HIGH);

// }

}

bool checkMedTimes() {

//return true if the current time is within a certain margin of error of any medTimes AND the medicine hasn't already been dispensed

DateTime now = rtc.now();

int i = 0;

while(i < 3) {

if( now.hour()==medTimes[i][0] && (abs(now.minute() - medTimes[i][1]) <= 5) && abs(now.unixtime() - lastDisp) > 600 ) {

return true;

}

}

return false;

}

void buttonPress() {

scan = true;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*KEYPAD FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Includes following functions:

// getKeyInput() \*monitors keypad for user input, debounces, and returns first keypress detected as associated character\*

// get2DigitInput() \*stores first 2 keypad presses detected and translates characters into 2 digit integer, MAY CAUSE ISSUES IF NON-NUM KEY INPUTS\*

char getKeyInput() {

char input = 0;

while(input = 0){

for (int row = 0; row < ROWS; row++) {

mcp.digitalWrite(rowPins[row], LOW); // Activate row

for (int col = 0; col < COLS; col++) {

// Check if button is pressed (active low)

if (mcp.digitalRead(colPins[col]) == LOW) {

// Debounce

delay(20);

if (mcp.digitalRead(colPins[col]) == LOW) {

Serial.print("Button ");

Serial.print(keys[row][col]);

Serial.println(" pressed");

input = keys[row][col];

// Wait for button release

while (mcp.digitalRead(colPins[col]) == LOW) {

delay(10); // Simple debounce

}

delay(20); // More debounce

}

}

}

mcp.digitalWrite(rowPins[row], HIGH); // Deactivate row

}

}

return input;

}

int get2DigitInput() {

char input1 = getKeyInput();

char input2 = getKeyInput();

int input1AsInt = input1 - '0';

int input2AsInt = input1 - '0';

return (input1AsInt \* 10) + input2AsInt;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FINGERPRINT SCANNER FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//NOTE: Fingerprint Scanner Code derived from enroll and fingerprint examples with slight modifications

//NOTE: SETUP FOR SCANNER MUST BE ADDED/INCLUDED IN SETUP FUNCTION OF SD\_Main!!!

// Includes following functions:

// storePrints() \*setup function for storing all user and admin fingerprints

// checkPrint() \*Gives user 5 tries to scan fingerprint, returns character indicating scanned fingerprint's role

// enrollPrint() \*Enrolls scanned fingerprint and stores the associated role in fingerprints[]

// getFingerprintEnroll() \*Original enroll example code\*

// getFingerprintID() \*Original get ID example code for reference\* \*\*NOT USED\*\*

// getFingerprintIDez() \*Original quick ID example code\*

int storePrints(int numUsers, int PPU) {

numPrints = PPU \* numUsers;

char scanFor = 'A';

for (int i = 1; i <= numPrints; i++) {

if (i > PPU) {

scanFor = 'U';

}

if (scanFor == 'A') {

clearScreen();

screenPrint(1, "Please scan next ");

screenPrint(2, " admin fingerprint ");

} else {

clearScreen();

screenPrint(1, "Please scan next ");

screenPrint(2, " user fingerprint ");

}

id = i;

enrollPrint(scanFor);

clearScreen();

screenPrint(1, "Print stored! ");

delay(1000);

}

}

char checkPrint() {

uint8\_t printID = -1;

int i = 1;

while(printID < 0 && i <= 5) {

printID = getFingerprintIDez();

i++;

}

if(printID == -1) {

return 'N';

} else {

return fingerprints[printID-1];

}

}

void enrollPrint(char role) {

Serial.print("Enrolling ID #");

Serial.print(id);

Serial.print(" as ");

Serial.println(role);

fingerprints[id-1] = role;

while (! getFingerprintEnroll() );

}

uint8\_t getFingerprintEnroll() {

int p = -1;

Serial.print("Waiting for valid finger");

while (p != FINGERPRINT\_OK) {

p = finger.getImage();

switch (p) {

case FINGERPRINT\_OK:

Serial.println("Image taken");

break;

case FINGERPRINT\_NOFINGER:

Serial.print(".");

break;

case FINGERPRINT\_PACKETRECIEVEERR:

Serial.println("Communication error");

break;

case FINGERPRINT\_IMAGEFAIL:

Serial.println("Imaging error");

break;

default:

Serial.println("Unknown error");

break;

}

}

// OK success!

p = finger.image2Tz(1);

switch (p) {

case FINGERPRINT\_OK:

Serial.println("Image converted");

break;

case FINGERPRINT\_IMAGEMESS:

Serial.println("Image too messy");

return p;

case FINGERPRINT\_PACKETRECIEVEERR:

Serial.println("Communication error");

return p;

case FINGERPRINT\_FEATUREFAIL:

Serial.println("Could not find fingerprint features");

return p;

case FINGERPRINT\_INVALIDIMAGE:

Serial.println("Could not find fingerprint features");

return p;

default:

Serial.println("Unknown error");

return p;

}

Serial.println("Remove finger");

delay(2000);

p = 0;

while (p != FINGERPRINT\_NOFINGER) {

p = finger.getImage();

}

Serial.print("ID "); Serial.println(id);

p = -1;

Serial.println("Place same finger again");

while (p != FINGERPRINT\_OK) {

p = finger.getImage();

switch (p) {

case FINGERPRINT\_OK:

Serial.println("Image taken");

break;

case FINGERPRINT\_NOFINGER:

Serial.print(".");

break;

case FINGERPRINT\_PACKETRECIEVEERR:

Serial.println("Communication error");

break;

case FINGERPRINT\_IMAGEFAIL:

Serial.println("Imaging error");

break;

default:

Serial.println("Unknown error");

break;

}

}

// OK success!

p = finger.image2Tz(2);

switch (p) {

case FINGERPRINT\_OK:

Serial.println("Image converted");

break;

case FINGERPRINT\_IMAGEMESS:

Serial.println("Image too messy");

return p;

case FINGERPRINT\_PACKETRECIEVEERR:

Serial.println("Communication error");

return p;

case FINGERPRINT\_FEATUREFAIL:

Serial.println("Could not find fingerprint features");

return p;

case FINGERPRINT\_INVALIDIMAGE:

Serial.println("Could not find fingerprint features");

return p;

default:

Serial.println("Unknown error");

return p;

}

// OK converted!

Serial.print("Creating model for #"); Serial.println(id);

p = finger.createModel();

if (p == FINGERPRINT\_OK) {

Serial.println("Prints matched!");

} else if (p == FINGERPRINT\_PACKETRECIEVEERR) {

Serial.println("Communication error");

return p;

} else if (p == FINGERPRINT\_ENROLLMISMATCH) {

Serial.println("Fingerprints did not match");

return p;

} else {

Serial.println("Unknown error");

return p;

}

Serial.print("ID "); Serial.println(id);

p = finger.storeModel(id);

if (p == FINGERPRINT\_OK) {

Serial.println("Stored!");

} else if (p == FINGERPRINT\_PACKETRECIEVEERR) {

Serial.println("Communication error");

return p;

} else if (p == FINGERPRINT\_BADLOCATION) {

Serial.println("Could not store in that location");

return p;

} else if (p == FINGERPRINT\_FLASHERR) {

Serial.println("Error writing to flash");

return p;

} else {

Serial.println("Unknown error");

return p;

}

return true;

}

uint8\_t getFingerprintID() {

uint8\_t p = finger.getImage();

switch (p) {

case FINGERPRINT\_OK:

Serial.println("Image taken");

break;

case FINGERPRINT\_NOFINGER:

Serial.println("No finger detected");

return p;

case FINGERPRINT\_PACKETRECIEVEERR:

Serial.println("Communication error");

return p;

case FINGERPRINT\_IMAGEFAIL:

Serial.println("Imaging error");

return p;

default:

Serial.println("Unknown error");

return p;

}

// OK success!

p = finger.image2Tz();

switch (p) {

case FINGERPRINT\_OK:

Serial.println("Image converted");

break;

case FINGERPRINT\_IMAGEMESS:

Serial.println("Image too messy");

return p;

case FINGERPRINT\_PACKETRECIEVEERR:

Serial.println("Communication error");

return p;

case FINGERPRINT\_FEATUREFAIL:

Serial.println("Could not find fingerprint features");

return p;

case FINGERPRINT\_INVALIDIMAGE:

Serial.println("Could not find fingerprint features");

return p;

default:

Serial.println("Unknown error");

return p;

}

// OK converted!

p = finger.fingerSearch();

if (p == FINGERPRINT\_OK) {

Serial.println("Found a print match!");

} else if (p == FINGERPRINT\_PACKETRECIEVEERR) {

Serial.println("Communication error");

return p;

} else if (p == FINGERPRINT\_NOTFOUND) {

Serial.println("Did not find a match");

return p;

} else {

Serial.println("Unknown error");

return p;

}

// found a match!

Serial.print("Found ID #"); Serial.print(finger.fingerID);

Serial.print(" with confidence of "); Serial.println(finger.confidence);

return finger.fingerID;

}

// returns -1 if failed, otherwise returns ID #

int getFingerprintIDez() {

uint8\_t p;

while (p != FINGERPRINT\_OK) {

p = finger.getImage();

// switch (p) {

// case FINGERPRINT\_OK:

// Serial.println("Image taken");

// break;

// case FINGERPRINT\_NOFINGER:

// Serial.print(".");

// break;

// case FINGERPRINT\_PACKETRECIEVEERR:

// Serial.println("Communication error");

// break;

// case FINGERPRINT\_IMAGEFAIL:

// Serial.println("Imaging error");

// break;

// default:

// Serial.println("Unknown error");

// break;

// }

}

// if (p != FINGERPRINT\_OK) return -1;

p = finger.image2Tz();

if (p != FINGERPRINT\_OK) return -1;

p = finger.fingerFastSearch();

if (p != FINGERPRINT\_OK) return -1;

// found a match!

Serial.print("Found ID #"); Serial.print(finger.fingerID);

Serial.print(" with confidence of "); Serial.println(finger.confidence);

return finger.fingerID;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SCREEN FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Note: Screen is 20 columns by 4 rows

// Includes following functions:

// screenPrint() \*prints parameter string to the screen as a single line, sets cursor to next line\*

// clearScreen() \*Clears all lines of text from the screen, resetting cursor\*

void screenPrint(int row, char text[]) {

lcd.setCursor(0, row);

lcd.print(" ");

lcd.setCursor(0, row);

lcd.print(text);

}

void clearScreen() {

lcd.clear();

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SETUP SEQUENCE FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Includes following functions:

// setupSequence() \*Takes user inputs to set up med time schedule and store all fingerprints, can be rerun to overwrite/reset schedule and stored prints\*

// getNumUsers() \*Prompts user for the number of users, A for independent user, B for caretaker, returns 1 for A and 2 for B\*

// getPPU() \*Prompts user for prints per user, returns integer prints per user\*

// getPPD() \*Prompts user for medpasses per day, returns integer keypad input (1, 2, or 3)\*

// setMedTimes() \*Constructs and returns med pass schedule array times[3][2] based on user inputs, of form [[hr, min], [hr, min], [hr, min]]\*

void setupSequence() {

int numUsers; //Indicates whether there is a separate caregiver (2) or not (1)

int PPU; //Indicates number of Prints Per User that are stored

int numPrints; //Indicates total number of prints stored

char scanFor; //Indicates whether current fingerprints being scanned are for an Admin ('A') or User ('U')

int PPD; //Indicates number of med Passes Per Day

finger.emptyDatabase(); //Clears any currently stored fingerprints from database if setup is being rerun

for(int i = 0; i < 20; i++) { //Clears fingerprints array of previously stored roles

fingerprints[i] = '0';

}

while (numUsers == 0 || PPU == 0) {

numUsers = getNumUsers();

PPU = getPPU();

if (numUsers > 2 || numUsers < 1 || PPU > 10 || PPU < 1) {

clearScreen();

screenPrint(1, "INVALID INPUT(S) ");

delay(2500);

numUsers = 0;

PPU = 0;

}

}

storePrints(numUsers, PPU);

PPD = 0;

while (PPD == 0) {

PPD = getPPD();

if (PPD < 1 || PPD > 3) {

clearScreen();

screenPrint(1, "INVALID INPUT(S) ");

delay(2500);

PPD = 0;

}

}

setMedTimes(PPD);

clearScreen();

screenPrint(1, "Setup complete! ");

delay(2500);

}

int getNumUsers() {

char input;

clearScreen();

screenPrint(0, "Select Use Case: ");

screenPrint(1, "A) Independent User ");

screenPrint(2, "B) User w/ Caregiver");

while(input != 'A' && input != 'B') {

input = getKeyInput();

}

clearScreen();

if(input == 'A') {

return 1;

} else {

return 2;

}

}

int getPPU() {

int input = 0;

while(input > 10 || input < 1) {

clearScreen();

screenPrint(0, "Please enter number ");

screenPrint(1, " of prints per user");

screenPrint(2, " (2 digits: 01-10) ");

input = get2DigitInput();

if(input > 10 || input < 1) {

clearScreen();

screenPrint(1, "INVALID INPUT(S) ");

delay(2500);

}

clearScreen();

}

return input;

}

int getPPD() {

char input;

int inputAsInt;

clearScreen();

screenPrint(0, "Please Input number ");

screenPrint(1, " of med passes per ");

screenPrint(2, " day (1-3) ");

while(input != '1' && input != '2' && input != '3') {

input = getKeyInput();

}

clearScreen();

inputAsInt = input - '0';

return inputAsInt;

}

void setMedTimes(int PPD) {

int hrs = 25;

int mins = 70;

for(int j = 0; j < 3; j++) {

for(int k = 0; k < 2; k++) {

medTimes[j][k] = -1;

}

}

for(int i = 0; i < PPD; i++) {

while((hrs > 24 || hrs < 0) || (mins > 59 || mins < 0)) {

clearScreen();

screenPrint(0, "Enter med pass time ");

screenPrint(1, " in 24 hour format ");

screenPrint(2, " (9:00am -> 0900) ");

screenPrint(3, " (3:30pm -> 1530) ");

hrs = get2DigitInput();

mins = get2DigitInput();

if((hrs > 24 || hrs < 0) || (mins > 59 || mins < 0)) {

clearScreen();

screenPrint(1, "INVALID INPUT(S) ");

delay(2500);

}

}

clearScreen();

screenPrint(1, "Time Stored! ");

delay(2500);

medTimes[i][0] = hrs;

medTimes[i][1] = mins;

hrs = 25;

mins = 70;

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*STEPPER FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Includes following functions:

// motorHalfSteps() \*Rotates stepper motor through a given number of halfsteps, starting at activeCoil input\*

// dispenseMeds() \*Rotate stepper enough to dispense next med slot, return number of filled slots remaining in cylinder\*

int motorHalfSteps(int numHalfSteps) {

for(int i = 1; i <= numHalfSteps; i++) {

switch(activeCoil) {

case 1:

activeCoil = 13;

digitalWrite(MOTOR3, HIGH);

break;

case 13:

activeCoil = 3;

digitalWrite(MOTOR1, LOW);

break;

case 3:

activeCoil = 32;

digitalWrite(MOTOR2, HIGH);

break;

case 32:

activeCoil = 2;

digitalWrite(MOTOR3, LOW);

break;

case 2:

activeCoil = 24;

digitalWrite(MOTOR4, HIGH);

break;

case 24:

activeCoil = 4;

digitalWrite(MOTOR2, LOW);

break;

case 4:

activeCoil = 41;

digitalWrite(MOTOR1, HIGH);

break;

case 41:

activeCoil = 1;

digitalWrite(MOTOR4, LOW);

break;

}

delay(10);

}

return activeCoil;

}

int dispenseMeds() {

char buff[21];

DateTime now = rtc.now();

lastDisp = now.unixtime();

motorHalfSteps(halfStepsToDispense);

slotsRemaining--;

itoa(slotsRemaining, buff, 10);

clearScreen();

screenPrint(1, "Slots Remaining: ");

screenPrint(2, buff);

}