Final Project: Math Alarm Clock

CMPE 3815: Microcontroller Systems

Danny Burger

Partner: Erik Simkins

Intro:

The Math Alarm Clock project is a different approach to ensuring users wake up by engaging their brain. Unlike conventional alarms, this device integrates a math problem-solving feature that requires users to solve equations to deactivate the alarm. The project makes use of an Arduino Uno, a Real Time Clock (RTC) module, a 4x4 keypad matrix, a buzzer, and a 2x16 LCD screen. This report details the project's goals, development, and functionality, along with a walkthrough of the code implementation.

Project Goals:

The goal of this final project was to create an alarm clock that required solving a math problem before the alarm could be turned off. The primary requirements were as follows:

1. The clock shall display the current time.
2. The current time shall be settable.
3. The clock shall have a settable alarm.
4. The clock shall require the user to answer a generated math problem correctly before the alarm is turned off

Beyond the three primary requirements, some stretch goals were set. These were goals that would be good additions to the product, but are not essential to product functionality.

1. 3d printed case
2. Different alarm sounds

The original project proposal mentioned using an OLED display, however the OLEDs available in the lab were too small for this application. A larger OLED could have been ordered, but a 16x2 I2C LCD display was selected instead to decrease cost.

Project Development:

After finalizing the concept, development of the product consisted of four main stages.

1. Wiring up the physical circuit.
2. Implementing the real time clock module to display current time.
3. Create a function that generates random math problems .
4. Combine the math and alarm functions.

Physical Circuit:

The physical circuit consists of an Arduino Uno, 16x2 I2C LCD, matrix keypad, RTC module, and an active (DC) buzzer. The circuit schematic is shown below. Wiring was made simple due to both the LCD display and the RTC module being I2C compatible.

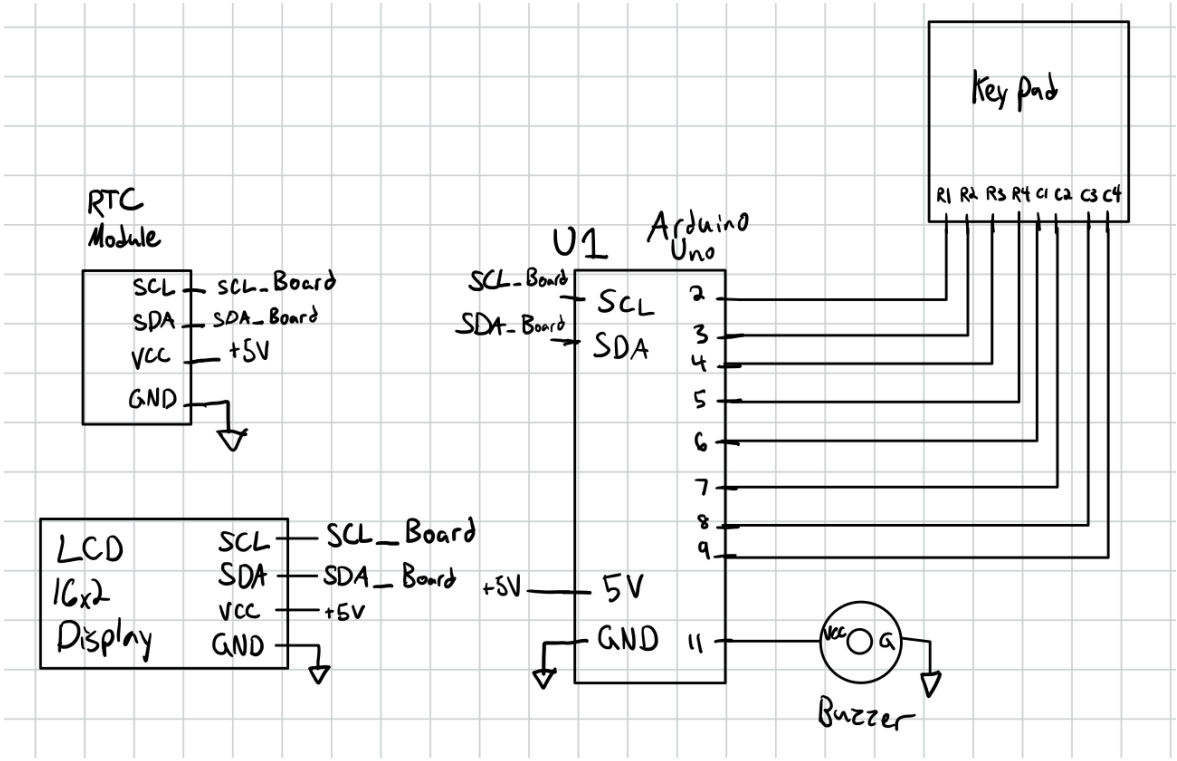


Figure 1 - Circuit schematic

Time Functions Code:

The two critical functions of an alarm clock are to accurately display the current time, and to alert or wake up the user at a preset time. To cover the first functionality, an Elegoo real time clock (RTC) module was used to accurately measure time. The RTC module utilizes a DS1307 chip which provides seconds, minutes, hours, day, date, month, and year information. An onboard button cell battery acts as a backup power supply so that timekeeping continues even when the module loses power. The "uRTCLib.h" library was used to interact with the RTC module. The library includes functions to set the current time and read any of the time units using rtc.hour(), rtc.day(), etc.

The current time is determined from the RTC module using the get_time() function. The function begins by refreshing the RTC chip using rtc.refresh(). After refreshing the time chip the hours, minutes and seconds are read and stored as integers.

```
//get hours, mins, secs
int hours = rtc.hour();
int mins = rtc.minute();
int secs = rtc.second();
```

Next, each time unit is printed to LCD and appended to a time string in hh:mm:ss format. After printing and appending, the time string is returned in order to be compared to an alarm time string.

```
//make hh format
if (hours < 10){
  lcd.print("0"); //print to LCD
  time = time + "0"; //append to time string
}
lcd.print(hours); //print to LCD
time = time + String(hours); //append to time string
```

The DS1307 chip is not able to determine the current time out of the box, so the user must set it. The function set_time() allows the user to set the current time. The function is called when the user presses the "D" key on the keypad. Key presses are registered using the keypadListen() function in void loop.

```
//read keypad press
char key = keypadListen();
Serial.println(key);

//set the time if D is pressed
if (key == 'D'){
  set_time();
}
```

Once the "D" key is pressed, the user is prompted to enter the hour on the LCD display. Input is registered using keypadListen() and error handling ensures the user entered two integers that make up a number between 00 and 23.

```
for (int i = 0; i < 2; i++) {
  char key = 0;

  // Wait for a key press
  while (!key) key = keypadListen();

  // Check if the key is a valid digit
  if (key >= '0' && key <= '9') {
    lcd.print(key); // Display the digit on the LCD
    h = h * 10 + (key - '0'); // Accumulate h
  }
  else {
    i--; // If invalid key, repeat the step
  }
}

// Ensure valid range for hours
if (h < 0 || h > 23) {
  lcd.clear();
  lcd.print("Invalid Hours!");
  delay(1500);
  lcd.clear();
  return; // Exit if invalid input
}
```

The same method is used to prompt the user to enter the minutes value of the current time, ensuring a value between 00 and 59 is entered. The seconds are set to 00. With the hours and minutes inputs gathered and stored, the rtc.set() function is used to set the RTC chips internal time to the user's input time.

```
//send time to RTC
rtc.set(s, m, h, 0, 0, 0, 0); //(second, minute, hour, dayOfWeek, dayOfMonth, month, year)
```

The set_alarm() function is practically identical to the set_time() function, but does not use rtc.set() to set the RTC chips time. Instead, the alarm time is returned as a string to be compared to the current time string. The toggle_alarm() function toggles the alarm function on or off. The disp_alarm() function is then used to display the alarm time and the state of the alarm (on or off.) The image below shows the format of the clock while not actively in an alarm state.

Figure 2 - Display format when not in alarm state.

Math Problem Function Code:

The math problem generation took a different route compared to the one week project earlier in the semester. Instead of generating a random math problem and making the user input the answer, the user is prompted with two math expressions and a solution. The user must determine which of the two expressions equal the provided solution. This section of the report will walk through the code used to generate the math problems.

The function begins by setting the random() seed to a floating pin A0, and clearing the LCD. Next, an expression consisting of four single digit operands are randomly generated in an array. The operands are generated using random() along with a ternary operator to determine + or -. After generating the expression, the solution is found by summing the array. This generated expression will be the "correct" expression

```
// create an expression with 3-4 operands that add or subtract to a number between x and y (probably 1ish and 20ish)
  // use random() to generate the operands, and to determine if + or - between them
  int numOperands = 4;
  int operands[numOperands];

  // create array of random operands
  for (int i = 0; i < numOperands; i++) {
    operands[i] = random(1, 10) * (random(0, 2) == 0 ? -1 : 1);
  }

  // solve the created problem
  int solution;
  for (int i = 0; i < numOperands; i++) {
    solution += operands[i];
  }
```

The correct expression is then used to create a "correct string" for displaying to the LCD. A for loop is used to add '+' characters before positive operands.

```
// create string of correct expression
String correctString = "";
for (int i = 0; i < numOperands; i++) {
  if (operands[i] > 0) {
    correctString += "+"; // Add a + sign for positive operands
  }
  correctString += String(operands[i]); // Append the operand value
}
```

Now that the correct expression has been generated and stored as a string, the wrong expression must be created. The idea is to change two of the operands to different values. This is accomplished by generating two random indexes that are ensured to be different using a do while statement.

```
// change two of the operands to create a wrong expression
// by generating two random indexes
int index1 = random(0, numOperands);
int index2;
do {
  index2 = random(0, numOperands); // Ensure the second index is different
}

while (index2 == index1);
```

Now, two of the operands as selected by the generated indexes are rerolled using random() and a ternary operator to determine the sign of the new operand. A “wrong” string is created for display

```
// assign new random values to the generated indexes
operands[index1] = random(1, 10) * (random(0, 2) == 0 ? -1 : 1);
operands[index2] = random(1, 10) * (random(0, 2) == 0 ? -1 : 1);

// Create a string of the updated operands with + for positive numbers
String wrongString = "";
for (int i = 0; i < numOperands; i++) {
  if (operands[i] > 0) {
    wrongString += "+"; // Add a + sign for positive operands
  }
  wrongString += String(operands[i]); // Append the operand value
}
```

Now that the correct and wrong expressions are generated, they cannot just be printed to row A or B in the same order each time. This was accomplished by selecting the row of the correct string using

random() and setting the wrong string to the opposite row. Next, using if else the correct row is corresponded to either the ‘A’ or ‘B’ character for comparison to key presses by the user, and printing to the LCD.

```
// next, roll random(0,2) for correct expression's cursor row
int cursorRowCorrect = random(0,2);

// set wrong expression to opposite of rolled
int cursorRowWrong = 1 - cursorRowCorrect;

char ansRow;
char otherRow;
// indicate which answer is correct for button press on keypad
if (cursorRowCorrect == 0) {
  ansRow = 'A';
  otherRow = 'B';
}

else {
  otherRow = 'A';
  ansRow = 'B';
}
```

Both expressions are printed to the LCD according to the randomly selected rows, and the solution is printed in the top right of the display.

```
// printing expressions to LCD
  // print correct expression
  lcd.setCursor(0,cursorRowCorrect);
  lcd.print(ansRow);
  lcd.print(":");
  lcd.print(correctString);

  // print wrong expression
  lcd.setCursor(0,cursorRowWrong);
  lcd.print(otherRow);
  lcd.print(":");
  lcd.print(wrongString);

  //print solution in top right corner
  lcd.setCursor(13,0);
  lcd.print(solution);
```

Now that the expressions and the solution are generated and displayed, the user inputs to the keypad must be read, and interpreted for correctness. The keypadListen() function is used with a while loop to wait for key presses. If statements are used to ensure only vaid button presses are accepted.

```
// set key to zero initially
char key = 0;
char keyAns;

// wait for a key press
while (!key) key = keypadListen(); {

  // validate key press
  if (key == 'A' || key == 'B') {
    keyAns = key;
  }
  else {
    key = 0; // If invalid key, repeat the step
  }
}
```

Once the ‘A’ or ‘B’ button is pressed, the character is stored in keyAns. keyAns is compared to the answer row variable that denotes the correct expression. If the key pressed is correct, a value of 1 or TRUE is returned via the “correct” variable. If the answer is incorrect, the function returns FALSE or 0.

```
// check for correct answer
if (keyAns == ansRow) {
  correct = 1;
  Serial.print("Correct: ");
  Serial.println(correct);
}

return correct;
```

Combining the functions:

Now that all of the key functions have been explained, void setup and void loop will be walked through. Void setup contains standard LCD initialization and pin definitions for the keypad and buzzer. Serial and I2C communication are also started in setup.

The void loop is where all of the aforementioned functions are used to accomplish the purpose of this project: create a math alarm clock. The loop begins with reading for key presses using keypadListen(). keypadListen() determines key presses by scanning for a logic low one column at a time as done in lab earlier this semester.

```
//read keypad press
char key = keypadListen();
```

Buttons A, B and D are used to call the set_time, set_alarm, and toggle_alarm functions.

```
//set the time if D is pressed
if (key == 'D'){
  set_time();
}

//set the alarm if A is pressed
if (key == 'A'){
  alarm_time = set_alarm();
}

//toggle alarm if B is pressed
int alarm_state;
if (key == 'B'){
  alarm_state = toggle_alarm();
}
```

Next, the current time and alarm time are printed to the LCD by calling the get_time and disp_alarm functions.

```
//display current time
String current_time = get_time();
//display alarm time
disp_alarm(alarm_time, alarm_state);
```

Finally, the last part of the loop code performs the alarm functionality within a while loop. The while loop is entered when the current time matches the alarm time, and the alarm is toggled on. The alarmExit == 0 is used to ensure that the alarm will not retrigger if the user completes the math problem within the same minute as the alarm time.

```
while(current_time == alarm_time && alarm_state == 1 && alarmExit == 0){
```

On entering the while loop, the LCD is cleared, the cursor set to 0,0 and the boolean correctFlag is initialized and set to false.

```
lcd.clear();
lcd.setCursor(0,0);
bool correctFlag = 0;
```

Another while loop is then entered that depends on the correctFlag being false. Inside this loop the buzzer is turned on via the buzzer_ON function and the mathProblem function called to the correctFlag variable. Recalling from the mathProblem() walkthrough, the function will return a 0 for a wrong answer and a 1 for a correct answer. Therefore, getting the problem correct will exit the while loop.

```
while (correctFlag == 0) {
  buzzer_ON();

  correctFlag = mathProblem();
}
```

Upon exiting the loop, the alarmExit boolean will be set to TRUE and the buzzer turned off.

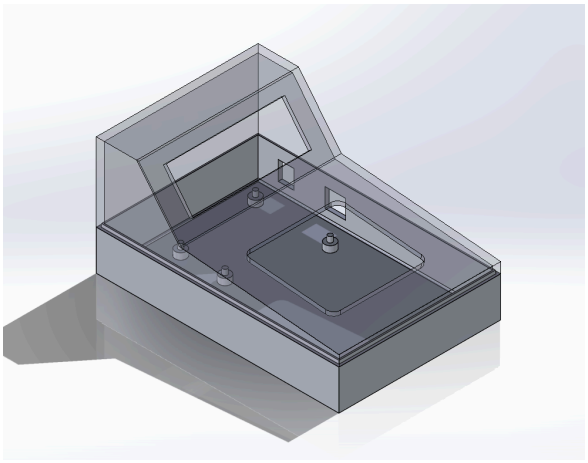
```
alarmExit = 1;
buzzer_OFF();
lcd.clear();
```

To ensure the alarm will not retrigger within the same minute if the user solves the problem quickly, an if statement is used to set the alarmExit boolean to FALSE unless the current time does not equal the alarm time. This concludes the code walkthrough.

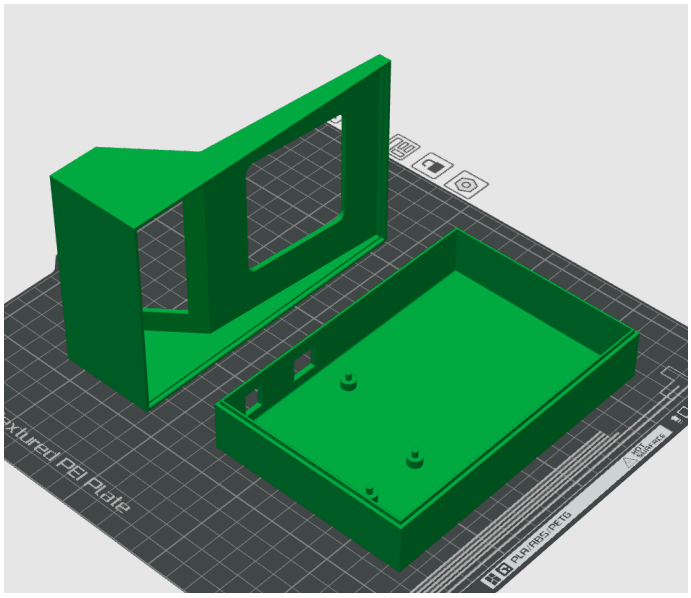
```
if (alarmExit == 1 && current_time != alarm_time) {
  alarmExit = 0; // Reset the flag for the next alarm cycle
}
```

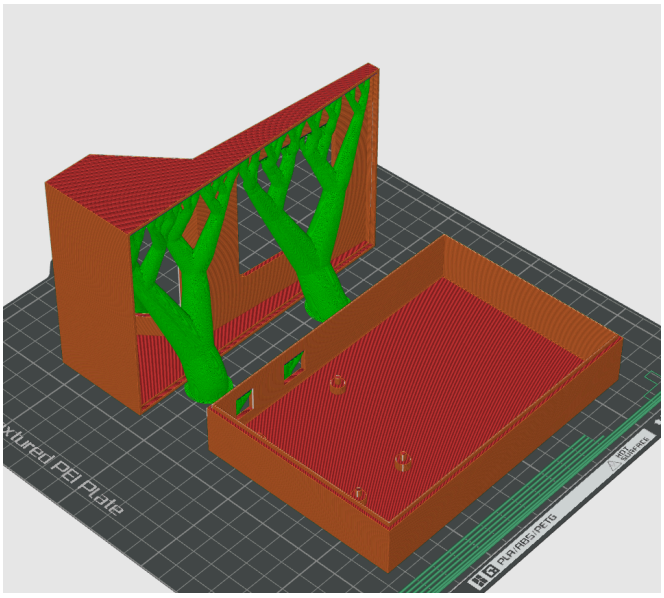
3d Printing the Case:

One other goal of this project was to create a case for the alarm clock. 3D printing was chosen for this task, due to its low cost and high availability on campus. The initial design inspiration came from an old accounting calculator. The case was modeled in SolidWorks around 3d models of the Arduino, keypad, and LCD components.



The 3D printing was carried out via the UVM FabLab using a Bambu Labs X1 Carbon printer. The build plate arrangement is shown below, along with the Gcode file showing support. The print took around four hours to complete. Pictures of the finished print are added in the appendix.





The physical circuit was inserted into the case, and it was revealed that the standoffs and side cutout were not dimensioned properly. The standoffs were ground down using a dremel which worked to fit the Arduino. Hot glue was used to secure the Arduino and LCD display. The matrix keypad was snap-fit. The ribbon cable made closing the case difficult, as it acted like a spring. Tape was used to secure the case closed. Shortening the wiring could rid the need for tape. The assembled alarm clock is shown below.

Figure 3 - Assembled alarm clock.

Appendix:

Pictures:

Code:

```
//lib//libraries to control the LCD and RTC
#include <LiquidCrystal_I2C.h>
#include "uRTCLib.h"

//set up LCD screen
LiquidCrystal_I2C lcd(0x27,16,2);

// uRTCLib rtc;
uRTCLib rtc(0x68);

//define column pins for keypad
int C1 = 6;
int C2 = 7;
int C3 = 8;
int C4 = 9;

//define row pins for keypad
int R1 = 2;
int R2 = 3;
int R3 = 4;
```

```
int R4 = 5;

//these values are used in the keypadListen function in the for loops
const int nRows = 4;
const int nCols = 4;

//buzzer pin
int buzzer_pin = 11;

//define keymap for keypad
char Keymap[nRows][nCols] = {
  {'A', 'B', 'C', 'D'},
  {'3', '6', '9', '#'},
  {'2', '5', '8', '0'},
  {'1', '4', '7', '*'}
};


//define corresponding pins
int pin_rows[nRows] = {R4, R3, R2, R1};
int pin_cols[nCols] = {C1, C2, C3, C4};

//define inital button values
int buttonPriorValue = 1;
int buttonValue = 0;

//define alarm toggle variable
int alarmEN = 0;

//define alarm to 00:00 to start
String alarm_time = "00:00";

//this function returns the char that is pressed on the keypad
char keypadListen(){
  //define inital keyvalue
  char keyValue = 0;

 //set all columns high
  digitalWrite(C1, HIGH);
```

```
  digitalWrite(C2, HIGH);
  digitalWrite(C3, HIGH);
  digitalWrite(C4, HIGH);

  //set each column low 1 by 1
  for(int col = 0; col < nCols; col++){
    digitalWrite(pin_cols[col], LOW);

    //scan rows to see if key is pressed
    for(int row = 0; row < nRows; row++){
      //if row is low return character
      buttonValue = debounce(pin_rows[row], buttonPriorValue);
      if(buttonValue == 0){
        if(buttonValue != buttonPriorValue){
          buttonPriorValue = buttonValue;
        }
        //map button press to keymap
        keyValue = Keymap[row][col];
        delay(100);
      }

    }
    //set column back high for next iteration
    digitalWrite(pin_cols[col], HIGH);

  }
  //return button that was pressed
  return keyValue;
}

int debounce(int PIN, int last_val){
  //read value
  int new_val = digitalRead(PIN);
  //check if is it not equal to last value
  if(new_val != last_val){
    delay(10);
    new_val = digitalRead(PIN);
  }
  return new_val;
```

```
}


// This function will create a math problem, consisting of a solution and two expressions.
// The user will need to select the correct expression that matches the given solution.
// Inputs to the function will be number of problems, maybe difficulty???
bool mathProblem() {
  randomSeed(analogRead(A0));
  lcd.clear();
// create an expression with 3-4 operands that add or subtract to a number between x and y (probably 1ish and 20ish)
  // use random() to generate the operands, and to determine if + or - between them
  int numOperands = 4;
  int operands[numOperands];

  // create array of random operands
  for (int i = 0; i < numOperands; i++) {
    operands[i] = random(1, 10) * (random(0, 2) == 0 ? -1 : 1);
  }

  // solve the created problem
  int solution;
  for (int i = 0; i < numOperands; i++) {
    solution += operands[i];
  }

  // create string of correct expression
  String correctString = "";
  for (int i = 0; i < numOperands; i++) {
    if (operands[i] > 0) {
      correctString += "+"; // Add a + sign for positive operands
    }
    correctString += String(operands[i]); // Append the operand value
  }


  Serial.print("Correct: ");
  Serial.println(correctString);
```

```
// change two of the operands to create a wrong expression
// by generating two random indexes
int index1 = random(0, numOperands);
int index2;
do {
  index2 = random(0, numOperands); // Ensure the second index is different
}

while (index2 == index1);

// assign new random values to the generated indexes
operands[index1] = random(1, 10) * (random(0, 2) == 0 ? -1 : 1);
operands[index2] = random(1, 10) * (random(0, 2) == 0 ? -1 : 1);

// Create a string of the updated operands with + for positive numbers
String wrongString = "";
for (int i = 0; i < numOperands; i++) {
  if (operands[i] > 0) {
    wrongString += "+"; // Add a + sign for positive operands
  }
  wrongString += String(operands[i]); // Append the operand value
}

Serial.print("Wrong: ");
Serial.println(wrongString);
Serial.print("Solution: ");
Serial.println(solution);

// next, roll random(0,2) for correct expression's cursor row
int cursorRowCorrect = random(0,2);

// set wrong expression to opposite of rolled
int cursorRowWrong = 1 - cursorRowCorrect;

char ansRow;
char otherRow;
// indicate which answer is correct for button press on keypad
if (cursorRowCorrect == 0) {
  ansRow = 'A';
```

```
  otherRow = 'B';
}

else {
  otherRow = 'A';
  ansRow = 'B';
}

// printing expressions to LCD
  // print correct expression
  lcd.setCursor(0,cursorRowCorrect);
  lcd.print(ansRow);
  lcd.print(":");
  lcd.print(correctString);

  // print wrong expression
  lcd.setCursor(0,cursorRowWrong);
  lcd.print(otherRow);
  lcd.print(":");
  lcd.print(wrongString);

  //print solution in top right corner
  lcd.setCursor(13,0);
  lcd.print(solution);

// set key to zero initially
char key = 0;
char keyAns;

// wait for a key press
while (!key) key = keypadListen(); {

  // validate key press
  if (key == 'A' || key == 'B') {
    keyAns = key;
  }
  else {
    key = 0; // If invalid key, repeat the step
  }
```

```
  }

  bool correct = 0;

  // check for correct answer
  if (keyAns == ansRow) {
    correct = 1;
    Serial.print("Correct: ");
    Serial.println(correct);
  }

  return correct;

}

void setup() {
  delay(1000);
  // put your setup code here, to run once:
  //setup for LCD
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0,0);

  //set columns as outputs
  pinMode(C1, OUTPUT);
  pinMode(C2, OUTPUT);
  pinMode(C3, OUTPUT);
  pinMode(C4, OUTPUT);

  //set rows as input pullup
  pinMode(R1, INPUT_PULLUP);
  pinMode(R2, INPUT_PULLUP);
  pinMode(R3, INPUT_PULLUP);
  pinMode(R4, INPUT_PULLUP);

  //set buzzer as output
  pinMode(buzzer_pin, OUTPUT);

  //setup serial monitor
```

```
  Serial.begin(9600);

  //setup RTC
  URTCLIB_WIRE.begin();
}

//this function gets the time from the RTC, displays it to the LCD, and returns a time string to be compared with the alarm string
String get_time(){
  //refresh to get recent time
  rtc.refresh();

  //get hours, mins, secs
  int hours = rtc.hour();
  int mins = rtc.minute();
  int secs = rtc.second();

  //store time to be compared for alarm
  String time = "";

  //set cursor to 4, 0
  lcd.setCursor(4, 0);

  //make hh format
  if (hours < 10){
    lcd.print("0"); //print to LCD
    time = time + "0"; //append to time string
  }
  lcd.print(hours); //print to LCD
  time = time + String(hours); //append to time string

  //add ":"
  lcd.print(":"); //print to LCD
  time = time + ":";  //append to time string

  //make mm format
  if (mins < 10){
    lcd.print("0"); //print to LCD
    time = time + "0"; //append to time string
```

```
  }
  lcd.print(mins); //print to LCD
  time = time + String(mins); //append to time string

  //print ss to LCD
  lcd.print(":");
  if (secs < 10){
    lcd.print("0");
  }
  lcd.print(secs);

  //return time string
  return time;
}

//this function sets the time from user inputs to the keypad
void set_time(){
  //set hours
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Hours: "); //prompt user for hrs

  int h = 0; //variable to store hh
  //get two char from user
  for (int i = 0; i < 2; i++) {
    char key = 0;

    // Wait for a key press
    while (!key) key = keypadListen();

    // Check if the key is a valid digit
    if (key >= '0' && key <= '9') {
      lcd.print(key); // Display the digit on the LCD
      h = h * 10 + (key - '0'); // Accumulate h
    }
    else {
      i--; // If invalid key, repeat the step
    }
  }
```

```
// Ensure valid range for hours
if (h < 0 || h > 23) {
  lcd.clear();
  lcd.print("Invalid Hours!");
  delay(1500);
  lcd.clear();
  return; // Exit if invalid input
}

//set mins
lcd.setCursor(0, 1);
lcd.print("Mins: "); //prompt user for mins

int m = 0; //variable to store mm
//get two char from user
for (int i = 0; i < 2; i++) {
  char key = 0;

  // Wait for a key press
  while (!key) key = keypadListen();

  // Check if the key is a valid digit
  if (key >= '0' && key <= '9') {
    lcd.print(key); // Display the digit on the LCD
    m = m * 10 + (key - '0'); // Accumulate m
  }
  else {
    i--; // If invalid key, repeat the step
  }
}

// Ensure valid range for hours
if (m < 0 || m > 59) {
  lcd.clear();
  lcd.print("Invalid Mins!");
  delay(1500);
  lcd.clear();
  return; // Exit if invalid input
```

```
  }
  //set secs to 00
  int s = 0;

  //send time to RTC
  rtc.set(s, m, h, 0, 0, 0, 0); //(second, minute, hour, dayOfWeek, dayOfMonth, month, year)
  lcd.clear(); //clear the LCD
}

//this function sets the alarm from user inputs to the keypad and retuens an alarm string
String set_alarm() {
  // Initialize the LCD display
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("A Hours: "); // Prompt user for hours

  int A_h = 0; // Variable to store hours
  // Get two characters for hours
  for (int i = 0; i < 2; i++) {
    char key = 0;

    // Wait for a key press
    while (!key) key = keypadListen();

    // Check if the key is a valid digit
    if (key >= '0' && key <= '9') {
      lcd.print(key); // Display the digit on the LCD
      A_h = A_h * 10 + (key - '0'); // Accumulate the number
    } else {
      i--; // If invalid key, repeat the step
    }
  }

  // Ensure valid range for hours
  if (A_h < 0 || A_h > 23) {
    lcd.clear();
    lcd.print("Invalid Hours!");
    delay(1500);
    lcd.clear();
```

```
  return "00:00"; // Exit if invalid input
}

lcd.setCursor(0,1);
lcd.print("A Mins: "); // Prompt user for minutes

int A_m = 0; // Variable to store minutes
// Get two characters for minutes
for (int i = 0; i < 2; i++) {
  char key = 0;

  // Wait for a key press
  while (!key) key = keypadListen();

  // Check if the key is a valid digit
  if (key >= '0' && key <= '9') {
    lcd.print(key); // Display the digit on the LCD
    A_m = A_m * 10 + (key - '0'); // Accumulate the number
  } else {
    i--; // If invalid key, repeat the step
  }
}

// Ensure valid range for minutes
if (A_m < 0 || A_m > 59) {
  lcd.clear();
  lcd.print("Invalid Mins!");
  delay(1500);
  lcd.clear();
  return "00:00"; // Exit if invalid input
}

// Format and return the alarm time
char alarmTime[6];
sprintf(alarmTime, "%02d:%02d", A_h, A_m); // Ensure two-digit formatting
delay(100);

lcd.clear();
return String(alarmTime);
```

```
}

// void buzzer() {
//   digitalWrite(buzzer_pin, HIGH);
//   delay(500);
//   digitalWrite(buzzer_pin, LOW);
//   delay(250);
// }

void buzzer_ON() {
  digitalWrite(buzzer_pin, HIGH);
}

void buzzer_OFF() {
  digitalWrite(buzzer_pin, LOW);
}


void disp_alarm(String alarm, int state){
  lcd.setCursor(0,1);

  lcd.print("A: ");
  lcd.print(alarm);

  if (state == 1){
    lcd.setCursor(12, 1);
    lcd.print(" ON");
  }

  if (state == 0){
    lcd.setCursor(12, 1);
    lcd.print("OFF");
  }
}

int toggle_alarm(){
  //toggle the alarm and return
  alarmEN = !alarmEN;
  return alarmEN;
```

```
}

bool alarmExit = 0; // alarm exit flag

void loop() {
  // put your main code here, to run repeatedly:
  //read keypad press
  char key = keypadListen();
  Serial.println(key);

  //set the time if D is pressed
  if (key == 'D'){
    set_time();
  }

  //set the alarm if A is pressed
  if (key == 'A'){
    alarm_time = set_alarm();
  }

  //toggle alarm if B is pressed
  int alarm_state;
  if (key == 'B'){
    alarm_state = toggle_alarm();
  }

  //display current time
  String current_time = get_time();
  //display alarm time
  disp_alarm(alarm_time, alarm_state);

  while(current_time == alarm_time && alarm_state == 1 && alarmExit == 0){
    lcd.clear();
    lcd.setCursor(0,0);
    bool correctFlag = 0;

    while (correctFlag == 0) {
      buzzer_ON();
```

```
    correctFlag = mathProblem();
   }

   alarmExit = 1;
   buzzer_OFF();
   lcd.clear();
  }

  if (alarmExit == 1 && current_time != alarm_time) {
   alarmExit = 0; // Reset the flag for the next alarm cycle
  }
}
raries to control the LCD and RTC
#include <LiquidCrystal_I2C.h>
#include "uRTCLib.h"

//set up LCD screen
LiquidCrystal_I2C lcd(0x27,16,2);

// uRTCLib rtc;
uRTCLib rtc(0x68);

//define column pins for keypad
int C1 = 6;
int C2 = 7;
int C3 = 8;
int C4 = 9;

//define row pins for keypad
int R1 = 2;
int R2 = 3;
int R3 = 4;
int R4 = 5;

//these values are used in the keypadListen function in the for loops
const int nRows = 4;
const int nCols = 4;
```

```
//buzzer pin
int buzzer_pin = 11;

//define keymap for keypad
char Keymap[nRows][nCols] = {
  {'A', 'B', 'C', 'D'},
  {'3', '6', '9', '#'},
  {'2', '5', '8', '0'},
  {'1', '4', '7', '*'}
};


//define corresponding pins
int pin_rows[nRows] = {R4, R3, R2, R1};
int pin_cols[nCols] = {C1, C2, C3, C4};

//define inital button values
int buttonPriorValue = 1;
int buttonValue = 0;

//define alarm toggle variable
int alarmEN = 0;

//define alarm to 00:00 to start
String alarm_time = "00:00";

//this function returns the char that is pressed on the keypad
char keypadListen(){
  //define inital keyvalue
  char keyValue = 0;

 //set all columns high
  digitalWrite(C1, HIGH);
  digitalWrite(C2, HIGH);
  digitalWrite(C3, HIGH);
  digitalWrite(C4, HIGH);

  //set each column low 1 by 1
  for(int col = 0; col < nCols; col++){
```

```
    digitalWrite(pin_cols[col], LOW);

    //scan rows to see if key is pressed
    for(int row = 0; row < nRows; row++){
      //if row is low return character
      buttonValue = debounce(pin_rows[row], buttonPriorValue);
      if(buttonValue == 0){
        if(buttonValue != buttonPriorValue){
          buttonPriorValue = buttonValue;
        }
        //map button press to keymap
        keyValue = Keymap[row][col];
        delay(100);
      }

    }
    //set column back high for next iteration
    digitalWrite(pin_cols[col], HIGH);

  }
  //return button that was pressed
  return keyValue;
}

int debounce(int PIN, int last_val){
  //read value
  int new_val = digitalRead(PIN);
  //check if is it not equal to last value
  if(new_val != last_val){
    delay(10);
    new_val = digitalRead(PIN);
  }
  return new_val;
}


// This function will create a math problem, consisting of a solution and two expressions.
// The user will need to select the correct expression that matches the given solution.
// Inputs to the function will be number of problems, maybe difficulty???
```

```
bool mathProblem() {
  randomSeed(analogRead(A0));
  lcd.clear();
// create an expression with 3-4 operands that add or subtract to a number between x and y (probably 1ish and 20ish)
  // use random() to generate the operands, and to determine if + or - between them
  int numOperands = 4;
  int operands[numOperands];

  // create array of random operands
  for (int i = 0; i < numOperands; i++) {
    operands[i] = random(1, 10) * (random(0, 2) == 0 ? -1 : 1);
  }

  // solve the created problem
  int solution;
  for (int i = 0; i < numOperands; i++) {
    solution += operands[i];
  }

  // create string of correct expression
  String correctString = "";
  for (int i = 0; i < numOperands; i++) {
    if (operands[i] > 0) {
      correctString += "+"; // Add a + sign for positive operands
    }
    correctString += String(operands[i]); // Append the operand value
  }


  Serial.print("Correct: ");
  Serial.println(correctString);

  // change two of the operands to create a wrong expression
  // by generating two random indexes
  int index1 = random(0, numOperands);
  int index2;
  do {
    index2 = random(0, numOperands); // Ensure the second index is different
```

```
}

while (index2 == index1);

// assign new random values to the generated indexes
operands[index1] = random(1, 10) * (random(0, 2) == 0 ? -1 : 1);
operands[index2] = random(1, 10) * (random(0, 2) == 0 ? -1 : 1);

// Create a string of the updated operands with + for positive numbers
String wrongString = "";
for (int i = 0; i < numOperands; i++) {
  if (operands[i] > 0) {
    wrongString += "+"; // Add a + sign for positive operands
  }
  wrongString += String(operands[i]); // Append the operand value
}

Serial.print("Wrong: ");
Serial.println(wrongString);
Serial.print("Solution: ");
Serial.println(solution);

// next, roll random(0,2) for correct expression's cursor row
int cursorRowCorrect = random(0,2);

// set wrong expression to opposite of rolled
int cursorRowWrong = 1 - cursorRowCorrect;

char ansRow;
char otherRow;
// indicate which answer is correct for button press on keypad
if (cursorRowCorrect == 0) {
  ansRow = 'A';
  otherRow = 'B';
}

else {
  otherRow = 'A';
  ansRow = 'B';
```

```
}

// printing expressions to LCD
  // print correct expression
  lcd.setCursor(0,cursorRowCorrect);
  lcd.print(ansRow);
  lcd.print(":");
  lcd.print(correctString);

  // print wrong expression
  lcd.setCursor(0,cursorRowWrong);
  lcd.print(otherRow);
  lcd.print(":");
  lcd.print(wrongString);

  //print solution in top right corner
  lcd.setCursor(13,0);
  lcd.print(solution);

// set key to zero initially
char key = 0;
char keyAns;

// wait for a key press
while (!key) key = keypadListen(); {

  // validate key press
  if (key == 'A' || key == 'B') {
    keyAns = key;
  }
  else {
    key = 0; // If invalid key, repeat the step
  }
}

bool correct = 0;

// check for correct answer
if (keyAns == ansRow) {
```

```
    correct = 1;
    Serial.print("Correct: ");
    Serial.println(correct);
  }

  return correct;

}

void setup() {
  delay(1000);
  // put your setup code here, to run once:
  //setup for LCD
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0,0);

  //set columns as outputs
  pinMode(C1, OUTPUT);
  pinMode(C2, OUTPUT);
  pinMode(C3, OUTPUT);
  pinMode(C4, OUTPUT);

  //set rows as input pullup
  pinMode(R1, INPUT_PULLUP);
  pinMode(R2, INPUT_PULLUP);
  pinMode(R3, INPUT_PULLUP);
  pinMode(R4, INPUT_PULLUP);

  //set buzzer as output
  pinMode(buzzer_pin, OUTPUT);

  //setup serial monitor
  Serial.begin(9600);

  //setup RTC
  URTCLIB_WIRE.begin();
}
```

```
//this function gets the time from the RTC, displays it to the LCD, and returns a time string to be compared with the alarm string
String get_time(){
  //refresh to get recent time
  rtc.refresh();

  //get hours, mins, secs
  int hours = rtc.hour();
  int mins = rtc.minute();
  int secs = rtc.second();

  //store time to be compared for alarm
  String time = "";

  //set cursor to 4, 0
  lcd.setCursor(4, 0);

  //make hh format
  if (hours < 10){
    lcd.print("0"); //print to LCD
    time = time + "0"; //append to time string
  }
  lcd.print(hours); //print to LCD
  time = time + String(hours); //append to time string

  //add ":"
  lcd.print(":"); //print to LCD
  time = time + ":";  //append to time string

  //make mm format
  if (mins < 10){
    lcd.print("0"); //print to LCD
    time = time + "0"; //append to time string
  }
  lcd.print(mins); //print to LCD
  time = time + String(mins); //append to time string

  //print ss to LCD
  lcd.print(":");
```

```
  if (secs < 10){
    lcd.print("0");
  }
  lcd.print(secs);

  //return time string
  return time;
}

//this function sets the time from user inputs to the keypad
void set_time(){
  //set hours
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Hours: "); //prompt user for hrs

  int h = 0; //variable to store hh
  //get two char from user
  for (int i = 0; i < 2; i++) {
    char key = 0;

    // Wait for a key press
    while (!key) key = keypadListen();

    // Check if the key is a valid digit
    if (key >= '0' && key <= '9') {
      lcd.print(key); // Display the digit on the LCD
      h = h * 10 + (key - '0'); // Accumulate h
    }
    else {
      i--; // If invalid key, repeat the step
    }
  }

  // Ensure valid range for hours
  if (h < 0 || h > 23) {
    lcd.clear();
    lcd.print("Invalid Hours!");
    delay(1500);
```

```
  lcd.clear();
  return; // Exit if invalid input
}

//set mins
lcd.setCursor(0, 1);
lcd.print("Mins: "); //prompt user for mins

int m = 0; //variable to store mm
//get two char from user
for (int i = 0; i < 2; i++) {
  char key = 0;

  // Wait for a key press
  while (!key) key = keypadListen();

  // Check if the key is a valid digit
  if (key >= '0' && key <= '9') {
    lcd.print(key); // Display the digit on the LCD
    m = m * 10 + (key - '0'); // Accumulate m
  }
  else {
    i--; // If invalid key, repeat the step
  }
}

// Ensure valid range for hours
if (m < 0 || m > 59) {
  lcd.clear();
  lcd.print("Invalid Mins!");
  delay(1500);
  lcd.clear();
  return; // Exit if invalid input
}
//set secs to 00
int s = 0;

//send time to RTC
rtc.set(s, m, h, 0, 0, 0, 0); //(second, minute, hour, dayOfWeek, dayOfMonth, month, year)
```

```
 lcd.clear(); //clear the LCD
}

//this function sets the alarm from user inputs to the keypad and retuens an alarm string
String set_alarm() {
 // Initialize the LCD display
 lcd.clear();
 lcd.setCursor(0, 0);
 lcd.print("A Hours: "); // Prompt user for hours

 int A_h = 0; // Variable to store hours
 // Get two characters for hours
 for (int i = 0; i < 2; i++) {
  char key = 0;

  // Wait for a key press
  while (!key) key = keypadListen();

  // Check if the key is a valid digit
  if (key >= '0' && key <= '9') {
   lcd.print(key); // Display the digit on the LCD
   A_h = A_h * 10 + (key - '0'); // Accumulate the number
  } else {
   i--; // If invalid key, repeat the step
  }
 }

 // Ensure valid range for hours
 if (A_h < 0 || A_h > 23) {
  lcd.clear();
  lcd.print("Invalid Hours!");
  delay(1500);
  lcd.clear();
  return "00:00"; // Exit if invalid input
 }

 lcd.setCursor(0,1);
 lcd.print("A Mins: "); // Prompt user for minutes
```

```
  int A_m = 0; // Variable to store minutes
  // Get two characters for minutes
  for (int i = 0; i < 2; i++) {
    char key = 0;

    // Wait for a key press
    while (!key) key = keypadListen();

    // Check if the key is a valid digit
    if (key >= '0' && key <= '9') {
      lcd.print(key); // Display the digit on the LCD
      A_m = A_m * 10 + (key - '0'); // Accumulate the number
    } else {
      i--; // If invalid key, repeat the step
    }
  }

  // Ensure valid range for minutes
  if (A_m < 0 || A_m > 59) {
    lcd.clear();
    lcd.print("Invalid Mins!");
    delay(1500);
    lcd.clear();
    return "00:00"; // Exit if invalid input
  }

  // Format and return the alarm time
  char alarmTime[6];
  sprintf(alarmTime, "%02d:%02d", A_h, A_m); // Ensure two-digit formatting
  delay(100);

  lcd.clear();
  return String(alarmTime);
}

// void buzzer() {
//   digitalWrite(buzzer_pin, HIGH);
//   delay(500);
//   digitalWrite(buzzer_pin, LOW);
```

```
//   delay(250);
// }

void buzzer_ON() {
  digitalWrite(buzzer_pin, HIGH);
}

void buzzer_OFF() {
  digitalWrite(buzzer_pin, LOW);
}


void disp_alarm(String alarm, int state){
  lcd.setCursor(0,1);

  lcd.print("A: ");
  lcd.print(alarm);

  if (state == 1){
    lcd.setCursor(12, 1);
    lcd.print(" ON");
  }

  if (state == 0){
    lcd.setCursor(12, 1);
    lcd.print("OFF");
  }
}

int toggle_alarm(){
  //toggle the alarm and return
  alarmEN = !alarmEN;
  return alarmEN;

}

bool alarmExit = 0; // alarm exit flag

void loop() {
```

```
// put your main code here, to run repeatedly:
//read keypad press
char key = keypadListen();
Serial.println(key);

//set the time if D is pressed
if (key == 'D'){
  set_time();
}

//set the alarm if A is pressed
if (key == 'A'){
  alarm_time = set_alarm();
}

//toggle alarm if B is pressed
int alarm_state;
if (key == 'B'){
  alarm_state = toggle_alarm();
}

//display current time
String current_time = get_time();
//display alarm time
disp_alarm(alarm_time, alarm_state);

while(current_time == alarm_time && alarm_state == 1 && alarmExit == 0){
  lcd.clear();
  lcd.setCursor(0,0);
  bool correctFlag = 0;

  while (correctFlag == 0) {
    buzzer_ON();

    correctFlag = mathProblem();
  }

  alarmExit = 1;
  buzzer_OFF();
```

```
    lcd.clear();
  }

  if (alarmExit == 1 && current_time != alarm_time) {
    alarmExit = 0; // Reset the flag for the next alarm cycle
  }
}
```