Lab 03

Keypad, Seven Segment, and LCD

Logan Barber, Ian Nail

February 11, 2022

Ian Nail

Logan Barber

ME-4370 - Stephen Canfield

1 Executive Summary

In this lab, the objective is to write a program to use hardware pushbuttons for a passcode program that requires the user to make a passcode and then enter a passcode.

The minimum objectives were:

1. Prompt the user to enter a 4 digit passcode
2. The passcode is entered using hard wired pushbuttons.
3. Save the 4 numbers into memory
4. Once the passcode has been entered the safe needs to be closed. For a recipe book, this can be a display, "Recipe locked".
5. Prompt the user to press any switch to unlock the safe.
6. Prompt the user enter their passcode
7. Check the passcode, open if correct, do not open if incorrect.
8. You should allow the user to enter the entire passcode before checking it. Nice features might be to show a character as the user enters the passcode.
9. Allow the user up to three attempts, if they fail on the third attempt, display some failure message, and end the program without opening the safe.
10. When the safe is opened, you can continue to the recipe.

The seven segment 4 display for the timer and the LCD display program from Lab02 was carried over as the basis for implementing the keypad.

To get an input from the keypad, the button matrix is scanned one row and column at a time as the program searches for a button press. Two 'for loops' are used to iteratively do the row and column search. The keypad has 16 buttons, so 4 rows and 4 columns needed 8 bits. PORTC was used with 4 bits as inputs and 4 bits as outputs. The inputs bits had the pull-up resistor enabled. The pull-up resistors were used for reading active lows when a button press occurs. The 16 buttons are scanned so quickly that it is very challenging to not have a button press registered. Debouncing code is used to to hold the processor in an infinite loop until the button is released. This prevents a single button press from being interpreted as multiple button presses. A 'do while' loop is used for setting the password and another 'do while' loop used for checking the entered password matched the password that had been set. After three failed password attempts the program gives a message and fails to continue. While the password is being set or entered, the LCD screen shows the characters from the button presses so the user can see that each button was registered. The pound sign is used to end the password that is entered. If a password is successful the program will continue on to give the recipe instructions and the timer becomes available to use.

AutoCad was used to draw up the circuit diagram of all the connections, components, and devices used.

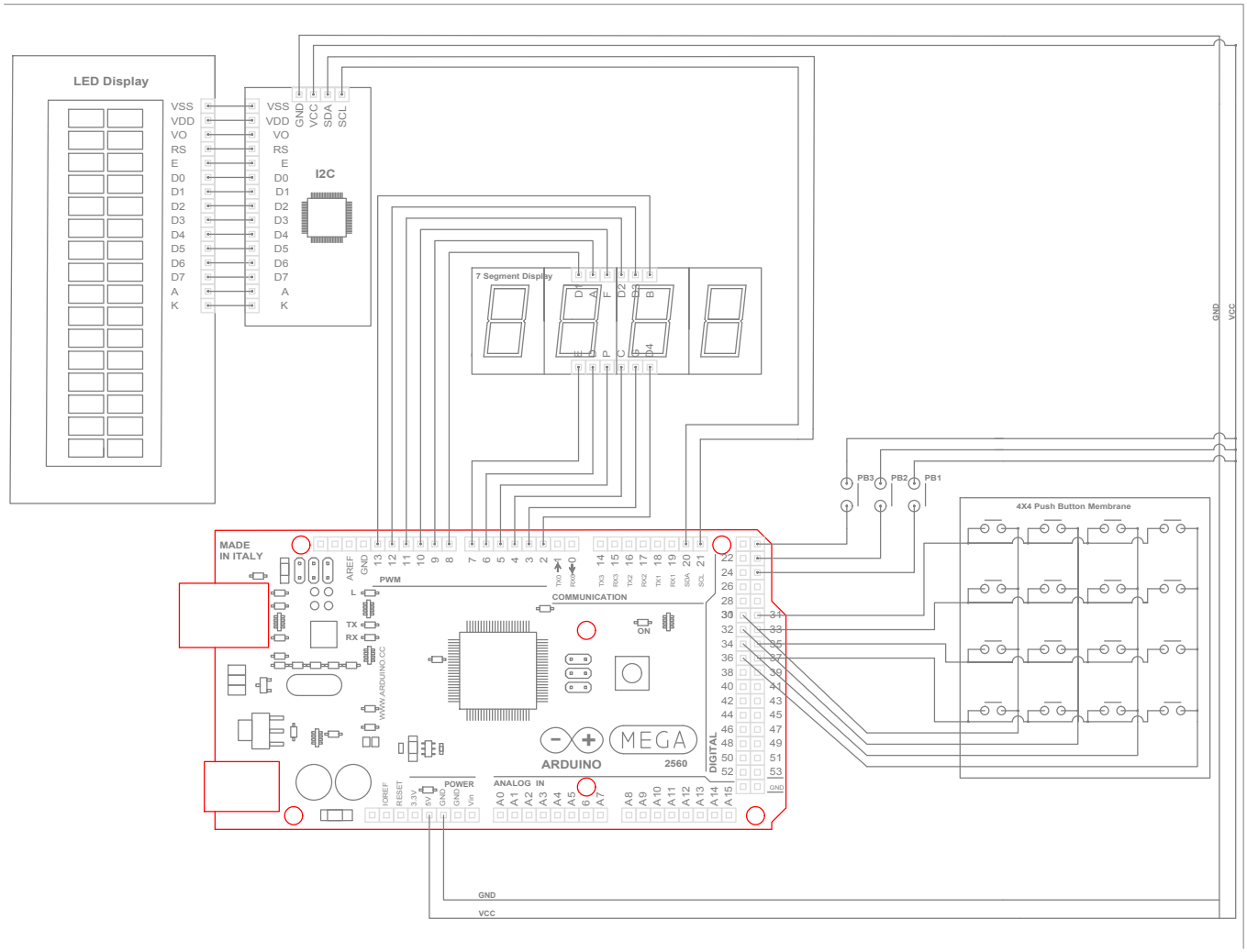

Figure 1: Circuit Diagram

2 Source Code

```
/* AUTHORS: A. LOGAN BARBER; IAN NAIL
 * FILE NAME: Lab03.ino
 * LAST UPDATED: 28 JANUARY 2022
 *
 *  PURPOSE: THIS FILE IS THE MAIN FILE FOR DISPLAYING A RECIPE ON AN LCD AND BEING ABLE TO SCROLL USING TWO BUTTONS.
 *       THIS FILE ALSO UTILIZES AND 4-7 SEGMENT DISPLAY AS A 10 MINUTE TIMER AND A KEYPAD FOR PASSWORD PROTECTION
 */

/*
 * BUTTON0:
 * DIGITAL PIN: 22
 * PORT: A
 * PORT PIN: 0
 */

 /*
 * BUTTON1:
 * DIGITAL PIN: 23
 * PORT: A
 * PORT PIN: 1
 */

 /*
 * BUTTON2:
 * DIGITAL PIN: 24
 * PORT: A
 * PORT PIN: 2
 */

 /*
  * 7 - SEGMENT DECODER
  *
  * D0:
  *       DIGITAL PIN: 53
  *       PORT: B
  *       PORT PIN: 0
  *
  * D1:
  *       DIGITAL PIN: 52
  *       PORT: B
  *       PORT PIN: 1
  *
  * D2:
  *       DIGITAL PIN: 51
  *       PORT: B
  *       PORT PIN: 2
  *
  * D3:
  *       DIGITAL PIN: 50
  *       PORT: B
  *       PORT PIN: 3
  */

  /*
   * DEMULTIPLEXER
```

```
   *
 * A:
   *       DIGITAL PIN: 10
   *       PORT: B
   *       PORT PIN: 4
   *
   * B:
   *       DIGITAL PIN: 11
   *       PORT: B
   *       PORT PIN: 5
   */

  /*
   * KEY PAD:
   *
   * PIN 1:
   *       DIGITAL PIN: 37
   *       PORT: C
   *       PORT PIN: 0
   *
   * PIN 2:
   *       DIGITAL PIN: 36
   *       PORT: C
   *       PORT PIN: 1
   *
   * PIN 3:
   *       DIGITAL PIN: 35
   *       PORT: C
   *       PORT PIN: 2
   *
   * PIN 4:
   *       DIGITAL PIN: 34
   *       PORT: C
   *       PORT PIN: 3
   *
   * PIN 5:
   *       DIGITAL PIN: 33
   *       PORT: C
   *       PORT PIN: 4
   *
   * PIN 6:
   *       DIGITAL PIN: 32
   *       PORT: C
   *       PORT PIN: 5
   *
   * PIN 7:
   *       DIGITAL PIN: 31
   *       PORT: C
   *       PORT PIN: 6
   *
   * PIN 8:
   *       DIGITAL PIN: 30
   *       PORT: C
   *       PORT PIN: 7
   */

// INCLUDE LIBRARIES
#include <LiquidCrystal_I2C.h>
#include <stdio.h>

```

```
#define TRUE 0x01
#define FALSE 0x00

// DEFINE MACROS FOR LCD SERIAL
#define ADDRESS 0x27
#define LCDCOLS 16
#define LCDROWS 2

// TIMER PARAMTERS
uint8_t buttonState = 0;
uint8_t minutes = 6; //start time -> CAN CHANGE TO WHATEVER TIME YOU WANT
uint8_t seconds = 30;  //start time -> CAN CHANGE TO WHATEVER TIME YOU WANT
uint8_t totalMinutes = 0;
uint8_t minutesTens = 0;
uint8_t minutesOnes = 0;
uint8_t secondsTens = 0;
uint8_t secondsOnes = 0;
uint8_t secondsTemp = 0;
float totalSeconds = minutes*60 + seconds;
float totalMilliseconds = totalSeconds*1000;
float totalMicroseconds = totalMilliseconds*1000;

// DEFINE NUMBERS FOR 7 SEGMENT DISPLAY
#define ZERO 0x00
#define ONE 0x01
#define TWO 0x02
#define THREE 0x03
#define FOUR 0x04
#define FIVE 0x05
#define SIX 0x06
#define SEVEN 0x07
#define EIGHT 0x08
#define NINE 0x09

// DEFINE KEYPAD ITEMS
#define ONE_PAD 0x77 // 0b01110111
#define TWO_PAD 0x7B // 0b01111011
#define THREE_PAD 0x7D // 0b01111101
#define A_PAD 0x7E // 0b01111110
#define FOUR_PAD 0xB7 // 0b10110111
#define FIVE_PAD 0xBB // 0b10111011
#define SIX_PAD 0xBD // 0b10111101
#define B_PAD 0xBE // 0b10111110
#define SEVEN_PAD 0xD7 // 0b11010111
#define EIGHT_PAD 0xDB // 0b11011011
#define NINE_PAD 0xDD //0b11011101
#define C_PAD 0xDE // 0b11011110
#define STAR_PAD 0xE7 // 0b11100111
#define ZERO_PAD 0xEB // 0b11101011
#define POUND_PAD 0xED // 0b11101101
#define D_PAD 0xEE // 0b11101110

// Keypad Parameters
const byte ROWS = 4; // number of rows on keypad
const byte COLS = 4; // number of columns on keypad
char keys[ROWS][COLS] = { // The buttons on the keypad
{'1','2','3', 'A'},
{'4','5','6', 'B'},
{'7','8','9', 'C'},
{'*','0','#', 'D'}};
```

```
int passwordSuceed = FALSE;
char keyResult = 'F';
int password = '0';

// DEFINE LETTERS FOR 7 SEGMENT DISPLAY
#define ZERO 0x00
#define ONE 0x01
#define TWO 0x02
#define THREE 0x03
#define FOUR 0x04
#define FIVE 0x05
#define SIX 0x06
#define SEVEN 0x07
#define EIGHT 0x08
#define NINE 0x09

// DEFINE THE DISPLAY SELECTION NUMBERS
const uint8_t D1 = 0x0F; // 0b00001111
const uint8_t D2 = 0x1F; // 0b00011111
const uint8_t D3 = 0x2F; // 0b00101111
const uint8_t D4 = 0x3F; // 0b00111111
const uint8_t arrD[4] = {D1, D2, D3, D4};

// MESSAGE TO PRINT
char password0[] = "Enter Password: ";
char message0[] = "Double Chocolate";
char message1[] = "Flower Brownies";
char message2[] = "1/2 Cup";
char message3[] = "Unsalted Butter";
char message4[] = "1 Gram";
char message5[] = "Flower";
char message6[] = "1/4 Cup ";
char message7[] = "Chocolate Chips";
char message8[] = "1 Tablespoon";
char message9[] = "Molasses";
char message10[] = "1 Teaspoon";
char message11[] = "Vanilla Extract";
char message12[] = "2 Large Eggs";
char message13[] = " ";
char message14[] = "1/4 Teaspoon";
char message15[] = "Kosher Salt";
char message16[] = "3/4 Cup All-";
char message17[] = "Purpose Flour";
char message18[] = "Bake for 10";
char message19[] = "minutes";
const uint8_t msgArrSize = 20;
char* msgArr[msgArrSize] = {message0, message1, message2, message3, message4,
    message5, message6, message7, message8, message9, message10, message11,
   message12,
    message13, message14, message15, message16, message17, message18, message19
   };

// INDEX VARIABLES
uint8_t index = 0; // HOLDS INDEX FOR MESSAGE
uint8_t i = 0; // HOLDS INDEX IN for LOOPS FOR SCROLLING
uint8_t t = 0; // HOLDS INDEX IN for LOOP FOR THE TIMER

// CREATE LiquidCrystal OBJECT
LiquidCrystal_I2C lcd(ADDRESS, LCDCOLS, LCDROWS);

```

```
// RUN THIS PROGRAM
void setup()
{
      // INITIALIZE THE LCD SCREEN
      lcd.begin();
      // SETUP BUTTON PINS AS INPUTS
      DDRA = 0x00; // 0b00000000

      // ENABLE INTERNAL PULL-UP RESISTOR FOR BUTTONS
      PORTA = 0x07; // 0b00000111

      // SETUP PORT B AS OUTPUT FOR THE LCD
      DDRB = 0x7F; // 0b01111111
      PORTB = 0x00;

      // SET UP PORT C AS OUTPUT AND INPUT
      DDRC = 0x0F;
      PORTC = 0xFF; // ENABLE PULL UP RESISTOR FOR INPUTS ON PORTC

      // CALCULATE INDIVIDUAL DIGITS
      totalMinutes = totalSeconds/60;
      minutesTens = totalMinutes/10;
      minutesOnes = totalMinutes%10;
      secondsTemp = int(totalSeconds)%60;
      secondsTens = secondsTemp/10;
      secondsOnes = secondsTemp%10;

       // This function sets the password and lockes the recipe till the password
     is entered
       passwordSetandLock();
}
// LOOP FOREVER
void loop()
{
      // IF BUTTON0 IS LOW SCROLL DOWN
      if((PINA & 0x07) == 0x06)
      {
            // DEBOUNCE BUTTON0
            delay(100);
            if((PINA & 0x07) == 0x06)
            {
              scroll_down();
            }
      }

      // ELSE IF BUTTON1 IS LOW SCROLL DOWN
      if((PINA & 0x07) == 0x05)
      {
            // DEBOUNCE BUTTON2
            delay(100);
            if((PINA & 0x07) == 0x05)
            {
              scroll_up();
            }
      }

      // IF BUTTON 2 IS LOW CHANGE THE BUTTON STATE
      else if((PINA & 0x07) == 0x03)
      {
            delay(100);
```

```
292             if((PINA & 0x07) == 0x03)
                {
294                 switch(buttonState)
                    {
296                     case 0:
                            buttonState = 1;
298                         break;

300                     case 1:
                            buttonState = 2;
302                         break;

304                     case 2:
                            buttonState = 0;
306
                            // RESET TIME
308                         totalSeconds = minutes*60 + seconds;
                            totalMilliseconds = totalSeconds*1000;
310                         totalMicroseconds = totalMilliseconds*1000;

312                         // CALCULATE INDIVIDUAL DIGITS
                            totalMinutes = totalSeconds/60;
314                         minutesTens = totalMinutes/10;
                            minutesOnes = totalMinutes%10;
316                         secondsTemp = int(totalSeconds)%60;
                            secondsTens = secondsTemp/10;
318                         secondsOnes = secondsTemp%10;
                            break;
320                 }
                }
322         }

324         // RUN TIMER IF BUTTON STATE IS IN STATE 1
            if(buttonState == 1)
326         {
                // TIME CALCULATIONS
328             totalMicroseconds = totalMicroseconds - 2000; //totalMilliseconds++'
    for stopwatch
                totalMilliseconds = totalMicroseconds/1000;
330             totalSeconds = (totalMilliseconds/1000+1);

332             // CALCULATE INDIVIDUAL DIGITS
                totalMinutes = totalSeconds/60;
334             minutesTens = totalMinutes/10;
                minutesOnes = totalMinutes%10;
336             secondsTemp = int(totalSeconds)%60;
                secondsTens = secondsTemp/10;
338             secondsOnes = secondsTemp%10;
            }
340
            // TIMER
342         for(t = 0; t < 4; ++t)
            {
344             switch(t)
                {
346                 case 0:
                        PORTB &= 0x0F;
348                     PORTB |= arrD[t];
                        seven_seg_writeNumber(minutesTens);
350                     delayMicroseconds(500);
```

```
                    break;

                case 1:
                    PORTB &= 0x0F;
                    PORTB |= arrD[t];
                    seven_seg_writeNumber(minutesOnes);
                    PORTB |= 0x40;
                    delayMicroseconds(500);
                    break;

                case 2:
                    PORTB &= 0x0F;
                    PORTB |= arrD[t];
                    seven_seg_writeNumber(secondsTens);
                    delayMicroseconds(500);
                    break;

                case 3:
                    PORTB &= 0x0F;
                    PORTB |= arrD[t];
                    seven_seg_writeNumber(secondsOnes);
                    delayMicroseconds(500);
                    break;

                default:
                    break;
            }
        }
}

/*
 * TYPE: FUNCTION
 * NAME: scroll_up
 * RETURN: void
 * NUMBER OF PARAMETERS: 2
 * PARAMETER NAMES: char* messagePtr, uint8_t sizeOfArray
 * PURPOSE: THIS FUNCTION SCROLLS THROUGH THE RECEIPE DISPLAYED ON THE LCD
 */
void scroll_up()
{
    // DECREMENT INDEX BY ONE
    index -= 2;

    // CHECK THE BOUNDS OF INDEX (REMEMBER index IS UNSIGNED)
    if(index > (msgArrSize - 2))
        index = 0;

    // CLEAR THE LCD SCREEN AND PRINT MESSAGES TO THE LCD
    lcd.clear();
    for(i = 0; i < 2; ++i)
    {
        lcd.setCursor(0, i);
        delayMicroseconds(500);
        lcd.print(msgArr[index + i]);
        delayMicroseconds(500);
    }
}

/*
 * TYPE: FUNCTION
```

```
 * NAME: scroll_down
 * RETURN: void
 * NUMBER OF PARAMETERS: 2
 * PARAMETER NAMES: char* messagePtr, uint8_t sizeOfArray
 * PURPOSE: THIS FUNCTION SCROLLS THROUGH THE RECEIPE DISPLAYED ON THE LCD
 */
void scroll_down()
{
      // INCREMENT INDEX
      index += 2;

      // CHECK THE BOUNDS OF INDEX
      if(index > (msgArrSize - 2))
            index = msgArrSize - 2;

      // CLEAR THE LCD SCREEN AND PRINT MESSAGES TO THE LCD
      lcd.clear();
      for(i = 0; i < 2; ++i)
      {
            lcd.setCursor(0, i);
            delayMicroseconds(1000);
            lcd.print(msgArr[index + i]);
            delayMicroseconds(1000);
      }
}

/*
 * TYPE: FUNCTION
 * NAME: seven_seg_writeNumber
 * RETURN: void
 * NUMBER OF PARAMETERS: 1
 * PARAMETER NAMES: int x
 * PURPOSE: THIS FUNCTION PICKS THE NUMBER FOR THE LCD
 */
void seven_seg_writeNumber(int x) //changes value of number
{
      switch(x)
      {
            case 1:
                  PORTB &= 0b00110000;
                  PORTB |= ONE;
                  break;

            case 2:
                  PORTB &= 0b00110000;
                  PORTB |= TWO;
                  break;

            case 3:
                  PORTB &= 0b00110000;
                  PORTB |= THREE;
                  break;

            case 4:
                  PORTB &= 0b00110000;
                  PORTB |= FOUR;
                  break;

            case 5:
                  PORTB &= 0b00110000;
```

```
                PORTB |= FIVE;
                break;

            case 6:
                PORTB &= 0b00110000;
                PORTB |= SIX;
                break;

            case 7:
                PORTB &= 0b00110000;
                PORTB |= SEVEN;
                break;

            case 8:
                PORTB &= 0b00110000;
                PORTB |= EIGHT;
                break;

            case 9:
                PORTB &= 0b00110000;
                PORTB |= NINE;
                break;

            default:
                PORTB &= 0b00110000;
                PORTB |= ZERO;
                break;
        }
}

/*
 * TYPE: FUNCTION
 * NAME: keyReturn
 * RETURN: char
 * NUMBER OF PARAMETERS: 1
 * PARAMETER NAMES: void
 * PURPOSE: THIS returns the character value of the button pressed on the keypad
 */
char keyReturn(void)
{
    /* for loop execution */
    byte outputROWS[ROWS] = {0b00000001,0b00000010,0b00000100,0b00001000}; //
    last 4 bits of PORTC that will be toggled to check if a button is pressed
    byte inputCOLS[COLS] = {0b00010000,0b00100000,0b01000000,0b10000000}; //
    first 4 bits of PORTC that will be be read from PINC to see if the pullup
    resistor has been grounded
    byte a = 0;
    byte b = 0;
    int valuePINC_B = 0;

    for (a = 0; a < 4; a = a+1) //first 'for' loop to check the rows of the
    button matrix
    {
        PORTC ^=  outputROWS[a]; // toggle the bit low

        for (b = 0; b < 4; b = b+1) //second for loop to check the columns of
    the button matrix
        {
            valuePINC_B = (PINC & inputCOLS[b]) >> (b+4); // get value to check
    if the bit is low (low means a button has been pressed at row a and colomn b
```

```
526             if (valuePINC_B != 1) // if the value is not high that means a
    button press is detected
                {
528                 keyResult = keys[a][b];
                    delay(15); //Debouncing code
530                 while(1 != (PINC & inputCOLS[b]) >> (b+4));
                    {
532                 }
                }
534         }

536         PORTC ^= outputROWS[a]; //toggle the bit high
    }
538
    return keyResult;
540 }

542 /*
 * TYPE: FUNCTION
544  * NAME: passwordSetandLock
 * RETURN: void
546  * NUMBER OF PARAMETERS: 0
 * PARAMETER NAMES: n/a
548  * PURPOSE: THIS SETS AND LOCKES THE RECIPE TILL THE CORRECT PASSWORD IS ENTERED
    . THREE MAX TRIES TO ENTER CORRECT PASSWORD
 */
550 void passwordSetandLock()
{
552  // PROMPT USER FOR PASSWORD
     lcd.print(password0);
554  lcd.setCursor(0, 1);

556  // GET USER PASSWORD
     do
558  {
        keyReturn();
560     passwordSuceed = 0;

562     if (keyResult != 'F')
        {
564       password = password + keyResult;
          lcd.print(keyResult);
566       if (keyResult == '#')
          {
568         passwordSuceed = TRUE;
          }
570
          keyResult = 'F';
572     }
     } while(passwordSuceed == FALSE);
574
      lcd.clear();
576   lcd.print("RECIPE LOCKED");
      lcd.setCursor(0, 1);
578
     //GET USER TO ENTER PASSWORD TO UNLOCK RECIPE
580  int passwordTest = '0';
     int passwordTries = 0;
582  do
```

```
    {
584     keyReturn();
        passwordSuceed = 0;
586
588     if (keyResult != 'F')
        {
590       passwordTest = passwordTest + keyResult;
          lcd.print(keyResult);
592         if ((keyResult == '#') && (passwordTest == password))
            {
594           passwordSuceed = TRUE;
            }
596         if ((keyResult == '#') && (passwordTest != password))
            {
598           passwordSuceed = FALSE;
              passwordTest = '0';
600           lcd.clear();
              lcd.print("Try Again");
602           lcd.setCursor(0, 1);
              passwordTries++;
604             if(passwordTries >= 3)
                {
606                 lcd.clear();
                    lcd.print("NO MORE TRIES");
608                 lcd.setCursor(0, 1);
                    lcd.print("LEFT! ");
610                 while(1)
                    {
612                 }
                }
614           }
          keyResult = 'F';
        }
    } while(passwordSuceed == FALSE);
618
    // PRINT MESSAGE
620 lcd.clear();
    lcd.print(msgArr[0]);
622 lcd.setCursor(0, 1);
    lcd.print(msgArr[1]);
624 }
```

Lab03.ino