```
/*
 * MEMS1049_FinalProject.c
 *
 * Created: 4/19/2025 9:32:50 PM
 * Author : orhen
 */

#include <avr/io.h>
#define FREQ_CLK 1000000
#include <string.h>
#include <avr/interrupt.h>
#define lcd_port PORTD // We are using port D for the LCD pins (0 = register
select, 1 = enable, 4-7 = data)
#define lcd_EN 1       // enable pin on LCD
#define lcd_RS 0


//water weight value
char sensorvalue = 0;

// user input start variable
int gouser=0;

// amount variable
int foodamount = 0;


//breakbeam value
char beamvalue=0;

//interupt values
volatile int loop=0;
volatile int number_of_msec=30000;
volatile char register_B_setting;
volatile char timer_preload;
int flag = 0;

// lcd functions
void wait(volatile unsigned int);
void LCD_write(unsigned char cmd);
void LCD_init(void);
void LCD_command(char);
void LCD_data(char);
void LCD_print(char *str);
void LCD_gotoxy(unsigned char x, unsigned char y);

// Motor Function
void dispensefood (int);
void step_CW(void);
void step_CCW(void);



////////////////////////////////////////////////////////////////////////////////
//////////////////////

//interrupt when time is reached to feed dog
ISR(TIMER2_OVF_vect)
{
```

```
	loop++;
	if (loop==number_of_msec) { // if true, the desired time has elapsed
		// Timer keeps running but need to clear the interrupt flag, reset the
loop counter, and toggle the LED
		TIFR2 = 0x1<<TOV2;  // Clear TOV0 (note that this is an odd bit in that
it is cleared by writing a 1 to it)
		loop = 0; // reset count for another 1 second delay
		flag =1;
	}
}//end interrupt


////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
int main(void)
{// start of main

	// Pin Setups

	//break beam set up
	DDRC = 0<<PC3; // Make PB4  input bit for break beam
	//motor setup
	DDRB=1<<PB3;//set motor controls (B3 and B6) as outputs
	DDRB=1<<PB6;
	//weight sensor setup
	DDRC=0<<PC1;//set weight sensor as input
	// switch setup
	DDRB = 0<<PB0;
	DDRB = 0<<PB1;
	DDRB = 0<<PB2;
	// LCD set up
	DDRD = 0xFF; // Set PD0, PD1, and PD4-7 to output for the LCD
	// water set up
	DDRC = 1<<PC2 | 1<<PC4;// | 0<<PC0; //PC2 and PC3 as direction and PC0 as
force sensor 0b00001100;
	//DDRD = 1<<PD3;

	// stuff to set up interupt
	switch(FREQ_CLK) {
		case 16000000:
		register_B_setting = 0b00000011; // this will start the timer in Normal
mode with prescaler of 64 (CS02 = 0, CS01 = CS00 = 1).
		timer_preload = 255-250; // For prescaler of 64, a count of 250 will
require 1 msec
		break;
		case 8000000:
		register_B_setting =  0b00000011; // this will start the timer in
Normal mode with prescaler of 64 (CS02 = 0, CS01 = CS00 = 1).
		timer_preload = 255 - 125; // for prescaler of 64, a count of 125 will
require 1 msec
		break;
		case 1000000:
		register_B_setting = 0b00000010; // this will start the timer in Normal
mode with prescaler of 8 (CS02 = 0, CS01 = 1, CS00 = 0).
		timer_preload = 255- 125; // for prescaler of 8, a count of 125 will
require 1 msec
		break;
	}
```

```
      TCCR2A = 0x00; // clears WGM00 and WGM01 (bits 0 and 1) to ensure
Timer/Counter is in normal mode.
      TIMSK2 = 1<<TOIE2;      // Enable TIMER2 overflow interrupt
      TCNT2 = timer_preload;  // preload load TIMER0
      sei();      //Enable Global Interrupt
      TCCR2B = register_B_setting;

      // stuff to set up LCD
      LCD_init();  // initialize the LCD
      wait(50);

// start of main while

      LCD_gotoxy(1, 1);         // Go to the location 1,1 of lcd // or can use
LCD_command(0x80); to move to location 1,2
      LCD_print("1=0.25cup: 2=   "); // Print the text
      LCD_gotoxy(1, 2);         // Go to the location 1,2 of lcd // or can use
LCD_command(0xA9); to move to location 1,2
      LCD_print("0.5cup: 3=1cup  ");  // Print the text
      wait(500);


      while(1)
      {// start of while 1

            // GET USER INPUTS - no longer asking user for time, only weight-
special while loop that can only run once
            while (gouser == 0)
            {// start inuput while


                  if (!(PINB & 0b00000001))
                  {
                        LCD_gotoxy(1, 1);         // Go to the location 1,1 of
lcd // or can use LCD_command(0x80); to move to location 1,2
                        LCD_print("               1"); // Print the text
                        LCD_gotoxy(1, 2);         // Go to the location 1,2 of lcd
      // or can use LCD_command(0xA9); to move to location 1,2
                        LCD_print("                ");  // Print the text
                        wait(1000);
                        foodamount = 90;
                        gouser = 1;

                  }
                  else if (!(PINB & 0b00000010))
                  {
                        LCD_gotoxy(1, 1);         // Go to the location 1,1 of
lcd // or can use LCD_command(0x80); to move to location 1,2
                        LCD_print("               2"); // Print the text
                        LCD_gotoxy(1, 2);         // Go to the location 1,2 of lcd
      // or can use LCD_command(0xA9); to move to location 1,2
                        LCD_print("                ");  // Print the text
                        wait(1000);
                        foodamount = 130;
                        gouser = 1;
                  }
                  else if (!(PINB & 0b00000100))
                  {
                        LCD_gotoxy(1, 1);         // Go to the location 1,1 of
```

```
lcd // or can use LCD_command(0x80); to move to location 1,2
                        LCD_print("              3"); // Print the text
                        LCD_gotoxy(1, 2);         // Go to the location 1,2 of lcd
      // or can use LCD_command(0xA9); to move to location 1,2
                        LCD_print("               ");  // Print the text
                        wait(1000);
                        foodamount = 180;
                        gouser = 1;
                  }


            }// end input while

            // Statement to start motor stuff

            if (flag == 1)
            {


            dispensefood(foodamount);

            //check food reservoir

            //AD conversion set up
            ADMUX=0b01100011;//reference AVcc, left justified, ADC3
            PRR=0x00;//clear power reduction bit
            ADCSRA = 0b10000111; // Set ADC Enable bit (7) in ADCSRA register, and
set ADC prescaler to 128 (bits 2-0 of ADCSRA = ADPS2-ADPS0 = 111)
            //read analog input from breakbeam
            ADCSRA = ADCSRA | 0b01000000; //Alternate code: ADCSRA |= (1<<ADSC); //
Start conversion
            while ((ADCSRA & 0b00010000) == 0); //  wait for conversion to finish
            beamvalue = ADCH; // Keep high byte of 10-bit result (throw away lowest
two bits) (0-255 range)
            if(beamvalue>=60)//Beam is not break, need to add food to container
            {
                  LCD_gotoxy(1, 1);         // Go to the location 1,1 of lcd // or
can use LCD_command(0x80); to move to location 1,2
                  LCD_print("Food Supply Low "); // Print the text
                  LCD_gotoxy(1, 2);         // Go to the location 1,2 of lcd // or
can use LCD_command(0xA9); to move to location 1,2
                  LCD_print("Refill Container");  // Print the text
                  wait (10000);
                  LCD_gotoxy(1, 1);         // Go to the location 1,1 of lcd // or
can use LCD_command(0x80); to move to location 1,2
                  LCD_print("               "); // Print the text
                  LCD_gotoxy(1, 2);         // Go to the location 1,2 of lcd // or
can use LCD_command(0xA9); to move to location 1,2
                  LCD_print("               ");  // Print the text
                  flag = 0;
            }
            else if (beamvalue<=60)//beam is broken
            {//do nothing
                  LCD_gotoxy(1, 1);         // Go to the location 1,1 of lcd // or
can use LCD_command(0x80); to move to location 1,2
                  LCD_print("               "); // Print the text
                  LCD_gotoxy(1, 2);         // Go to the location 1,2 of lcd // or
can use LCD_command(0xA9); to move to location 1,2
                  LCD_print("               ");  // Print the text
```

```
            flag = 0;
          }

          } // close flag if


          // water valve code


           PORTC = PORTC | 0b00000100; //forward direction 0b00001000;
           PRR = 0x00;
           ADCSRA = 0b10000111;
           ADMUX = 0b00100000;
           int div = 100;


          if (PINC & 0b00100000)
           {

                ADCSRA = ADCSRA | 0b01000000; //Alternate code: ADCSRA |=
(1<<ADSC); // Start conversion
                while ((ADCSRA & 0b00010000) == 0); // Alternate code: while
((ADCSRA & (1<<ADIF)) ==0); // wait for conversion to finish
                sensorvalue = ADCH; //+ ADCL; // Keep high byte of 10-bit result
(throw away lowest two bits)
                      if (sensorvalue < div)
                      {
                            //    PORTB = PORTB | 0b00000100;
                            PORTD = PORTD | 0b00001000;
                      }
                      else
                      {
                      //    PORTB &= ~(1 << PB2);
                      PORTD &= ~(1 << PD3);

                }

          } else
                { //alert using LCD screen
                //PORTB &= ~(1 << PB2);
                PORTD &= ~(1 << PD3);
                LCD_gotoxy(1, 1);         // Go to the location 1,1 of lcd // or
can use LCD_command(0x80); to move to location 1,2
                LCD_print("Water Supply Low"); // Print the text
                LCD_gotoxy(1, 2);         // Go to the location 1,2 of lcd // or
can use LCD_command(0xA9); to move to location 1,2
                LCD_print("Refill Reservoir");  // Print the text
                wait (5000);
                LCD_gotoxy(1, 1);         // Go to the location 1,1 of lcd // or
can use LCD_command(0x80); to move to location 1,2
                LCD_print("                "); // Print the text
                LCD_gotoxy(1, 2);         // Go to the location 1,2 of lcd // or
can use LCD_command(0xA9); to move to location 1,2
                LCD_print("                ");  // Print the text

          }
```

```
      } // end of while 1
} // end of main


////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

// LCD Functions

void LCD_init(void) { // initialize the LCD
      LCD_command(0x02); // set the LCD to 4-bit control mode
      wait(1);
      LCD_command(0x28); // set the LCD to 2 lines, 5X7 dots, 4-bit mode
      wait(1);
      LCD_command(0x01); // clear LCD
      wait(1);
      LCD_command(0x0E); // turn LCD cursor ON
      wait(1);
      LCD_command(0x80); //  set cursor at first line, first position (positions
are defined by lower 7 bits and are 1-40 for line 1 and 41-80 for line 2, though
      // positions 17-40 and 57-80 are not visible on a 16-column display
      wait(1);
      return;
} // end LCD_init()

void LCD_gotoxy(unsigned char x, unsigned char y) {  // move the LCD cursor to a
given position on the LCD screen
      unsigned char firstCharAdr[] = {0x80, 0xC0, 0x94, 0xD4};
      LCD_command(firstCharAdr[y - 1] + x - 1);
      wait(1);
} //end LCD_gotoxy

void LCD_command(char command_value) { // send a command to the LCD driver -- 1/2
nibble at a time
      lcd_port = command_value & 0xF0;          // Send upper nibble (mask lower
nibble because PD4-PD7 pins are used for data)
      lcd_port &= ~(1 << lcd_RS); // lcd_RS = 0 for command
      lcd_port |= (1 << lcd_EN);  // toggle lcd_EN bit
      wait(1);
      lcd_port &= ~(1 << lcd_EN); // toggle bit back off
      wait(10);
      lcd_port = ((command_value << 4) & 0xF0); // Send lower nibble (shift 4-bits
and mask for lower nibble)
      lcd_port &= ~(1 << lcd_RS); // lcd_RS = 0 for command
      lcd_port |= (1 << lcd_EN);  // toggle lcd_EN bit
      wait(1);
      lcd_port &= ~(1 << lcd_EN); // toggle bit back off
      wait(10);
} // end LCD_command()

void LCD_data(char data_value) { // send data to the LCD driver -- 1/2 nibble at a
time
      lcd_port = data_value & 0xF0; // Send upper nibble (mask lower nibble because
PD4-PD7 pins are used for data)
      lcd_port |= (1 << lcd_RS);  // lcd_RS = 1 for data
      lcd_port |= (1 << lcd_EN);  // toggle lcd_EN bit
```

```
      wait(1);
      lcd_port &= ~(1 << lcd_EN); // toggle bit back off
      wait(10);
      lcd_port = ((data_value << 4) & 0xF0); // Shift 4-bit and mask
      lcd_port |= (1 << lcd_RS);  // lcd_RS = 1 for data
      lcd_port |= (1 << lcd_EN);  // toggle lcd_EN bit
      wait(1);
      lcd_port &= ~(1 << lcd_EN); // toggle bit back off
      wait(10);
} // end LCD_data()

void LCD_print( char *str) { // send string information to the LCD driver to be
printed (the argument stores the address of the string in pointer *str)
      int i = 0;
      while (str[i] != '\0') {     // loop until NULL character in the string
            LCD_data(str[i]); // sending data on LCD byte by byte
            i++;
      }
      return;
} // end LCD_print()

void wait(volatile unsigned int number_of_msec) {
      // This subroutine creates a delay equal to number_of_msec*T, where T is 1
msec
      // It changes depending on the frequency defined by FREQ_CLK
      char register_B_setting;
      char count_limit;

      // Some typical clock frequencies:
      switch(FREQ_CLK) {
            case 16000000:
            register_B_setting = 0b00000011; // this will start the timer in Normal
mode with prescaler of 64 (CS02 = 0, CS01 = CS00 = 1).
            count_limit = 250; // For prescaler of 64, a count of 250 will require
1 msec
            break;
            case 8000000:
            register_B_setting =  0b00000011; // this will start the timer in
Normal mode with prescaler of 64 (CS02 = 0, CS01 = CS00 = 1).
            count_limit = 125; // for prescaler of 64, a count of 125 will require
1 msec
            break;
            case 1000000:
            register_B_setting = 0b00000010; // this will start the timer in Normal
mode with prescaler of 8 (CS02 = 0, CS01 = 1, CS00 = 0).
            count_limit = 125; // for prescaler of 8, a count of 125 will require 1
msec
            break;
      }

      while (number_of_msec > 0) {
            TCCR0A = 0x00; // clears WGM00 and WGM01 (bits 0 and 1) to ensure
Timer/Counter is in normal mode.
            TCNT0 = 0;  // preload value for testing on count = 250
            TCCR0B =  register_B_setting;  // Start TIMER0 with the settings
defined above
            while (TCNT0 < count_limit); // exits when count = the required limit
for a 1 msec delay
            TCCR0B = 0x00; // Stop TIMER0
```

```
            number_of_msec--;
      }
}

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

// Motor Function
void dispensefood (int foodamount)
{

      //AD conversion set up for sensor
      ADMUX=0b01100001;//reference AVcc, left justified, ADC1
      PRR=0x00;//clear power reduction bit
      ADCSRA = 0b10000111; // Set ADC Enable bit (7) in ADCSRA register, and set
ADC prescaler to 128
      // open food door
      for(int x=1;x<=100;x++) //number of steps to open, about 45 degrees
      {
            step_CW();//one step CW
            wait(10);
      }

      //wait until weight sensor reaches predetermined weight (variable=food
amount)
      char weightvalue=0;
      while(1)
      {
            // Read analog input
            ADCSRA = ADCSRA | 0b01000000; // Start conversion
            while ((ADCSRA & 0b00010000) == 0); // wait for conversion to finish
            weightvalue = ADCH; // Keep high byte of 10-bit result (throw away
lowest two bits)
            if(weightvalue>=foodamount)
            {
                  break;
            }else
            {
                  //nothing keep checking
            }
      }

      //close food door
      for(int x=1;x<=100;x++)//number of steps to close, about 45 degrees
      {
            step_CCW();//one step CW
            wait(10);
      }
}//end dispensefood function


void step_CW() {
      PORTB=1<<PB6;//set DIR pin high for cw
      PORTB=1<<PB3;
      PORTB=0<<PB3;
}// end step_CW


void step_CCW()
```

```
{
      PORTB=0<<PD6;
      PORTB=1<<PB3;
      PORTB=0<<PB3;
}// end step_CCW

////////////////////////////////////////////////////////////////////////////////
///////
```