```
/*****************************************************
* MainProgramEJB.c                                   *
* EEL 5666C                                          *
* IMDL Spring 2008                                   *
* This program is the main program for which the     *
* robot, GIR, will perform all of its behaviors.     *
*****************************************************/

#include <avr/io.h>

#define DIR_L PORTE2
#define DIR_R PORTE5
#define MOTOR_L OCR3A         //pin3 port E
#define MOTOR_R OCR3B         //pin4 port E

void init_timer();
void motor_move();
void dc_move();

// initialize PWM timer

#include <avr/io.h>

#define S1R (1<<0) // Echo output 1
#define S1 (1<<1) // Trigger input 1
#define S2R (1<<6) // Echo output 2
#define S2 (1<<7) // Trigger input 2
#define TRAILS 10
// PORTD , PIND == PORTA, PINA
#define MIN(x,y) ((x) < (y)) ? (x) :(y)
#define MAX(x,y) ((x) > (y)) ? (x) :(y)

void lcd_delay();             // short delay (50000 clocks)
void lcd_init();              // sets lcd in 4 bit mode, 2-line mode,
with cursor on and set to blink
void lcd_cmd();                     // use to send commands to lcd
void lcd_disp();              // use to display text on lcd
void lcd_clear();             // use to clear LCD and return cursor to
home position
void lcd_row(int row);        // use to put the LCD at the desired row
void config_adc(void);
void init_adc( void );

int getSonar1();              // use to read value from sonar module
int getSonar2();

/* IMPORTANT!

Before using this code make sure your LCD is wired up correctly. The LCD
is connected to PortC of the At-Mega128 in the following manner:

PortC bit 7 :     LCD data bit 7 (MSB)
PortC bit 6 :     LCD data bit 6
PortC bit 5 :     LCD data bit 5
PortC bit 4 :     LCD data bit 4 (LSB)

PortC bit 3 :     (not connected)
PortC bit 2 :     LCD enable pin (clock)
PortC bit 1 :     LCD R/W (read / write) signal
PortC bit 0 :     LCD RS (register select) pin
```

```
Also remember you must connect a potentiometer (variable resistor) to
the vcc, gnd, and contrast pins on the LCD.
The output of the pot (middle pin) should be connected to the contrast
pin.  The other two can be on either pin.*/

void init_timer()
{
      //initializes motors, enable OC3A, B, and C

      DDRE = 0xFF ;     //PORT E out
      //DDR_OC3A = 0b1;
      //DDR_OC3B = 0b1;


      TCCR3A = 0xA8;    //10101000 clear OCA,B,C on campare match- non-
inv PWM
      TCCR3B = 0x12;    //00010010 mode  8 - PWM phase/freq correct, clk
(io)/8
      ICR3 = 18432;     //PWM = 50hz.  1/(14.7456 MHz/8)*2*18432 = 1/50
      TCNT3 = 0x00;     //

      MOTOR_L = 0;      //not moving
      MOTOR_R = 0;

      PORTE = 0x0;
}


void motor1_move(uint16_t speed)
{
      int increment_R = 0;

      if(MOTOR_R > speed)
      {
            increment_R = -1;
      }
      else if (MOTOR_R < speed)
      {
            increment_R = 1;
      }
      else
      {
            increment_R = 0;
      }

      while (MOTOR_R != speed) // if either motor does not equal speed,
run this loop
      {
            if (MOTOR_R != speed)
            {
                  MOTOR_R =MOTOR_R + increment_R;
            }

            //delayus(50);

            //long x = 0;
            //for (x = 0; x < 294000; x += 1)
            //{
                  //delay for .02 sec
```

```
            //}

            //lcd_cmd(0x01);        // clrscn rtn home
            //lcd_int(MOTOR_L);
      }
}

void motor2_move(uint16_t speed)
{
      int increment_L = 0;

      if(MOTOR_L > speed)
      {
            increment_L = -1;
      }
      else if (MOTOR_L < speed)
      {
            increment_L = 1;
      }
      else
      {
            increment_L = 0;
      }


      while (MOTOR_L != speed) // if either motor does not equal speed,
run this loop
      {
            if (MOTOR_L != speed)
            {
                  MOTOR_L = MOTOR_L + increment_L;
            }

            //delayus(50);

            //long x = 0;
            //for (x = 0; x < 294000; x += 1)
            //{
                  //delay for .02 sec
            //}

            //lcd_cmd(0x01);        // clrscn rtn home
            //lcd_int(MOTOR_L);*/
      }
}


void lcd_delay()        // delay for 10000 clock cycles
{
      long int ms_count = 0;
      while (ms_count < 1000)
      {
            ms_count = ms_count + 1;
      }
}

void lcd_cmd( unsigned int myData )
{

      /* READ THIS!!!
```

```
      The & and | functions are the BITWISE AND and BITWISE OR functions
respectively.  DO NOT
      confuse these with the && and || functions (which are the LOGICAL
AND and LOGICAL OR functions).

      The logical functions will only return a single 1 or 0 value, thus
they do not work in this scenario
      since we need the 8-bit value passed to this function to be
preserved as 8-bits
      */

      unsigned int temp_data = 0;

      temp_data = ( myData | 0b00000100 );      // these two lines leave
the upper nibble as-is, and set
      temp_data = ( temp_data & 0b11110100 );   // the appropriate
control bits in the lower nibble
      PORTC = temp_data;
      lcd_delay();
      PORTC = (temp_data & 0b11110000);         // we have written upper
nibble to the LCD

      temp_data = ( myData << 4 );              // here, we reload
myData into our temp. variable and shift the bits
                                                                  // to
the left 4 times.  This puts the lower nibble into the upper 4 bits

      temp_data = (temp_data & 0b11110100);     // temp_data now
contains the original
      temp_data = (temp_data | 0b00000100);     // lower nibble plus
high clock signal

      PORTC = temp_data;                                  // write the
data to PortC
      lcd_delay();
      PORTC = (temp_data & 0b11110000);         // re-write the data to
PortC with the clock signal low (thus creating the falling edge)
      lcd_delay();

}

void lcd_disp(unsigned int disp)
{

      /*

      This function is identical to the lcd_cmd function with only one
exception.  This least significant bit of
      PortC is forced high so the LCD interprets the values written to
is as data instead of a command.

      */

      unsigned int temp_data = 0;

      temp_data = ( disp & 0b11110000 );
      temp_data = ( temp_data | 0b00000101 );
      PORTC = temp_data;
      lcd_delay();
```

```
	PORTC = (temp_data & 0b11110001);
	lcd_delay();                                    // upper nibble

	temp_data = (disp << 4 );
	temp_data = ( temp_data & 0b11110000 );
	temp_data = ( temp_data | 0b00000101 );
	PORTC = temp_data;
	lcd_delay();
	PORTC = (temp_data & 0b11110001);
	lcd_delay();                                    // lower nibble

}


void lcd_init()
{
	lcd_cmd(0x33);          // writing 0x33 followed by
	lcd_cmd(0x32);          // 0x32 puts the LCD in 4-bit mode

	lcd_cmd(0x28);          // writing 0x28 puts the LCD in 2-line
mode

	lcd_cmd(0x0F);          // writing 0x0F turns the display on,
curson on, and puts the cursor in blink mode

	lcd_cmd(0x01);          // writing 0x01 clears the LCD and sets
the cursor to the home (top left) position

	//LCD is on... ready to write

}

void lcd_string(char *a)
{

	/*
	This function writes a string to the LCD.  LCDs can only print one
character at a time so we need to
	print each letter or number in the string one at a time.  This is
accomplished by creating a pointer to
	the beginning of the string (which logically points to the first
character).  It is important to understand
	that all strings in C end with the "null" character which is
interpreted by the language as a 0.  So to print
	an entire string to the LCD we point to the beginning of the
string, print the first letter, then we increment
	the pointer (thus making it point to the second letter), print
that letter, and keep incrementing until we reach
	the "null" character".  This can all be easily done by using a
while loop that continuously prints a letter and
	increments the pointer as long as a 0 is not what the pointer
points to.
	*/

	while (*a != 0)
	{
		lcd_disp((unsigned int) *a);  // display the character that
our pointer (a) is pointing to
		a++;                                    // increment a
	}
```

```
      return;

}


void lcd_int(int value)
{

      /*
      This routine will take an integer and display it in the proper
order the
      LCD.*/

      int temp_val;
      int x = 10000;                // since integers only go up to
32768, we only need to worry about
                                          // numbers containing at most
a ten-thousands place

      while (value / x == 0)  // the purpose of this loop is to find out
the largest position (in decimal)
      {                                   // that our integer contains.
As soon as we get a non-zero value, we know
            x/=10;                        // how many positions there
are int the int and x will be properly initialized to the largest
      }                                   // power of 10 that will
return a non-zero value when our integer is divided by x.


      while (x >= 1)                      // this loop is where the
printing to the LCD takes place.  First, we divide
      {                                                 // our integer by
x (properly initialized by the last loop) and store it in
            temp_val = value / x;         // a temporary variable so our
original value is preserved.  Next we subtract the
            value -= temp_val * x;        // temp. variable times x from
our original value.  This will "pull" off the most
            lcd_disp(temp_val+ 0x30);     // significant digit from our
original integer but leave all the remaining digits alone.
                                                  // After this, we
add a hex 30 to our temp. variable because ASCII values for integers
            x /= 10;                            // 0 through 9
correspond to hex numbers 30 through 39.  We then send this value to the
      }                                           // LCD (which
understands ASCII).  Finally, we divide x by 10 and repeat the process
                                                  // until we get a
zero value (note: since our value is an integer, any decimal value
      return;                                     // less than 1
will be truncated to a 0)

}

void lcd_clear()        // this function clears the LCD and sets the
cursor to the home (upper left) position
{
      lcd_cmd(0x01);

      return;
}
```

```
void lcd_row(int row)   // this function moves the cursor to the
beginning of the specified row without changing
{                                  // any of the current text on the
LCD.

      switch(row)
      {

            case 0: lcd_cmd(0x02);
            case 1:     lcd_cmd(0xC0);
      }

      return;
}

void sonar_init()
{
      DDRA = S1|S2;
}

int getSonar1()
{

      unsigned long total = 0;
      unsigned int n,i;
      unsigned int min,max=0;

      min = 0xffff;

      for (i = 0;i < TRAILS; i++)
      {
            n = 0;
            PORTA =  (PINA & (~S1)); //(PIND & 0x7F); // these two lines
create a rising edge
            PORTA =  (PINA | S1); //(PIND | 0x80);    // on PORTD pin 7
(2)

      //lcd_clear();
      //lcd_string("rising edge done");

            while (n < 40)
            {
                  //waste enough clock cycles for at least 10us to pass
                  n += 1;
                  n++;
            //    lcd_clear();
            //    lcd_int(n);
            }

            PORTA =  (PINA & (~S1));//    PORTD = (PIND & 0x7F);  //
force PORTD pin 7(2) low to create a falling edge
                                          // this sends out the trigger

            while (!(PINA & S1R))
            {
                  // do nothing as long as echo line is low
            }

            n = 0;      //re-use our dummy variable for counting
```

```
            while (PINA & S1R)
            {
                  n += 1;           // add 1 to n as long as PORTD pin 0
is high
            }

            //when we get here, the falling edge has occured
            total +=n;
            min = MIN (n,min);
            max = MAX (n,max);
      }
      total-= (min + max);
      n = (int) (total/TRAILS - 2);

      return n;

}

int getSonar2()
{

      unsigned long total = 0;
      unsigned int n,i;
      unsigned int min,max=0;

      min = 0xffff;

      for (i = 0;i < TRAILS; i++)
      {
            n = 0;
            PORTA =  (PINA & (~S2)); //(PIND & 0x7F); // these two lines
create a rising edge
            PORTA =  (PINA | S2); //(PIND | 0x80);    // on PORTD pin 7
(2)

      //lcd_clear();
      //lcd_string("rising edge done");

            while (n < 40)
            {
                  //waste enough clock cycles for at least 10us to pass
                  n += 1;
                  n++;
            //    lcd_clear();
            //    lcd_int(n);
            }

            PORTA =  (PINA & (~S2));//    PORTD = (PIND & 0x7F);  //
force PORTD pin 7(2) low to create a falling edge
                                          // this sends out the trigger

            while (!(PINA & S2R))
            {
                  // do nothing as long as echo line is low
            }

            n = 0;      //re-use our dummy variable for counting

            while (PINA & S2R)
            {
```

```
                  n += 1;           // add 1 to n as long as PORTD pin 0
is high
            }

            //when we get here, the falling edge has occured
            total +=n;
            min = MIN (n,min);
            max = MAX (n,max);
      }
      total-= (min + max);
      n = (int) (total/TRAILS - 2);

      return n;

}


void init_adc( void )
{
  ADCSRA |= 0b01000000;   // start free running conversions
}

void config_adc(void)
{
   DDRF = 0b00000000; // set port F to all input
                  // Note: when JTAGEN fuse is set, F4 - F7 don't work
   PORTF = 0x00;      // make sure pull up resistor is not enabled

   ADMUX = 0b01100000; // 5V reference, select channel0 (pin F0)
   ADCSRA |= 0b10100111; // turn on ADC, don't start conversions
                    // free funning
                    // divide clock by 128

}


int main(void)
{

      long i; long SonarVal1 = 0, SonarVal2 = 0;
      char s1[20]="GIR reportn' 4 duty!";

      sonar_init(); //0x80;   // sonar is connected with trigger on
PORTD.7 and echo on PORTD.0

      DDRC = 0xFF;      // set all pins on portC to output  (could also
use DDRC = 0b11111111)

      lcd_init();       // set lcd in 4 bit mode, 2-line mode, with
cursor on and set to blink

      DDRD = 0b00000000;


      unsigned int Dvalue;

      lcd_string("Please place item...");       // if your LCD is wired
up correctly, you will see this text
                                                                  // on
it when you power up your Micro-controller board.
```

```
      for (i = 0; i < 1000; i++)
        {
          lcd_delay();        //delay to read LCD (humans reading)
        }

      lcd_row(1);

      s1[20]=0;

      lcd_string(s1);

      init_timer();

      long counter=0;

   config_adc();

   init_adc(); // initialize ADC converter

   DDRC = 0xFF;               // set portC to output  (could also use
DDRC = 0b11111111)
   DDRE = 0xFF;         // set portE to output
 //DDRE = 0b11111111;   // set portE to output
   DDRB = 0xFF;               // set portB to output


int temp;
   int analogLow = 0;
   int analogHigh = 0;

      counter++;

        analogLow = ADCL; // read ACD low register
        analogHigh = ADCH; // read ACD high register

        lcd_clear();
        lcd_string("Low = ");
        lcd_int(analogLow);
        lcd_row(1);
        lcd_string("High = ");
        lcd_int(analogHigh);

        for (i = 0; i < 100; i++)
        {
          lcd_delay();        //delay to read LCD (humans reading)
        }

      PORTE = 0xAA;     // set alternating pins on portE to 5v
      temp = PINB;            // read portB, store value to temp
      PORTB = !(temp);  // complement portB and write back


            int path = 0;

            if (analogHigh > 150)
{           path = 1;
}

            else if (analogHigh > 90)
```

```
{           path = 2;
}

            else if (analogHigh = 0)
{           path = 3;
}

            if (path == 1)
{
            lcd_string("PATH 1");
            motor1_move(1600);
            motor2_move(1400);


            for (i = 0; i<800; i++){      // for a certain time move in
a straight line
                                                        // to catch the
correct line to follow
            lcd_delay();

            }

            motor1_move(1505);
            motor2_move(1500);
}


            if (path == 2)
{
            lcd_string("PATH 2");
            motor1_move(1505);
            motor2_move(1400);

            for (i = 0; i<800; i++)
            {
            lcd_delay();      // turn left for a certain time
            }                       // to catch the correct line to
follow

            motor1_move(1505);
            motor2_move(1500);
}


            if (path ==3)
{

            lcd_string("PATH 3");
            motor1_move(1600);
            motor2_move(1500);

            for (i = 0; i<800; i++){
            lcd_delay();      // turn right for a certain time
            }                       // to catch the correct line to
follow

            motor1_move(1505);
            motor2_move(1500);

}
```

```
	while(1)
	{

		SonarVal1 = getSonar1();
		SonarVal2 = getSonar2();
		lcd_clear();
		lcd_string("S:");
		lcd_int(SonarVal1);
		lcd_string(" ");
		lcd_int(SonarVal2);
		Dvalue = PIND & 0b00000111;

		for (i=0; i<10; i++)
		{
			lcd_delay();
		}


		//if (pin_b==0x06)
		if (Dvalue == 0b00000110)
			{
			motor1_move(1600); //1300 was right, 1500 was center,
1700 was left
			motor2_move(1500);
			}
		if (Dvalue == 0b00000011)
			{
			motor2_move(1400);
			motor1_move(1505);
			}
		if (Dvalue == 0b00000101)
			{
			motor1_move(1600);
			motor2_move(1400);
			}
		if (Dvalue == 0b00000000)
			{
			motor1_move(1505);
			motor2_move(1500);
			}

		if (Dvalue == 0b00000111)
			{


		for (i=0; i<800; i++)
		{
		lcd_delay();
		}

			motor1_move(1400);
			motor2_move(1400);
		}

		if (SonarVal1 <850 || SonarVal2 < 850)
		{
		motor2_move(1500);
```

```
            motor1_move(1505);
            }

      }

      }

      return 0;
}
```