```
*************userspace.c****************

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <termios.h>
#include <unistd.h>

//int serial_putchar(char, FILE *);
//int serial_getchar(FILE *):
//static FILE serial_stream = FDEV_SETUP_STREAM (serial_putchar, serial_getchar, _FDEV_SETU
P_RW);
int init_serial_user();

int main (int argc, char *argv[])
{
        if (argc != 2) {
                return -1;
        }
        //Sample Count
        int samples = atoi(argv[1]);

        printf("Sample count = %d\n", samples);

        //Integer counter for seconds
        int seconds = 0;
        //Integer counter for minutes

        double total = 0;
        double decimal = 0;
        double average = 0;
        int results = 0;

        //Data buffer for later
        char data[100];
        int fdserial = 0;

        //Get our serial file to setup comm between AVR and RPI
        fdserial = init_serial_user();

        //Create file stream for input and output, as well as where to store voltages
        FILE *usart_in;
        FILE *usart_out;
        FILE *data_out;
        int from_string = 0;

        usart_in = fdopen(fdserial, "r");
        usart_out = fdopen(fdserial, "w");
        data_out = fopen("rail_temps.dat", "a");
        if (usart_in == NULL || usart_out == NULL || data_out == NULL) {
                printf("Error opening file(s).\n");
                return 0;
        }
        for (int iter = 0; iter < samples; iter++) {
        for (int i = 0; i < 32; i++) {

        fgets(data, 100, usart_in);

        //Do appropriate equation for voltage
        results = atoi(data);
        printf("%d, raw \n", results);
        //results = results - 273;
        decimal = (double)(results * 0.83) - 273;

        total += decimal;
                }
        average = total / 32;
        average = average;

        fprintf(data_out, "%lf\n", average);
        total = 0;
        average = 0;
        decimal = 0;

        }


        fclose(data_out);
        fclose(usart_in);
        fclose(usart_out);
        return 0;
}

int init_serial_user()
{
        int fd;
        struct termios tc;

        //Step 1, open hardware serial (NOT USB) in dev
        fd = open ("/dev/serial0", O_RDWR | O_NOCTTY);

        if (fd <= 0) {
                printf("Serial port inexistent or failed to open.\n");
                return -1;
        }
        //Setup Termios
        tcgetattr(fd, &tc);

        //Set the flags
        tc.c_iflag = IGNPAR;
        tc.c_oflag = 0;
        tc.c_cflag = CS8 | CREAD | CLOCAL;
        tc.c_lflag = ICANON; // Always set, unless whatever is being done REQUIRES this not
 to be

        //Set the baud
        cfsetispeed(&tc, B9600); //Input Baud rate
        cfsetospeed(&tc, B9600); //Output Baud rate

        tcsetattr(fd, TCSANOW, &tc);
        return fd;
}



****************avrcode.c*******************

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include <stdio.h>
#include <math.h>
#include <avr/sleep.h>
#include <string.h>

#define F_CPU 8000000UL
#include <util/delay.h>

//STDIO Functions
int serial_putchar(char, FILE *);
int serial_getchar(FILE *);
static FILE serial_stream = FDEV_SETUP_STREAM (serial_putchar, serial_getchar, _FDEV_SETUP_
RW);
```

```
void init_pwm(void);
void update_clock_speed(void);
void init_usart(void);
void init_ADC(void);
int read_ADC(void);

int serial_putchar(char value, FILE *);
int serial_getchar(FILE *);
//This code is distributed with no warranty expressed or implied.
//It does not contain any known bugs, but has not been tested.
//What it is intended to do is use the first two bytes of eeprom
//as an offset and direction for adjusting the internal oscillator
//The first byte is an unsigned byte that is the amount to adjust
//the OSCCAL register.  The next byte will be 0 or 1 depending on
//whether the adjustment should be positive (0) or negative (1).
//The value 0xff is intentionally avoided to distinguish unprogrammed
//eeprom locations.

//main first calls update_clock_speed to make the adjustments to
//the oscillator calibration and then calls init_pwm to set up
//a 100Hz 50% duty cycle square wave on OC1A (pin 15 on the 28 pin
//DIP package).


int main()
{
  update_clock_speed();  //adjust OSCCAL
  init_pwm();     //set up hardware PWM
  init_usart();
  init_ADC();
  _delay_ms(2000);
while(1) {
        printf("%d\n", read_ADC());
        }
}
//read the first two bytes of eeprom, if they have been programmed
//use the first byte as an offset for adjusting OSCCAL and the second as
//the direction to adjust 0=increase, 1=decrease.
//Any offset is allowed and users are cautioned that it is possible to
// adjust the oscillator beyond safe operating bounds.
void update_clock_speed(void)
{
  char temp;
  temp=eeprom_read_byte((void *)1); //read oscillator offset sign
                                    //0 is positive 1 is  negative
                                    //erased reads as ff (so avoid that)
  if(temp==0||temp==1)      //if sign is invalid, don’t change oscillator
  {
      if(temp==0)
          {
             temp=eeprom_read_byte((void *)0);
                 if(temp != 0xff) OSCCAL+=temp;
          }
          else
          {
             temp=eeprom_read_byte((void *)0);
                 if(temp!=0xff) OSCCAL -=temp;
          }
  }
}


void init_pwm(void)
{
  // **************************************************************
  // ***   Timer 1                                                *
  // **************************************************************
  DDRB |= (1<<PB1);  //set OC1A as an output
  OCR1A=19999;    //set initial compare at 50%
  ICR1=39999U; // 8 MHz /40000/2 = PWM frequency = 100 Hz
  TCCR1A = (1<<COM1A1); //zeros in COM1B1,COM1B0,WGM11,WGM10
  //internal clock, no prescaler , PWM mode 8
  TCCR1B = (1<<WGM13) | (1<<CS10);
}

void init_usart(void)
{
        //Set baud rate
        UBRR0H = 0;
        UBRR0L = 51;
        UCSR0A = 0;
        //Enable reciever and transmitter
        UCSR0B = (1<<RXEN0) | (3<<TXEN0);

        //Set 8 data, 2 stop
        UCSR0C = (1 << USBS0) | (3 << UCSZ00);
        stdin = &serial_stream;
        stdout = &serial_stream;
}
void init_ADC()
{
        ADMUX = (3 << REFS0) | 8; // sets the mux for adc to recieve avcc
        ADCSRA = (1 << ADEN) | (6 << ADPS0); //Toggle ADC Enable bit to initialize the ADC
        ADCSRB = 0;
        DIDR0 = 0; //Advice from Bruce to enable digital input
}

int serial_putchar (char val, FILE *fp)
{
        while((UCSR0A & (1 << UDRE0)) == 0); //Wait for AVR register to empty, and signal i
t’s ready
        UDR0 = val; //Sets the bit to the specified value
        return 0;
}

int serial_getchar (FILE *fp)
{
        while ((UCSR0A & (1 << RXC0)) == 0); //Wait for RPI to be ready for transfer
        return UDR0; //Transmits the data held in serial register

}
int read_ADC(void)
{
                ADCSRA |= (1 << ADSC);
                while ((ADCSRA & (1 << ADSC))); //Wait for ADC to finish conversion
                return ADC;
}
```