Serak Gebremedhin
Github root directory: https://github.com/Ber-geb/solid-octo-tribble.git

Date Submitted:

Task 00: Execute provided code

Modified Code:

```
#include <stdint.h>
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include"inc/hw_ints.h"
#include"driverlib/interrupt.h"
#include "driverlib/adc.h"
//#define TARGET_IS_BLIZZARD_RB1
#include "driverlib/rom.h"
#include "driverlib/timer.h"

volatile uint32_t ui32TempAvg; //holds temperature average value
volatile uint32_t ui32TempValueC; //holds temperature value in celsius
volatile uint32_t ui32TempValueF; //holds temperature value in fahrenheit
uint32_t ui32Period;
uint32_t ui32ADC0Value[4]; //array holds data read
char str[10];

// Implementation of itoa()
char* itoa( uint32_t num, char* str, int base)
{
    int i = 0;

    /* Handle 0 explicitely, otherwise empty string is printed for 0 */
    if (num == 0)
    {
        str[i++] = '0';
        str[i] = '\0';
        return str;
    }

    // Process individual digits
    while (num != 0)
    {
        int rem = num % base;
        str[i++] = (rem > 9)? (rem-10) + 'a' : rem + '0';
        num = num/base;
    }

    str[i] = '\0'; // Append string terminator

    return str;
}
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
void UARTIntHandler(void)
{
    uint32_t ui32Status;

    ui32Status = UARTIntStatus(UART0_BASE, true); //get interrupt status

    UARTIntClear(UART0_BASE, ui32Status); //clear the asserted interrupts

    while(UARTCharsAvail(UART0_BASE)) //loop while there are chars
    {
        UARTCharPutNonBlocking(UART0_BASE, UARTCharGetNonBlocking(UART0_BASE));//echo
character
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2); //blink LED
        SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //delay ~1 msec
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0); //turn off LED
    }
}

int main(void) {


    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ); //set up clock

    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0); //enable UART0 peripheral
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); //enable GPIOA peripheral

    GPIOPinConfigure(GPIO_PA0_U0RX); //configure pin as receiver
    GPIOPinConfigure(GPIO_PA1_U0TX); //configure pin as transmitter
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1); //set pintype to UART

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //initialize GPIO peripheral
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
//enable GPIO pin for output for LED

    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
UART_CONFIG_PAR_NONE));
    //Initializes parameters for UART: 114200, 8-1-N

//    IntEnable(INT_UART0); //enable uart0 interrupt
 //   UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //enable receiver
interrupts

    /*
    UARTCharPut(UART0_BASE, 'E'); //calls to create a prompt
    UARTCharPut(UART0_BASE, 'n');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'T');
    UARTCharPut(UART0_BASE, 'e');
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
    UARTCharPut(UART0_BASE, 'x');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, ':');
    UARTCharPut(UART0_BASE, ' '); //"Enter text: "
     */

    // str[0] = 'p';
    // UARTCharPut(UART0_BASE, str[0]);


    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0); //enables ADC0 peripheral

    ROM_ADCHardwareOversampleConfigure(ADC0_BASE, 64); //API call with 64 samples to
be averaged

    ROM_ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0); //use ADC0,
sample sequencer 1
    //want the processor to trigger sequence and want to use highest priority

    ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS); //first step of adc
sequencer
    ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS); //second step
    ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS); //third step
    ROM_ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
//fourth step will
    //configure interrupt flag, sample the temperature sensor, and tell adc logic
that this
    //is the last step

    ROM_ADCSequenceEnable(ADC0_BASE, 1); //enable ADC sequencer 1

    ROM_ADCIntClear(ADC0_BASE, 1); //clear interrupt status flag
    ROM_ADCProcessorTrigger(ADC0_BASE, 1); //trigger adc conversion

    while(!ROM_ADCIntStatus(ADC0_BASE, 1, false)) //wait for conversion to finish
    {
    }

    ROM_ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value); //copies data from ...
    //the specified saple sequencer output FIFO to a buffer in memory
    ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] + 2)/4;
    //calculate the average temperature value
    ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10; //calculate temp C
    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5; //calculate temp F

    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0); //enable timer
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC); //configure timer 0

    IntEnable(INT_TIMER0A); //enable timer interrupt
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    IntMasterEnable(); //enable all interrupts

    ui32Period = (SysCtlClockGet() / 10) * 5; //0.5 ms period
    TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period -1);
```

```
    TimerEnable(TIMER0_BASE, TIMER_A);

    while (1) //infinite loop
    {
        //echoes what is types in terminal
        //if (UARTCharsAvail(UART0_BASE)) UARTCharPut(UART0_BASE,
UARTCharGet(UART0_BASE));

    }
}

void Timer0IntHandler(void)
{
    if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0); //turn
off led
    }
    else
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4); //turn on led
    }

    // Clear the timer interrupt
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

    ROM_ADCIntClear(ADC0_BASE, 1); //clear interrupt status flag
    ROM_ADCProcessorTrigger(ADC0_BASE, 1); //trigger adc conversion

    while(!ROM_ADCIntStatus(ADC0_BASE, 1, false)) //wait for conversion to finish
    {
    }

    ROM_ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value); //copies data from ...
    //the specified sample sequencer output FIFO to a buffer in memory
    ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] + 2)/4;
    //calculate the average temperature value
    ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10; //calculate temp C
    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5; //calculate temp F

    itoa(ui32TempValueF, str, 10);

    UARTCharPut(UART0_BASE, str[1]);
    UARTCharPut(UART0_BASE, str[0]);
    UARTCharPut(UART0_BASE, '\r'); //carriage return
    UARTCharPut(UART0_BASE, '\n'); //new line
;
}
```

Youtube Link:

https://youtu.be/xu-XoFlhbhE

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

Serak Gebremedhin
Github root directory: https://github.com/Ber-geb/solid-octo-tribble.git

---

Task 01:

Youtube Link:

https://youtu.be/5HC2qxpemiA

Modified Code:

```
#include <stdint.h>
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include"inc/hw_ints.h"
#include"driverlib/interrupt.h"
#include "driverlib/adc.h"
//#define TARGET_IS_BLIZZARD_RB1
#include "driverlib/rom.h"
#include "driverlib/timer.h"

volatile uint32_t ui32TempAvg; //holds temperature average value
volatile uint32_t ui32TempValueC; //holds temperature value in celsius
volatile uint32_t ui32TempValueF; //holds temperature value in fahrenheit
uint32_t ui32Period;
uint32_t ui32ADC0Value[4]; //array holds data read
char str[10];

// Implementation of itoa()
char* itoa( uint32_t num, char* str, int base)
{
    int i = 0;

    /* Handle 0 explicitely, otherwise empty string is printed for 0 */
    if (num == 0)
    {
        str[i++] = '0';
        str[i] = '\0';
        return str;
    }

    // Process individual digits
    while (num != 0)
    {
        int rem = num % base;
```

```
        str[i++] = (rem > 9)? (rem-10) + 'a' : rem + '0';
        num = num/base;
    }

    str[i] = '\0'; // Append string terminator

    return str;
}


void UARTIntHandler(void)
{
    uint32_t ui32Status;

    ui32Status = UARTIntStatus(UART0_BASE, true); //get interrupt status

    UARTIntClear(UART0_BASE, ui32Status); //clear the asserted interrupts

    while(UARTCharsAvail(UART0_BASE)) //loop while there are chars
    {
        UARTCharPutNonBlocking(UART0_BASE, UARTCharGetNonBlocking(UART0_BASE));//echo
character
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2); //blink LED
        SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //delay ~1 msec
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0); //turn off LED
    }
}

int main(void) {


    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ); //set up clock

    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0); //enable UART0 peripheral
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); //enable GPIOA peripheral

    GPIOPinConfigure(GPIO_PA0_U0RX); //configure pin as receiver
    GPIOPinConfigure(GPIO_PA1_U0TX); //configure pin as transmitter
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1); //set pintype to UART

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //initialize GPIO peripheral
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
//enable GPIO pin for output for LED

    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
UART_CONFIG_PAR_NONE));
    //Initializes parameters for UART: 114200, 8-1-N

//    IntEnable(INT_UART0); //enable uart0 interrupt
 //   UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //enable receiver
interrupts

    /*
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
    UARTCharPut(UART0_BASE, 'E'); //calls to create a prompt
    UARTCharPut(UART0_BASE, 'n');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'T');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'x');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, ':');
    UARTCharPut(UART0_BASE, ' '); //"Enter text: "
     */

    // str[0] = 'p';
    // UARTCharPut(UART0_BASE, str[0]);


    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0); //enables ADC0 peripheral

    ROM_ADCHardwareOversampleConfigure(ADC0_BASE, 64); //API call with 64 samples to
be averaged

    ROM_ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0); //use ADC0,
sample sequencer 1
    //want the processor to trigger sequence and want to use highest priority

    ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS); //first step of adc
sequencer
    ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS); //second step
    ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS); //third step
    ROM_ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
//fourth step will
    //configure interrupt flag, sample the temperature sensor, and tell adc logic
that this
    //is the last step

    ROM_ADCSequenceEnable(ADC0_BASE, 1); //enable ADC sequencer 1

    ROM_ADCIntClear(ADC0_BASE, 1); //clear interrupt status flag
    ROM_ADCProcessorTrigger(ADC0_BASE, 1); //trigger adc conversion

    while(!ROM_ADCIntStatus(ADC0_BASE, 1, false)) //wait for conversion to finish
    {
    }

    ROM_ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value); //copies data from ...
    //the specified saple sequencer output FIFO to a buffer in memory
    ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] + 2)/4;
    //calculate the average temperature value
    ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10; //calculate temp C
    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5; //calculate temp F

    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0); //enable timer
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC); //configure timer 0

    IntEnable(INT_TIMER0A); //enable timer interrupt
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    IntMasterEnable(); //enable all interrupts

    ui32Period = (SysCtlClockGet() / 10) * 5; //0.5 ms period
    TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period -1);
    TimerEnable(TIMER0_BASE, TIMER_A);

    while (1) //infinite loop
    {
        //echoes what is types in terminal
        //if (UARTCharsAvail(UART0_BASE)) UARTCharPut(UART0_BASE,
UARTCharGet(UART0_BASE));

    }
}

void Timer0IntHandler(void)
{
    if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0); //turn
off led
    }
    else
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4); //turn on led
    }

    // Clear the timer interrupt
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

    ROM_ADCIntClear(ADC0_BASE, 1); //clear interrupt status flag
    ROM_ADCProcessorTrigger(ADC0_BASE, 1); //trigger adc conversion

    while(!ROM_ADCIntStatus(ADC0_BASE, 1, false)) //wait for conversion to finish
    {
    }

    ROM_ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value); //copies data from ...
    //the specified sample sequencer output FIFO to a buffer in memory
    ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] + 2)/4;
    //calculate the average temperature value
    ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10; //calculate temp C
    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5; //calculate temp F

    itoa(ui32TempValueF, str, 10);

    UARTCharPut(UART0_BASE, str[1]);
    UARTCharPut(UART0_BASE, str[0]);
    UARTCharPut(UART0_BASE, '\r'); //carriage return
    UARTCharPut(UART0_BASE, '\n'); //new line
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
;
}
```

Task 02:

Youtube Link:

https://youtu.be/fPus-2z6V3c

Modified Code:

```
#include <stdint.h>
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include"inc/hw_ints.h"
#include"driverlib/interrupt.h"
#include "driverlib/adc.h"
//#define TARGET_IS_BLIZZARD_RB1
#include "driverlib/rom.h"
#include "driverlib/timer.h"

volatile uint32_t ui32TempAvg; //holds temperature average value
volatile uint32_t ui32TempValueC; //holds temperature value in celsius
volatile uint32_t ui32TempValueF; //holds temperature value in fahrenheit
uint32_t ui32Period;
uint32_t ui32ADC0Value[4]; //array holds data read
char str[10];

// Implementation of itoa()
char* itoa( uint32_t num, char* str, int base)
{
    int i = 0;

    /* Handle 0 explicitely, otherwise empty string is printed for 0 */
    if (num == 0)
    {
        str[i++] = '0';
        str[i] = '\0';
        return str;
    }

    // Process individual digits
    while (num != 0)
```

```
    {
        int rem = num % base;
        str[i++] = (rem > 9)? (rem-10) + 'a' : rem + '0';
        num = num/base;
    }

    str[i] = '\0'; // Append string terminator

    return str;
}


void UARTIntHandler(void)
{
    uint32_t ui32Status;

    ui32Status = UARTIntStatus(UART0_BASE, true); //get interrupt status

    UARTIntClear(UART0_BASE, ui32Status); //clear the asserted interrupts

    while(UARTCharsAvail(UART0_BASE)) //loop while there are chars
    {
        UARTCharPutNonBlocking(UART0_BASE, UARTCharGetNonBlocking(UART0_BASE));//echo
character
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2); //blink LED
        SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //delay ~1 msec
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0); //turn off LED
    }
}

int main(void) {


    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ); //set up clock

    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0); //enable UART0 peripheral
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); //enable GPIOA peripheral

    GPIOPinConfigure(GPIO_PA0_U0RX); //configure pin as receiver
    GPIOPinConfigure(GPIO_PA1_U0TX); //configure pin as transmitter
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1); //set pintype to UART

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //initialize GPIO peripheral
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
//enable GPIO pin for output for LED

    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
UART_CONFIG_PAR_NONE));
    //Initializes parameters for UART: 114200, 8-1-N

    //    IntEnable(INT_UART0); //enable uart0 interrupt
    //   UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //enable receiver
interrupts
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
    /*
    UARTCharPut(UART0_BASE, 'E'); //calls to create a prompt
    UARTCharPut(UART0_BASE, 'n');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'T');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'x');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, ':');
    UARTCharPut(UART0_BASE, ' '); //"Enter text: "
     */

    // str[0] = 'p';
    // UARTCharPut(UART0_BASE, str[0]);


    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0); //enables ADC0 peripheral

    ROM_ADCHardwareOversampleConfigure(ADC0_BASE, 64); //API call with 64 samples to
be averaged

    ROM_ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0); //use ADC0,
sample sequencer 1
    //want the processor to trigger sequence and want to use highest priority

    ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS); //first step of adc
sequencer
    ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS); //second step
    ROM_ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS); //third step
    ROM_ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);
//fourth step will
    //configure interrupt flag, sample the temperature sensor, and tell adc logic
that this
    //is the last step

    ROM_ADCSequenceEnable(ADC0_BASE, 1); //enable ADC sequencer 1

    ROM_ADCIntClear(ADC0_BASE, 1); //clear interrupt status flag
    ROM_ADCProcessorTrigger(ADC0_BASE, 1); //trigger adc conversion

    while(!ROM_ADCIntStatus(ADC0_BASE, 1, false)) //wait for conversion to finish
    {
    }

    ROM_ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value); //copies data from ...
    //the specified saple sequencer output FIFO to a buffer in memory
    ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] + 2)/4;
    //calculate the average temperature value
    ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10; //calculate temp C
    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5; //calculate temp F
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
    /*
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0); //enable timer
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC); //configure timer 0

    IntEnable(INT_TIMER0A); //enable timer interrupt
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    IntMasterEnable(); //enable all interrupts

    ui32Period = (SysCtlClockGet() / 10) * 0.1; //0.5 s period
    TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period -1);
    TimerEnable(TIMER0_BASE, TIMER_A);
     */
    while (1) //infinite loop
    {

        ROM_ADCIntClear(ADC0_BASE, 1); //clear interrupt status flag
        ROM_ADCProcessorTrigger(ADC0_BASE, 1); //trigger adc conversion

        while(!ROM_ADCIntStatus(ADC0_BASE, 1, false)) //wait for conversion to finish
        {
        }

        ROM_ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value); //copies data from ...
        //the specified sample sequencer output FIFO to a buffer in memory
        ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] + 2)/4;
        //calculate the average temperature value
        ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10; //calculate temp
C
        ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5; //calculate temp F

        itoa(ui32TempValueF, str, 10);
        if (UARTCharGet(UART0_BASE) == 'T'){ //check temperature
            UARTCharPut(UART0_BASE, UARTCharGet(UART0_BASE));
            UARTCharPut(UART0_BASE, ' ');
            UARTCharPut(UART0_BASE, str[1]);
            UARTCharPut(UART0_BASE, str[0]);
            UARTCharPut(UART0_BASE, '\r'); //carriage return
            UARTCharPut(UART0_BASE, '\n'); //new line
        }


       if (UARTCharGet(UART0_BASE) == 'R'){
            UARTCharPut(UART0_BASE, UARTCharGet(UART0_BASE));
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 2); //turn on red led
            UARTCharPut(UART0_BASE, '\r'); //carriage return
            UARTCharPut(UART0_BASE, '\n'); //new line
        }
        if (UARTCharGet(UART0_BASE) == 'r'){
            UARTCharPut(UART0_BASE, UARTCharGet(UART0_BASE));
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0); //turn off red led
            UARTCharPut(UART0_BASE, '\r'); //carriage return
            UARTCharPut(UART0_BASE, '\n'); //new line
        }
       if (UARTCharGet(UART0_BASE) == 'B') {
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
            UARTCharPut(UART0_BASE, UARTCharGet(UART0_BASE));
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4); //turn on blue led
            UARTCharPut(UART0_BASE, '\r'); //carriage return
            UARTCharPut(UART0_BASE, '\n'); //new line
        }
       if (UARTCharGet(UART0_BASE) == 'b') {
            UARTCharPut(UART0_BASE, UARTCharGet(UART0_BASE));
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0); //turn off blue led
            UARTCharPut(UART0_BASE, '\r'); //carriage return
            UARTCharPut(UART0_BASE, '\n'); //new line
        }
        if (UARTCharGet(UART0_BASE) == 'G') {
            UARTCharPut(UART0_BASE, UARTCharGet(UART0_BASE));
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 8); //turn on green led
            UARTCharPut(UART0_BASE, '\r'); //carriage return
            UARTCharPut(UART0_BASE, '\n'); //new line
        }
        if (UARTCharGet(UART0_BASE) == 'g') {
            UARTCharPut(UART0_BASE, UARTCharGet(UART0_BASE));
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0); //turn off green led
            UARTCharPut(UART0_BASE, '\r'); //carriage return
            UARTCharPut(UART0_BASE, '\n'); //new line
        }



    }
}

/*
void Timer0IntHandler(void)
{

    if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0); //turn
off led
    }
    else
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4); //turn on led
    }

    // Clear the timer interrupt
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

    ROM_ADCIntClear(ADC0_BASE, 1); //clear interrupt status flag
    ROM_ADCProcessorTrigger(ADC0_BASE, 1); //trigger adc conversion

    while(!ROM_ADCIntStatus(ADC0_BASE, 1, false)) //wait for conversion to finish
    {
    }

    ROM_ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value); //copies data from ...
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
    //the specified sample sequencer output FIFO to a buffer in memory
    ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] + 2)/4;
    //calculate the average temperature value
    ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10; //calculate temp C
    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5; //calculate temp F

    itoa(ui32TempValueF, str, 10);
    if (UARTCharGet(UART0_BASE) == 'T'){ //check temperature
        UARTCharPut(UART0_BASE, UARTCharGet(UART0_BASE));
        UARTCharPut(UART0_BASE, str[1]);
        UARTCharPut(UART0_BASE, str[0]);
        UARTCharPut(UART0_BASE, '\r'); //carriage return
        UARTCharPut(UART0_BASE, '\n'); //new line
    }

}
*/
```
