Task 00: Execute supplied code

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"

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);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //enable GPIO port for LED
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2); //enable pin for LED PF2

    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

    IntMasterEnable(); //enable processor interrupts
    IntEnable(INT_UART0); //enable the UART interrupt
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT); //only enable RX and TX
interrupts

    UARTCharPut(UART0_BASE, 'E');
```

```
    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, ' ');

    while (1) //let interrupt handler do the UART echo function
    {
//      if (UARTCharsAvail(UART0_BASE)) UARTCharPut(UART0_BASE,
UARTCharGet(UART0_BASE));
    }

}
```

Task 01: Modify the original code to print Capital letters when small letters are entered and vice versa

```
int main(void)
{
       .
       .
       .

    int inChar; //variable
    while (1)
    {
        //if there is a char in receiver, its read, and then written to the
transmitter
        if (UARTCharsAvail(UART0_BASE))
        {
            inChar = UARTCharGet(UART0_BASE); //get char
            if(inChar >= 97 && inChar <= 122) //check lower case
            {
                inChar -= 32;
            }
            //if upper case add 32 to make lower case
            else if(inChar >= 65 && inChar <= 90)
            {
                inChar += 32;
            }
            //char on terminal
            UARTCharPut(UART0_BASE, inChar);
        }

    }

}
```

Task 02: Continuously display the temperature of the device.

```
#include "driverlib/adc.h"


int main(void)
{
    int one, ten, hundred; //variables

    //temp variables
    uint32_t ui32ADC0Value[4];      //array to store 4 values
    volatile uint32_t ui32TempAvg;      //average temp
    volatile uint32_t ui32TempValueC;   //celsius temp
    volatile uint32_t ui32TempValueF;   //fahr temp

        .
        .
        .

    //ADC sequencer
    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);

    //Configuring steps 0-2 on sequencer 1 to sample temp sensor
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);

    //Sample temp sensor and configure interrupt flag to set
    ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);

    //enable ADC
    ADCSequenceEnable(ADC0_BASE, 1);

        .
        .
        .
   while(1)
    {
    //UARTCharPut calls to create a prompt
    UARTCharPut(UART0_BASE, 'T');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'm');
    UARTCharPut(UART0_BASE, 'p');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, '(');
    UARTCharPut(UART0_BASE, 'F');
    UARTCharPut(UART0_BASE, ')');
    UARTCharPut(UART0_BASE, ':');
    UARTCharPut(UART0_BASE, ' ');

    //clear ADC interrupt and trigger sequencer
    ADCIntClear(ADC0_BASE, 1);
    ADCProcessorTrigger(ADC0_BASE, 1);

    while(!ADCIntStatus(ADC0_BASE, 1, false))
    {
```

```
    }

    SysCtlDelay(5000000);

    //ADC value from ADC Sampler Sequence
    ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);

    //Average of temp sensor data
    ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] + 2)/4;

    //calculate celsius value of temp
    ui32TempValueC = (1475- ((2475*ui32TempAvg)) / 4096) /10;

    //calculate Fahrenheit temp
    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;

    //get values
    hundred = ui32TempValueF / 100;
    ten = ui32TempValueF % 100 / 10;
    one = ui32TempValueF % 100 % 10;

    //convert values to ASCII
    hundred += 48;
    ten += 48;
    one += 48;
    //if its 0 enter number else make space
    if(hundred > '0')
    {
        UARTCharPut(UART0_BASE, hundred);
    }
    else
    {
        UARTCharPut(UART0_BASE, ' ');
    }

    //send ten and one
    UARTCharPut(UART0_BASE, ten);
    UARTCharPut(UART0_BASE, one);
    UARTCharPut(UART0_BASE, '\n');
    UARTCharPut(UART0_BASE, '\r');
   }
}
```