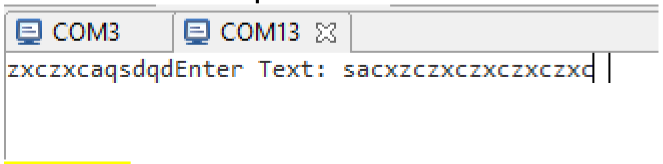
Ron Joshua Recrio
Github root directory: https://github.com/recrio/submissions.2

Date Submitted: **12/13/2019**

Task 00: Execute provided code


COM3 COM13

```
zxczxcaqsdqdEnter Text: sacxzczxczxczxczxc
```

Task 01:

Youtube link:
https://youtu.be/cWX2zC-QPf8
Modified Code:

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


//global variables
    uint32_t ui32ADC0Value[4];
    volatile uint32_t ui32TempAvg;
    volatile uint32_t ui32TempValueC;
    volatile uint32_t ui32TempValueF;
    char str_temp[10]; // variable used to store temp value in string

// given print function
void print_string(char * str) {
    while(*str != '\0')
    {
        UARTCharPut(UART0_BASE,*str);
        ++str;
    }
}


//reverses characters in char array
void reverse(char str[], int len)
{
    int start, end;
    char temp;
    for(start=0, end=len-1; start < end; start++, end--) {
        temp = *(str+start);
        *(str+start) = *(str+end);
        *(str+end) = temp;
    }
}


// useful conversion function
char* itoa(int num, char* str, int base)
{
    int i = 0;
    bool isNegative = false;
    if (num == 0) {
        str[i] = '0';
        str[i + 1] = '\0';
        return str;
    }

    if (num < 0 && base == 10) {
        isNegative = true;
        num = -num;
    }

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

    if (isNegative){
        str[i++] = '-';
    }

    str[i] = '\0';
    reverse(str, i);
    return str;
}



int main(void) {
      // Set clock
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);

      // Enable ADC and UART
      SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

      // Enable Timer and GPIO
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

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

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

    //Configure ADC
    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0); // using ADC sample
sequencer 1 (SS1), set as the highest priority, and processor will trigger ADC
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS); // ADC sample step 0
    ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS); // ADC sample step 1
    ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS); // ADC sample step 2
    ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END); //ADC
sample step 3, set ADC interrupt flag, end sampling
    ADCSequenceEnable(ADC0_BASE, 1);    // enable ADC0

    //Configure Interrupts
    IntMasterEnable(); //enable processor interrupts
    IntEnable(INT_TIMER1A); //enables timer1A interrupt in the interrupt vector table
    TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT); //interrupt is triggered at
TIMEOUT of timer1A

    //Configure Timer1
    TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
    TimerEnable(TIMER1_BASE, TIMER_A);
    TimerLoadSet(TIMER1_BASE, TIMER_A, SysCtlClockGet()/2);


    while (1)
    {
        //wait for interrupt
    }

}


void Timer1IntHandler(void) {
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);

    ADCIntClear(ADC0_BASE, 1);
    ADCProcessorTrigger(ADC0_BASE, 1);

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

    ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);

    // Calculations
    ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
ui32ADC0Value[3] + 2)/4;
    ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;

       // Conversion using itoa function
    print_string(itoa(ui32TempValueF, str_temp, 10));
    print_string("\r\n"); //carriage return and line feed to current and previous
temp values
}
```

Task 02:

Youtube Link:

```
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/debug.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\uart.h"
#include "\ti\tivaware_c_series_2_1_4_178\inc\tm4c123gh6pm.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\adc.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\debug.h"
#include "\ti\tivaware_c_series_2_1_4_178\driverlib\interrupt.h"

#ifdef DEBUG
void__error__(char *pcFilename, uint32_t ui32Line)
{
}
#endif

void outputN(uint32_t);
void outputC(char data);
void UARTIntHandler(void);

uint32_t ui32ADC0Value[1];
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;

int main(void) {
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
//temp

    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    ADCHardwareOversampleConfigure(ADC0_BASE, 32);
    ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0); //temp
    ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);
//temp

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

    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
    // Enable interrupts
    IntMasterEnable();
    IntEnable(INT_UART0);
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);

    ADCSequenceEnable(ADC0_BASE, 3);
    ADCIntEnable(ADC0_BASE, 3);
    // Wait for input forever
    while (1)
    {
    }
}

void outputN(uint32_t n) {
    if (n >= 10) {
        outputN(n / 10);
        n = n % 10;
    }
    outputC(n + '0');
}

void outputC(char data) {
    while ((UART0_FR_R&UART_FR_TXFF) != 0);
    UART0_DR_R = data;
}

void UARTIntHandler(void)
{
    uint32_t ui32Status;
    ui32Status = UARTIntStatus(UART0_BASE, true); //get interrupt status
    UARTIntClear(UART0_BASE, ui32Status); //clear the asserted interrupts

    switch (UARTCharGet(UART0_BASE)) {
    case 'B':
        UARTCharPut(UART0_BASE, 'B');
        UARTCharPut(UART0_BASE, ' ');
        UARTCharPut(UART0_BASE, 'O');
        UARTCharPut(UART0_BASE, 'n');
        UARTCharPut(UART0_BASE, '\n');
        UARTCharPut(UART0_BASE, '\r');

        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2); //LED on
        SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //software debounce
        break;
```

```
case 'b':
    UARTCharPut(UART0_BASE, 'B');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'O');
    UARTCharPut(UART0_BASE, 'f');
    UARTCharPut(UART0_BASE, 'f');
    UARTCharPut(UART0_BASE, '\n');
    UARTCharPut(UART0_BASE, '\r');

    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0); //LED off
    SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //software debounce
    break;
case 'R':
    UARTCharPut(UART0_BASE, 'R');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'O');
    UARTCharPut(UART0_BASE, 'n');
    UARTCharPut(UART0_BASE, '\n');
    UARTCharPut(UART0_BASE, '\r');

    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1); //LED on
    SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //software debounce
    break;
case 'r':
    UARTCharPut(UART0_BASE, 'R');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'O');
    UARTCharPut(UART0_BASE, 'f');
    UARTCharPut(UART0_BASE, 'f');
    UARTCharPut(UART0_BASE, '\n');
    UARTCharPut(UART0_BASE, '\r');

    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0); //LED Off
    SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //software debounce
    break;
case 'G':
    UARTCharPut(UART0_BASE, 'G');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'O');
    UARTCharPut(UART0_BASE, 'n');
    UARTCharPut(UART0_BASE, '\n');
    UARTCharPut(UART0_BASE, '\r');

    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GPIO_PIN_3); //LED On
    SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //software debounce
    break;
case 'g':
    UARTCharPut(UART0_BASE, 'G');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'O');
    UARTCharPut(UART0_BASE, 'f');
    UARTCharPut(UART0_BASE, 'f');
    UARTCharPut(UART0_BASE, '\n');
    UARTCharPut(UART0_BASE, '\r');
```

```
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0); //LED Off
            SysCtlDelay(SysCtlClockGet() / (1000 * 3)); //software debounce

            break;
        case 'T':
            UARTCharPut(UART0_BASE, 'T');
            UARTCharPut(UART0_BASE, ':');
            UARTCharPut(UART0_BASE, ' ');

            ADCIntClear(ADC0_BASE, 3);
            ADCProcessorTrigger(ADC0_BASE, 3);
            // While not done
            while (!ADCIntStatus(ADC0_BASE, 3, false))
            {
            }
            // Take values
            ADCSequenceDataGet(ADC0_BASE, 3, ui32ADC0Value);
            ui32TempValueC = (1475 - ((2475 * ui32ADC0Value[0])) / 4096) / 10;
            ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
            // Output numbers
            outputN(ui32TempValueF);
            UARTCharPut(UART0_BASE, '\n');
            UARTCharPut(UART0_BASE, '\r');
            break;
        }

    }
```

Youtube Link:
Modified Code: